Department of Production & Management Engineering
Technical University of Crete

École doctoral décision informatique mathématiques organization
Université Paris Dauphine

An Agent-based Workflow Management System for Marketing Decision Support

by

Pavlos Delias

Submitted for the partial fulfillment of the Requirements for the degree of

Doctor of Philosophy

September 2009

Declarations

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Parts of the thesis have been published in academic journals or conference proceedings. Please cite as appropriate when referring to this text.

The thesis is approved by:

1. Nikolaos Matsatsinis (*advisorin Technical University of Crete)*
2. Alexis Tsoukiás *(advisor in Université Paris-Dauphine)*
3. Athanasios Mygdalas
4. *Yannis Siskos*
5. *Constantine Tsouros*
6. *Evangelos Grigoroudis*
7. *Yannis Marinakis*

Acknowledgements

This work is part of the 03ED375 research project, implemented within the framework of the "Reinforcement Programme of Human Research Manpower" (PENED) and co-financed by National and Community Funds (75% from E.U.- European Social Fund and 25% from the Greek Ministry of Development-General Secretariat of Research and Technology).

Contents

List of Abbreviations ix

List of Figures x

List of Tables xi

Short Vitae xii

1 Introduction 2

1.1 Practical and Theoretical Value 2

1.2 Motivation and Major Assumptions 3

1.3 Thesis Structure 4

2 State of the Art 7

2.1 Background 7

2.2 Research Agenda 8

2.2.1 Trends and Standards 8

2.2.2 Specifying the Requirements of a WFMS 9

2.2.3 Limitations of Existing Systems 9

2.3 The advantages of using an agent approach 11

2.4 Workflow Taxonomy 13

2.4.1 Classification Approaches 13

2.4.2 Agent Related Classification Approaches in WFMS 14

2.4.3 Rallying Agents and Web Services to Manage Workflows 15

2.4.4 Workflow Agents under the Grid Umbrella 16

3 A Functional Classification Scheme for Agent-involved Workflow Management Systems 17

3.1 The All-embracing Mentality 18

3.2 Scheme Presentation 20

3.2.1 Process Definition Tools Component 20

3.2.2 Workflow Client Applications Interface 20

3.2.3 Invoked Applications Interface 21

3.2.4 Other Enactment Services Component (Workflow Interoperability Interface) 22

3.2.5 Administration and Monitoring Tools Component.................................... 22

3.2.6 Workflow Enactment Service Component 23

3.3 How agents are used? (A survey of the Related Literature) 24

3.3.1 Process Definition Tools Component 24

3.3.2 Workflow Client Applications Interface 27

3.3.3 Invoked Applications Interface 29

3.3.4 Other Enactment Services Component (Workflow Interoperability Interface) 31

3.3.5 Administration and Monitoring Tools Component.................................... 33

3.3.6 Workflow Enactment Service Component 34

3.4 Overall Metrics 40

4 Design and Implementation...42

4.1 Pivot Processes 43

4.1.1 Direct Mail Campaign Automation... 44

4.1.1.1 Key actors involved 46

4.1.2 Customer Contact Center Management 47

4.1.2.1 Key actors involved 48

4.2 The WADE platform.. 49

4.3 Agents Communication Support 52

4.3.1 Interaction Protocols 52

4.3.2 Joined Interaction Protocols 54

4.3.3 Unspecified Interactions following a workflow logic 55

4.4 Business Logic Support 58

4.4.1 Rely on the Workflow Definition 59

4.4.1.1 Importing an XPDL document 59
4.4.1.2 Construct a JAVA class containing the definition 61
4.4.2 Use an Application Engine and an application specific ontology 64
4.4.3 Business logic support using both methods in combination 69
4.5 Manual Intervention 70
4.6 Statefulness through Document-Centric Stigmergy 73
4.6.1 A supportive database schema 75
4.7 Process Monitoring & Auditing 79
4.7.1 Why is it important? 79
4.7.2 Implementing the monitoring component as a kernel service 79
4.7.3 Benefits and Cost 83
5 Results 85
5.1 The Graphical User Interface 86
5.1.1 Starting the application 86
5.1.2 Platform related actions 87
5.1.3 Workflow related actions 89
5.1.4 Application configuration and management related actions 92
5.1.5 Other actions 93
5.2 Evaluate the Prototype against the Classification Criteria 94
5.2.1 Process Definition Tools 94
5.2.1.1 Analyze, model, compose, describe, and document a Business Process 94
5.2.1.2 Process Definition Write / Edit 94
5.2.1.3 Definition retrieval 94
5.2.2 Workflow Client Applications 94
5.2.2.1 Worklist Handling 94
5.2.2.2 Process control 94
5.2.2.3 Data Handling 94
5.2.2.4 User Interface 95

5.2.3 Invoked Applications 95
5.2.3.1 Worklist Handling 95
5.2.3.2 Process Control 95
5.2.3.3 Data Handling 95
5.2.3.4 Service Discovery 95
5.2.4 Workflow Interoperability 95
5.2.4.1 Common Interpretation of Process Definition 95
5.2.4.2 Workflow Data Interchange 96
5.2.5 Administration and Monitoring Tools 96
5.2.5.1 User / Role Management 96
5.2.5.2 Audit Management 96
5.2.5.3 Resource Control 96
5.2.5.4 Process Monitoring 96
5.2.6 Workflow Enactment Service 96
5.2.6.1 Runtime Control Environment 96
5.2.6.2 Definition Interpretation 97
5.2.6.3 Execution of Tasks 97
5.2.6.4 Scheduling 97
5.2.6.5 Data Functions 97
5.2.6.6 Task Assignment 97
5.2.6.7 Resource Allocation 97
5.3 Exploiting the Prototype to Deploy Algorithms. The Case of a Scheduling Algorithm. 98
5.3.1 The algorithm's context and similar works 98
5.3.2 The resource allocation decision 101
5.3.3 Optimization Criteria 102
5.3.4 The scheduling algorithm 104
5.3.4.1 Expressing the optimization metric with a matrix representation 104

5.3.4.2 Optimization in the Continuous Domain 106

5.3.4.3 Discrete approximation of the results 107

5.3.5 Evaluating the algorithm's performance 108

5.3.5.1 Defining parameters for the system's load condition 108

5.3.5.2 Efficiency criteria for evaluating the execution case 110

5.3.5.3 Efficiency criteria for evaluating the design case 111

5.3.6 Experimental Results 112

5.3.6.1 Testing the algorithm under different load conditions 113

5.3.6.2 Comparing the proposed algorithm with other approaches 116

6 Conclusions 119

6.1 Future Work 121

List of Abbreviations

WF	Workflow
WFMS	Workflow Management Systems
WfMC	Workflow Management Coalition
AWfMS	Agent-involved Workflow Management Systems
GUI	Graphical User Interface
WS	Web Service
ICT	Information and Communication Technology
CSCW	Computer Supported Cooperative Work
API	Application Programming Interface
FIPA	Foundation for Intelligent Physical Agents
ACL	Agent Communication Language
ECA	Event-Condition-Action
PN	Petri Net
BDI	Belief-Desire-Intention
RPC	Remote Procedure Call
ORB	Object Request Brokers
XML	Extensible Markup Language
XPDL	XML Process Definition Language
HTML	HyperText Markup Language
WWW	World Wide Web
BPEL4WS	Business Process Execution Language for Web Services
UDDI	Universal Description, Discovery and Integration
AI	Artificial Intelligence
SLA	Service Level Agreement
AMS	Agent Management System
DF	Directory Facilitator
CFA	Configuration Agent
CA	Controller Agent
IP	Interaction Protocol
FSM	Finite State Machine
UML	Unified Modeling Language

List of Figures

Figure 1 Workflow Reference Model - Components & Interfaces. source WfMC [66]. 19
Figure 2 The proposed classification scheme...19
Figure 3 Distribution of the reviewed publications according to their type. 41
Figure 4 Chronological distribution of the reviewed publications 41
Figure 5 Basic steps in developing effective communications (source: [3, p. 541])...............45
Figure 6 Basic phases of the contact center management process 48
Figure 7 The WADE-based application concept 49
Figure 8 The WADE Architecture...51
Figure 9 The workflow of the `SolicitDesign` class 53
Figure 10 The contract net protocol implemented during an instance of the `SolicitiDesign` workflow 54
Figure 11 Join two interaction protocols during one process 55
Figure 12 The ReviewDrafts workflow diagram..56
Figure 13 Main interactions within a sample instance of the ReviewDrafts workflow process...57
Figure 14 Mapping an ad-hoc message exchange pattern to a workflow class......................58
Figure 15 Workflow diagram of the PreparePiece process..63
Figure 16 Class Diagram for the PreparePiece process and related tools 64
Figure 17 The Contact Center Ontology 67
Figure 18 Messages exchanged during the ontology-based workflow execution *(Source: Application runtime – JADE Sniffer Agent).* 68
Figure 19 The proposed database schema. 76
Figure 20 A database schema which does not exploit application's features. 78
Figure 21 Class Diagram of the monitoring package 81
Figure 22 Basic behaviour of the monitoringWF service 83
Figure 23 The application's starting screen..87
Figure 24 Starting the multi agent platform and providing domain information.................88
Figure 25 The Platform pane after the initialization of the platform with a specific configuration...89
Figure 26 The workflow pane...90
Figure 27 Choosing to continue an existing instance 91
Figure 28 Checking instance's requirements..91

Figure 29 Providing workflow parameters 92
Figure 30 Editing the configurations' files. 93
Figure 31 The basic steps of spectral clustering...104
Figure 32 The waste factor versus granularity for different values of the low bound *B*. 114
Figure 33 The waste factor versus the number of pending tasks for three different granularity values 115
Figure 34 The algorithm's efficiency versus granularity when different number of iteration are used in the k-means descritization phase 116
Figure 35 Comparison of different algorithms when the tasks' load augments. The granularity is fixed to 0.1% 116
Figure 36 Comparison of the waste factor versus granularity for different algorithms for *B=5* (left) and *B=10* (right) 117
Figure 37 Tasks' overlapping versus the number of agents for different algorithms..........118

List of Tables

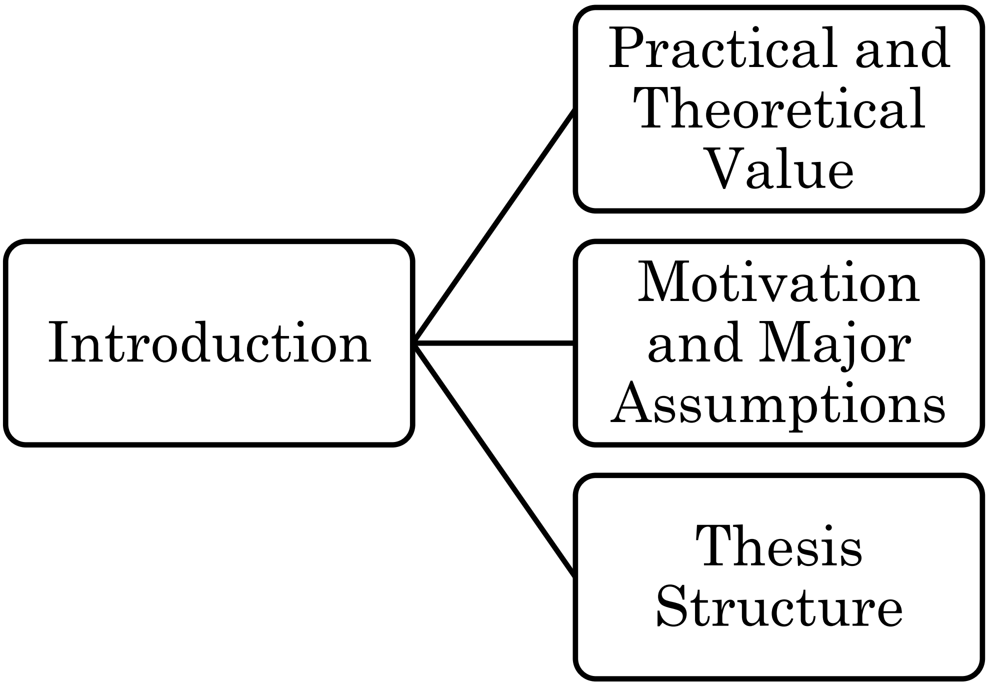
Table 1 Classification of the existing literature in AWfMS. 40
Table 2 SAP Business Workflow in Campaign Automation. *Source: http://help.sap.com/ saphelp_crm70/helpdata/EN/45/cbced6f771fae10000000a1553f6/content.htm* 45
Table 3 Importing a XPDL definition 60

Short Vitae

Pavlos Delias received his diploma in Production Engineering & Management from Technical University of Crete in 2002. He received his Master Degree from the same university in Management Engineering in 2005, and the next year was registered as a PhD candidate under the cotutelle framework with Technical University of Crete and Université Paris Dauphine.

He was awarded a scholarship from the Hellenic Foundation of Scholarships (IKY) for his Master while his PhD is founded by the general secretary or research and technology of the Hellenic ministry of Development. He is a research assistant in projects concerning decision making and information technologies. Currently he is with the Decision Support Systems Laboratory of the Technical University of Crete, Greece.

CHAPTER 1


Practical and Theoretical Value
Introduction
Motivation and Major Assumptions
Thesis Structure

1 Introduction

Workflow Management Systems (WFMS) are systems that define, create and manage the execution of workflows through the use of software, interactions with workflow participants and, where required, invocations of information technology tools and applications [1]. They are typically used in organizations to provide administrative and supervisory functions. On the other hand, software agents come along with a plentiful terminology including agent architectures, multi-agent system architectures, agent frameworks, and agent infrastructures [2].

This thesis focuses on examining the integration of these two fields, revealing the stimulation and the advantages of such a mixing. In particular, thesis' overall goal is to clear the vague picture of the consolidation of workflow management systems and software agents and to provide a unifying framework for this intersected area.

In order to better demonstrate the results of elaborating on the unifying framework, *marketing* was selected as the application domain. Marketing processes intrinsically fit the workflow management concept because they are far more flexible and versatile than production processes. In marketing domain [3], it is common for the process flows not to be rigidly defined, heterogeneous resources to be involved, and high customization per customer to be required. However, the regular activities required to carry out a marketing process (e.g., writing a report, extracting data from databases, organizing campaigns, schedule meetings, etc.) have good potentials to be monitored by information systems. To such a context, automation prospects are significant and the application of workflow logic has noteworthy contribution potentials. Although the focus is on the marketing field, thesis' contributions are domain-abstract, i.e., they can be applied in general to any business domain that requires the implementation of workflow logic.

1.1 Practical and Theoretical Value

Thesis' contribution is threefold. The first part concerns an extensive literature review and a classification of existing works according to a pioneering classification scheme. The proposed scheme exploits popular standards of the field in an attempt to catalog what software agents can do in workflow management systems. Such tabulation is unique in the literature as it is not just a simple summary of the sources, but it also has an organizational pattern and combines both summary and synthesis. It gives a new interpretation of existing material and it opens a new way to criticize works in the field. The meticulous survey of the intersected area of Workflow Management Systems and software agents, which is presented in this thesis, provides a handy guide to the topic. It also provides a solid background for researchers that would like to direct their research efforts at the field.

The second part refers to the design and development of a prototype workflow management system utilizing the agent paradigm. Based on an open source platform [4], the prototype demonstrates how a number of workflow management functions can benefit from multi-agent systems' features. The development of a prototype is a valuable apparatus to validate the unifying framework in the sense that it helps reveal possible problem areas and provides new insights of the envisioned field. Since an analytical documentation of the developed software is attached, the prototype may operate as a practical basis for developers, should they need to re-use its components. Besides this practical convenience, and the potentials of using the prototype as a ready-to-use workflow management platform, the developed system can operate as a test-bed to test specific algorithms or/and provide the general context to test the integration of supplementary modules and services.

In fact, exploiting the prototype as a test-bed for specific algorithms is the matter of the third contribution of the thesis. Considering the specific marketing business processes that were elaborated, and the modus operandi of the multi-agent platform, a compelling scheduling algorithm is proposed. The algorithm exploits concepts of the generalized eigenvalue analysis to optimize a scheduling problem in tandem with resource allocation issues. The algorithm is integrated in a particular business process, nevertheless, to test algorithm's efficiency, and to compare it with other approaches many experiments were conducted beyond the prototype's scope. Hence, the algorithm is serviceable as a distinct unit, and it can be used outside the workflow context as well, as long as the modeling themes are valid.

1.2 Motivation and Major Assumptions

Workflow Management Systems (WFMS) and software agents are both established areas in research and in business environments as well. The former is a category of business information systems, emerging to provide automation solutions, while the latter supplies the information systems field with a serviceable paradigm. These two disciplines (WFMS & agents) can be combined to produce effective tools; they can be joined to ameliorate each other's niches. Indeed, such attempts exist in the literature as this thesis exhaustively presents. Yet in these works, it is hard to distinguish a unifying background which would be able to clarify the overall picture, make the researchers' contributions more identifiable and provide a solid basis for future advancements.

More specifically, so far, when considering joining the two disciplines, there were no justified answers (positive or negative) of generic validity to the question "Does it worth to mix WFMS and agents?" Hopefully, the text that follows in the next chapters, it can reply to this question. Without claiming that the agents' paradigm is the most suitable to be applied in WFMS, this thesis puts on display the cases in which the blending of the two areas seems promising. A major endeavor was to suggest a method to criticize works of the field, so that involvement of the agents in WFMS is justified and relative research is stimulated.

Eventually, this endeavor proved to be exceptionally broad as it cuts a generous swath across many fields: workflow standards, terminology and glossary, process modeling languages, workflow enactment services, human interactions, applications integration, system architectures, implementation approaches, operational facilities, optimization algorithms, multi-agent systems design, etc. Thus, in order to narrow this broad spectrum, a critical assumption of this work is that the workflow management systems field is described by the definitions of the Workflow Management Coalition[1] (WfMC). The WfMC 's terminology and glossary [1] are adopted throughout this text, leaving outside the scope of the thesis the debate about what a workflow management system is.

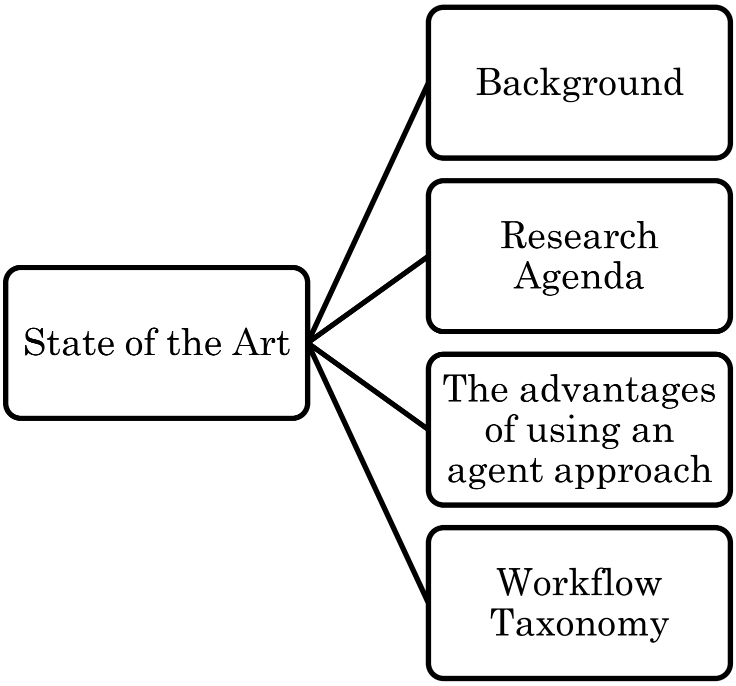
1.3 Thesis Structure

This chapter provides a general overview of the problem and discloses the motivation to research in the topic. The second chapter describes the general background and the mainstream research efforts. In the third chapter, the classification scheme is introduced and an extended survey matches existing works against the proposed criteria. The fourth chapter explains the design and implementation concepts of the prototype system that was developed, while the general results are presented in chapter five. The results refer to the presentation of the actual software tool that it was

[1] www.wfmc.org

developed and to the presentation of the scheduling algorithm as well. The documentation of the source code of the tool is attached as an appendix. Finally, the conclusions' chapter discusses the implications of the results and concludes the thesis.

CHAPTER 2



State of the Art
Background
Research Agenda
The advantages of using an agent approach
Workflow Taxonomy

2 State of the Art

2.1 Background

Workflow Management Systems (WFMS) emerged in the Information Systems landscape as a promising office information systems technology at the 70s. During the 80s, they have evolved into enactment machines of operational models. Their critical feature of that time was that they were too rigid to support the integration of human activities. This essential requirement advantaged the development of systems that could support collaborative work. Singh and Huhns [5] support that "Workflows have been with us from the dawn of time" and sectionalize the systems into five generations: Starting from the "*manual*" ones which were a side-effect of bureaucracy, they continue with the "*closed*" ones that focused mainly on data processing and on the automation of the existing manual activities. The third generation concerned the "*database-centric*" systems. It was then when data and process appeared to decouple themselves. The next generation refers to the current situation. This generation's systems provide the separation of control from the application. Finally, Singh and Hunhs predict that the next generation will incorporate agent-based systems.

Abott and Sarin [6] provide a different taxonomy of the WFMS. They name as the "*first generation*" systems the systems that were "*application-specific*". Those systems were tightly related to specific functions (e.g., document management) and they were closed and proprietary. During the second generation, the workflow logic is separated from the application one, while the integration of third-party tools becomes available. Current situation is mapped on the third generation: Contemporary WFMS provide access to other applications through APIs and they integrate third-party tools as well. They adopt standards-based architectures and they become far more user-friendly. Abott and Sarin's prediction for the next generation describes a ubiquitous environment, interchange of data and control is the focal event.

Sheth and his colleagues [7] illustrated the evolution of the WF runtime system architectures. Starting from centralized / one-engine early systems, the architectures evolved to more distributed ones, including web-orientation and mobile-agents enhancements. As depicted in [7] the evolution will continue by supporting organic processes. In [8] a very explanatory figure demonstrating the history of automation and workflow systems is provided.

Concluding, it is evident that the WFMS development keeps pace with the technological evolution. Eventually, WFMS will make progress towards more open and ubiquitous environments. As WFMS evolve, they reveal their interdisciplinary nature and researchers are becoming more aware of it.

2.2 Research Agenda

2.2.1 Trends and Standards

The term "*workflow*" (*WF*) is overloaded to the point where it is hard to distinguish what a WFMS is meant to achieve. This happens mainly, because there is a variety of scenaria where workflow technology is applied: diverging from *Human WF* to *Document Management*, *Business Rule-Driven WF*, *ISO certification claim*, *Process Controlling*, *Composite WF for Service Oriented Architectures*, *Groupware*, *Grid Computing*, *Enterprise Application Integration*, just to name a few.

Due to its interdisciplinary nature, workflow research cuts a generous swath across many fields. Storh et al. [9] propose to classify the active research efforts into 3 categories: *Technical* issues, *Management and organizational* issues, and *Market, Economic and Social* issues. Li et al. [10] discern two trends in current workflow research community. One trend embraces the *Web Services* (*WS*) paradigm and strives to develop *WS*-related architectures and methodologies (Choreography, Orchestration, Process Definition Interchange, Service Discovery, Messaging, Transports, Interoperability, Security). The other focuses on overcoming the limitations of traditional workflow management concerning adaptability and flexibility.

The interdisciplinary nature of workflow also led to a rather vexing effect: a bold confusion in the *WF*-related standards. One can refer to [11, 12] and to pages 118-138 of [8] for a discussion on the topic. Beyond any doubt, significant progress has been done in the field, *Workflow Management Coalition* [2] (*WfMC*) acting as a vital catalyst. Nevertheless, declaring my personal opinion, I share the view that as workflow standards are still evolving, and as existing workflow systems support their own

[2] www.wfmc.org

proprietary technologies, it will take some time for any standards to be settled down as a global accepted reference [13].

2.2.2 Specifying the Requirements of a WFMS

WFMS are currently an active field of enterprise information systems. WfMC [14] estimates that there are over 200 commercial WFMS and that hundreds of companies integrate WFMS into their information and communication (ICT) infrastructure. Besides the fundamental specifications of a WFMS (the description of which is beyond the scope of this thesis), there are some functional requirements that could put added value:

- WFMS should find a way to manage the dynamic nature of business processes. As business processes become more volatile, and as they start crossing the organization's boundaries, their interactions need a rather sophisticated supervisor.
- Within business processes, many tasks are interrelated; responsibilities and data are distributed [15, 16]. This natural concurrency demands efficient techniques for task assignment, resource allocation and scheduling. Moreover, in the case of multiple service providers, the WFMS should be able to semantically discover the appropriate service providers; negotiate with them, and finally allocate them the work.
- Failures and exceptions must be tackled adaptively and efficiently.
- Contemporary WFMS must be able to operate in a pervasive computing environment. They should be able to integrate external applications, other WFMS, heterogeneous devices and legacy systems.
- Operating in the web appears a sine qua non requirement, while supporting the users with friendly and customizable interface would promote their application.
- Scalability, security, and reliability always remain critical requirements.

2.2.3 Limitations of Existing Systems

Considering the above requirements, many researchers have exposed the limitations of existing systems [16-24]:

- WFMS lack of adaptability: most of them require an a priori representation of a business process and all potential deviations from that process [20]. They suffer from disadvantages such as not supporting the dynamic incorporation/modification of process models, poor adaptability of process models at runtime, and they are incapable of integrating distributed process models [25]. The static workflow definition and its passive interpretation does not allow WFMS to demonstrate flexible behavior and to deal with real-life situations, such as fast changing customer requirements and enterprise goal shifts [22, 26].

- They are unable to cope with dynamic changes in resource levels and task availability, as they tend to lack the necessary tools to redistribute work items automatically as and when required [18]. WFMS lack of resources management facilities [18, 20, 23]. They focus on the administration of processes and they pay less or even hardly any attention to the problems such as the resource allocation and the resource restriction [27]. Resource conflicts are seldom monitored as WFMS tend to manage independently resources in an organization. This kind of conflicts may lead to wasteful architectures and to declined quality of service, while it becomes even more critical in the case of cross-organizational workflows. In addition, tasks are associated with users (actors) rather than roles [17]. Role management is a feature that still does not exist in many systems. In general, limited or non-existing optimization features (e.g., scheduling, resource allocation etc.) are incorporated.
- Authors of [2, 20, 28-30] noticed very early that semantics is a feature that can lift up workflow functionality and that existing systems lack of them. Through the use of semantics the decisions will be further automated; negotiation among actors will be enabled; optimization of processes and learning features will be disposable, and compensation activities will have a formal basis to lie on. Unfortunately, the use of semantics is still in infantile level of integration in existing WFMS.
- WFMS can not respond in a reactive way to exceptions that may occur during the execution of a process instance, and their exception handling is rather inadequate [18, 19].
- WFMS operate in splendid isolation and they represent islands of automation that provide inflexible tactical solutions [21]. They lack of heterogeneity [20] and they have poor support of interoperability [31]. Although WfMC strives to establish generic interfaces and to enable interoperability, when WFMS need to exchange data they use proprietary APIs calls [23]. This fact limits significantly their extensibility [16].

- Existing WFMS tend to be centralized while their runtime components are based on the client-server model [32]. Relying on a single central control does not allow systems to support reliable and consistent process execution with acceptable failure resiliency, performance, and scalability. Additionally, existing WFMS have a weak support of correctness and reliability [31].

2.3 The advantages of using an agent approach

Without any doubt, there is no single solution for all the WFMS problems and limitations. Moreover, the decomposition of workflows into agent-oriented architectures does not seem an appropriate solution at first sight, since workflows are intrinsically addressed by procedural programs. Therefore, an additional challenge of building agent-oriented workflow architectures lies in providing abstractions that maintain an explicit representation of the control flow and of the global workflow behavior. Yet, *software agents* constitute an attractive metaphor with significant potentials to advance the *WF* development. In [33], Lange and Oshima promote the use of mobile agents in the distributed systems field by demonstrating seven arguments. In the same paper, they present a few application areas where the agents' paradigm could flourish (workflow is indeed included). This section supports this claim by providing some extra justifications.

First of all, agents inherit three powerful characteristics from their object-oriented nature: encapsulation, inheritance and polymorphism. This way, agents allow workflow developers to customize WF objects through subclassing (for example, add a new role by appending extra properties), and improve *WF* features through aggregation. Through polymorphism, agents allow to mix and match existing features, dynamically add new features, and adjust the system architecture to a particular domain more easily than any procedural program.

In addition, mobility infuses agents with the ability of migration. This potential allows one to decentralize a WFMS [34] and exploit the benefits of both distributed WFMS [31, 35, 36] and of the agents paradigm in distributed systems [33]. By their nature, agents support heterogeneity. Using an abstract communication and coordination level, agents can be incorporated into the varying hardware and operating systems architectures that dwell in a business process [34]. This enhanced coordination ability allow agents to act as configuration facilitators [37, 38] and advances them as a promising technology for application integration [39]. Agents modular nature can provide highly reusable workflow architectures [40] which not only are an alternative technology to existing workflow systems but most importantly, they also offer an alternative vision of how organizations can be structured and managed [20].

Agents (being autonomous) can relief *WF* engines from some computation load. Consequently the engines' workloads shall be reduced favoring significantly WFMS scalability [41]. They enable the recovery process as they are stateful entities, contributing significantly to the fault tolerance of the system. The encapsulation of state also supports the asynchronous execution of a business process, a popular case when human participants are involved [34].

As a more general contribution, it shall be noticed that the agent paradigm supports the vision of human substitution: the inherent autonomy of software agents can fulfill activities on behalf of a human with an expected quality of service. Another core feature of agents, *reactivity*, provides them with an intrinsic capability to adapt to dynamic changes in the environment [40]. Actions do not need to be rigidly prescribed, since agents can anticipate their environment timely as well as efficiently respond to the changes that occur [16, 42].

Besides reactivity, pro-activeness can boost agents' intelligence. Agents can adopt feedback mechanisms to guide themselves during future actions [16]. They can implement intelligent decision-making techniques such as negotiation [15], semantics [23, 43, 44], and planning [25, 45]. Moreover, agents are able to perform optimization tasks as routing and scheduling [41, 46], task assignment [47], resource allocation [17]. In [27], Qiu et al. advocate that problems such as resource collision and low efficiency of resource utilization can not be readily addressed unless agents join the system.

Nevertheless, designing an agent-based system is far more complicated than relying on a traditional WFMS. One shall always balance the trade-off between design and development complexity and efficiency and effectiveness. A list of cases when the agent paradigm appears to be an eminently suitable technology for workflow management is provided below:

- Process definitions can not describe entirely the problem solution [15], or can not predict all possible paths of the process execution.
- Interactions among tasks and/or participants are fairly sophisticated [15], or tasks themselves are rather complex.

- The processes comprise rich social interactions among the workflow participants.
- Applications that are modular, decentralized, and changeable [48].
- The environment demands asynchronous communication [49].
- The environment is radically heterogeneous.
- The applications call for extensive human participants integration [34], or imply long tasks.
- An explicit organizational structure (with analytical description of job roles and responsibilities per role) exists.

2.4 Workflow Taxonomy

2.4.1 Classification Approaches

It is hard to define the term workflow because it is an extremely broad concept. In a previous section (2.2.1) just a few of its flavors were mentioned; one of course can find in the literature a lot of additional applications. This variety is obviously inherited to the WFMS as well. McCready [50] was the first that tried to shed a light to the confusing field of WFMS, classifying them into three categories: administrative, production, and ad hoc systems. Georgakopoulos et al. [31] noticed that the dimensions along which WFMS are classified are:

- repetitiveness and predictability of workflows and tasks
- how the workflow is initiated and controlled, i.e., from human-controlled to automated
- requirements for WFMS functionality

Stohr and Zhao [9] place these three categories along a flexibility axis; production systems being the most rigid and specific and ad-hoc systems being the most flexible ones. Leymann and Roller [51] introduce a new category of WFMS: the collaborative ones. They plot WFMS on a two-axis area: *Business value* and *Repetition*. Van der Aalst [52] uses two different axes: the centricity one (systems can be either information-centric, either process centric) and the structure one (loose or tight). He distinguishes the WFMS into collaborative, adaptive, and production.

Georgakopoulos et al. [31] characterize the WFMS by a single criterion: human engagement. They use human-oriented systems (computer supported cooperative work - CSCW) on the one side and system-oriented (Transaction Processing Systems) on the other. Nutt [53] by his turn, refines the CSCW characterization along three axes: *Coordination support, Computation support,* and *Logical immersion*. Verginadis [54] proposes a classification approach according to the control of the processes. He distinguish three categories: Systems that base their control on *WF* engines; systems that use agents (in any shape), and systems that are based on the Web Services paradigm.

2.4.2 Agent Related Classification Approaches in WFMS

The term "*agent-based workflow*" was first introduced in 1996 [28], when Chang and Scott labeled their approach as such. The first definitions emerged three years later [18, 21]. The first categorization of the *Agent-involved* WFMS (*AWfMS*) is provided in [55], wherein authors distinguish two classes: *Agent-based* and *Agent-enhanced* workflow. The former refers to systems where "*the software agents take full responsibility for process provisioning, enactment and compensation, with each agent managing and controlling a given activity or set of activities*". The latter is "*a technique whereby intelligent, distributed, autonomous software agents are used to improve the management of business processes under the control of a workflow management system*". This distinction is preserved in [23] as well, wherein authors merely add an ultimate conclusion that agent-based workflow systems are distributed systems consisting of multiple agents and that the whole business process is formed by the pieces of sub-networks within those agents. They also highlight the fact that in agent-enhanced workflow, there is a *Workflow Engine* present, which controls the activities, and the creation – elimination of agents as well. Verginadis [54] appends an additional class in this classification: *Agent-enabled* systems. In the agent-enabled case, broker agents enable workflow instances in distributed *WF* engines. They are used as front-end and they communicate through *APIs* with the *WF* engines.

Joeris proposes a categorization according to agents functionality [46]. He distinguishes three cases: Agents as cooperating actors, as a key infrastructure technology for building *WF* engines, and mobile agents realizing a migrating workflow. The former case concerns a role-based scenario, where agents adopt different roles and carry out the relative tasks. The second case is the activity-based one: agents act as task coordinators and workflow managers. Finally, the last case describes workflow instances migrating to different "*service stations*", where tasks can be performed. Mobile agents can control the migration by selecting appropriate "*service stations*" and can control the execution of tasks and collect their results.

In this thesis, the general term "***Agent-involved workflow management systems***" (***AWfMS***) is introduced, to refer to all the above cases, and in extend, to the overall situation where agents and WFMS are crossed.

2.4.3 Rallying Agents and Web Services to Manage Workflows

Web services (WS) are an attractive infrastructure for workflow since not only they can expose invokable operations but they can support as well an ordered set of messages among them. The advances in *WS* composition and related technologies [11] point out the high potential of *WS* for workflow. This paragraph delves into how agents can enhance Web Services under the workflow concept.

Should anyone collate the workflow properties of the two technologies (*WS* and *software agents*), he will indeed come up with a visible overlapping, as both the composition languages of *WS* and the interaction protocols of agents share the same goals: They both support structured communication among actors; they both distinguish the "role" concept (termed either as partner [56], or role [57]), and they both support common flow mechanisms. So, are the two technologies competing each other?

According to [58], these two technologies seem to be complementary as agents can support WS deficiencies for workflow. More specifically, quoting Huhns: "*WS do not possess any meta-level awareness; they do not inherently understand ontologies; nor they are capable of proactive behaviors, namely: autonomous actions; intentional communication and deliberatively cooperative behavior".* Since agents possess all the above features, they come forward as a great supply.

Two general models of collaboration emerge [22]: The first suggests modeling the Web Service as an agent and treating it as a semi-autonomous one. This way *WS* are enhanced with FIPA-compliant communication, statefullness, and negotiation abilities [59]. The second model proposes exploiting *WS* to describe the external behaviors of agents. The latter approach seems to contribute more in interoperability issues while preserving the flexible interaction patterns provided by agents.

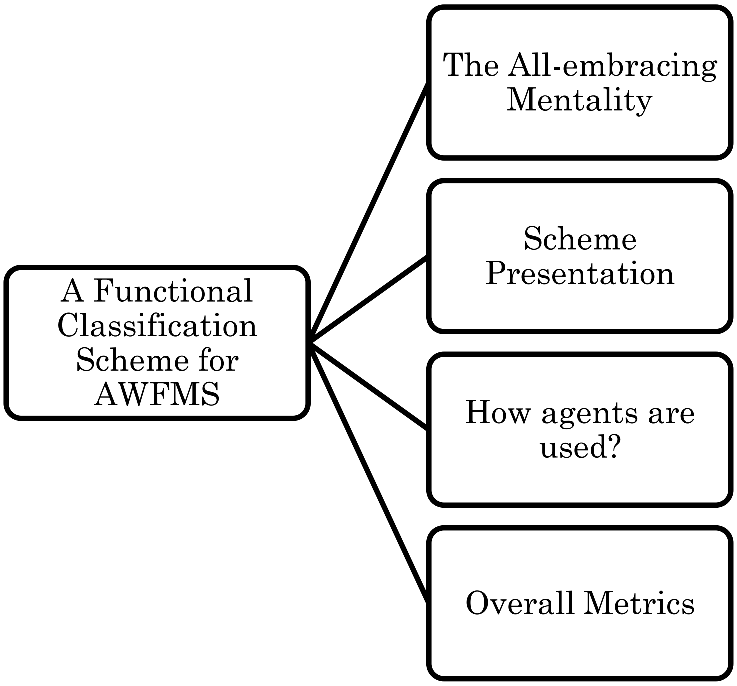
However in the literature, research efforts seem to focus on designing agents that support Web service composition. Vidal [60] exploits agents to overcome the static nature of WS workflows while in [38] agents characterize web services and manage data dealing with composition. In [61] agents forward the instructions of the *WF Engine* to services via messages so that workflow planning based on semantic information is achieved.

2.4.4 Workflow Agents under the Grid Umbrella

The common use of grid and agents is eloquently described by the aphorism of [62] that "*Brain meets Brawn*", parallelizing agents with "*Brain*" and Grid with "*Brawn*". In this context, agents can contribute by making the grid more autonomous and by providing to it flexible behaviors. Since workflow management is one of grid core services, agents' contributions in this particular field shall be briefly discussed.

A natural usage of agents within the grid workflow framework is to exploit their interaction protocols to provide workflow modeling [63] and to coordinate workflow execution [64] in general. Such an approach would supply the system with the advantage of using agents' reasoning models for a sophisticated execution control e.g., for abstracting the flow rules from the strategy that the actors involved may adopt [63]. Moreover, as demonstrated in [64], agents consist a promising infrastructure for the workflows of integration: They provide a flexible integration interface and a reliable and fairly intelligent distributed control mechanism. Additional features of agents, such as the brokering of services [65] and the semantic information exchange [63] allows agents to get more involved in the grid workflow field.

CHAPTER 3


A Functional Classification Scheme for AWFMS
The All-embracing Mentality
Scheme Presentation
How agents are used?
Overall Metrics

3 A Functional Classification Scheme for Agent-involved Workflow Management Systems

3.1 The All-embracing Mentality

Section 2.4.2 presented how researchers classify Agent-involved Workflow Management Systems (AWfMS). Although these approaches provide an abstract view of *how* agents can be used in a WFMS, they offer very little information about *what* they can do. A more specific cataloging of AWfMS is needed. In this thesis, a functional classification scheme is proposed. A functional decomposition of workflow management in [8, p.101]. Ideally, a WFMS should implement all the functions described there (if not more). However, when it comes to the information system perspective, different issues occur. As section 2.2 demonstrated, the development of WFMS shall not lead to islands of automation and systems must be operable in a more open and ubiquitous environment. Therefore, the proposed schema promotes the use of the WfMC standards by suggesting a functional decomposition along the Workflow Reference Model of WfMC [66].

A hierarchy of twenty four functions (utilities) under six branches is proposed (Figure 2). Each branch is associated with a reference model component (Figure 1) so that the proposed scheme fully adopts to the WfMC standards. Besides, WfMC [14] associates every component with an interface, which enables products to conform and / or to interoperate at a variety of levels. This allows mapping quite straightforwardly many dissimilar approaches against a single, unifying framework.

Furthermore, as the reference model is quite popular (hundreds of citations appear in the literature), the proposed scheme claims to be an animated framework. As the reference model does not refer specifically to agents, there was a need to slightly modify the described functions, by appending some functions that derive from agency. An important notice is that the final scheme has a fair orientation towards the use of agents in workflow, so it may not fit a functional classification of traditional WFMS. The classification scheme is illustrated in Figure 2, and explained in detail in the next paragraphs. With respect to the author's knowledge, no such standards-based classification has been suggested so far.

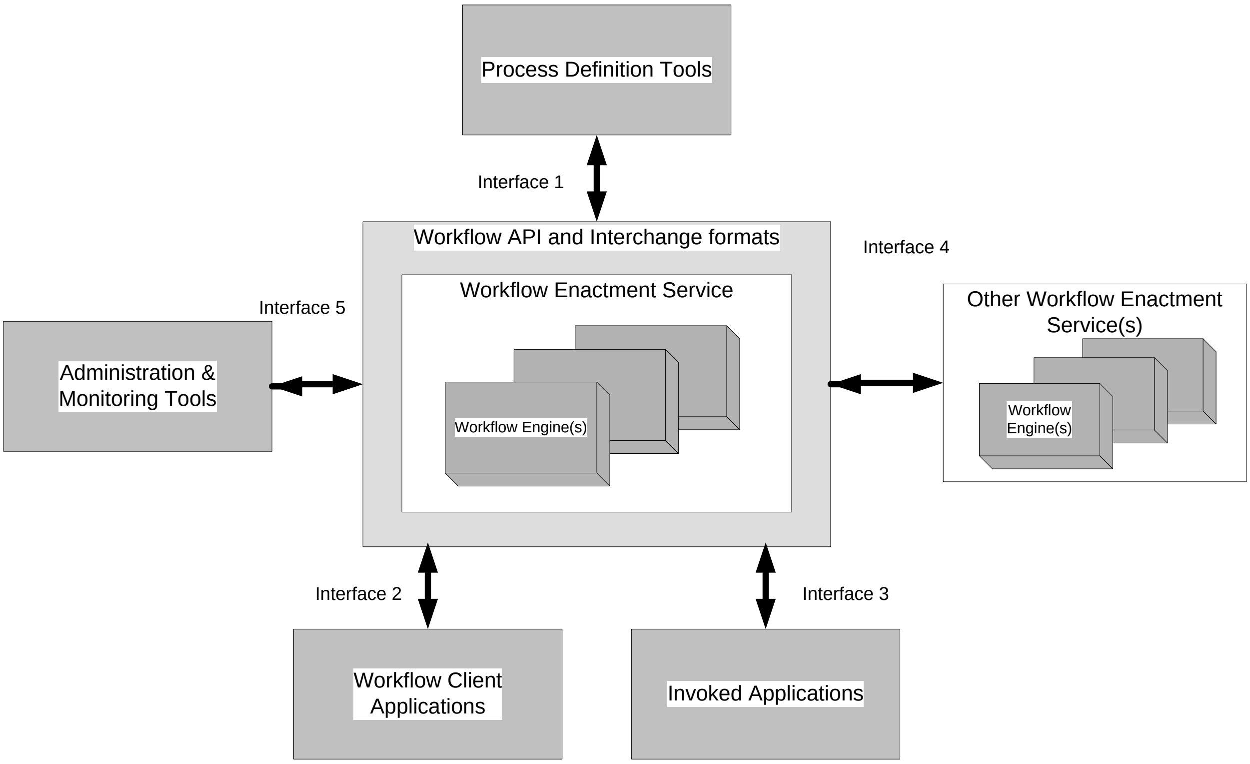


Figure 1 Workflow Reference Model - Components & Interfaces. source WfMC [66].

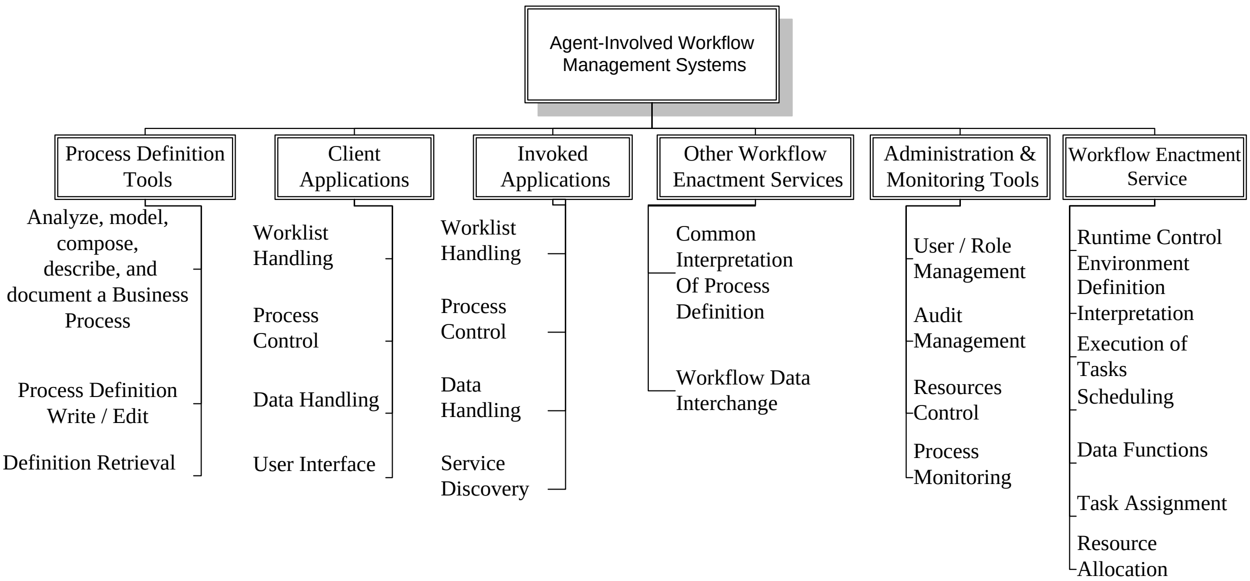


Figure 2 The proposed classification scheme

Based on the proposed scheme, and trying to map each approach against it, a total of 105 publications were reviewed, published from 1996 up to 2008 (see section 3.4 for the summary statistics). When a publication described agents to perform any of the functions listed in the scheme, a check mark was to the corresponded criterion (function). There was no consideration of the extend that agents were used, just if they were indeed used. In addition, the agent definition of [67] is adopted, which defines an agent as a computer system, *situated* in some environment, that is capable of *flexible autonomous* action in order to meet its design objectives. Finally, no distinction was made between systems and methodologies.

3.2 Scheme Presentation

3.2.1 Process Definition Tools Component

The functions described by WfMC in this interface are summarized into three utilities:

1) **Analyze, model, compose, describe, and document a Business Process:** This utility might seem a composite one, but actually the above functions share something in common. These facilities are applied to the process definition during build time. The resulting definition is not operable without agents.

2) **Process Definition Write / Edit:** Agents are capable and authorized to create, edit, and delete objects within a Process Definition. They may also edit any of the objects' properties.

3) **Definition Retrieval:** Agents may get attributes' values from a specific definition. They can also retrieve a list of process definitions that fulfill certain criteria and finally, they can retrieve the whole definition itself.

3.2.2 Workflow Client Applications Interface

This category embraces the interaction between client applications and the core WFMS (usually the *WF engine*). Four distinct activities are listed:

1) **Worklist Handling:** Agents may query the worklist and present to the user the relevant work items. They can query instances and fetch its details to the user. In those queries, agents may search for work-item-level data or for attribute-level ones. Moreover, they may undertake worklist-related notification tasks. Finally, work item decomposition into atomic tasks, when takes place at the client side is considered as a worklist handling operation.

2) **Process Control:** Agents act on behalf of a user in order to create, start, suspend, resume, or even terminate a process instance. Finally, they are able to

shift the process status and to force a change of its state. They play the role of a supervisor, otherwise played by humans.

3) **Data Handling:** According to [66], workflow data are sorted into three types: *WF control* data, *WF relevant* data, and *WF application* data. In this criterion, transactions on all these three types of data are included. Of course, in the case of WF control data, agents communicate the data to the WF engine (or to the alike enactment service) where eventually another agent receives the information, so the corresponded criterion in the WF Enactment service interface (see section 3.2.6) is checked as well.

4) **User Interface:** The explanation of this criterion is intuitive. Agents are the connection tool between the user and the system. An agent is a user representative. A graphical user interface is not considered a sufficient condition in order to get a mark in this criterion. There has to be a fair mapping of the user against an agent.

3.2.3 Invoked Applications Interface

The criteria included in this interface are reasonably similar with the previous paragraph's ones. They expose agents as a promising technology, mainly due to agents' autonomy. Agents are expected to invoke tools or to be themselves the invoked ones. Four patterns are identified:

1) **Worklist Handling:** The activities included here are the same with those of the previous interface, except that agents do not communicate with users but with applications.

2) **Process Control:** Two major approaches are distinguished under this heading. The one is that agents control the applications that they invoke while the other one is that agents are the invoked applications themselves. In the case that agents invoke applications, they carry the orders of the enactment service (usually a WF engine) to applications about starting, suspending, resuming or even aborting. They are also responsible for the synchronization between applications and the WF engine(s). In the case where agents themselves are the invoked application, they have autonomous control of the instance execution.

3) **Data Handling:** Same as "*Data Handling*" criterion of the previous branch. The concern is in all three types of data.

4) **Service Discovery:** This is a function not explicitly included in the reference model, but quite popular in the literature. The rise of Web services advanced radically the field. Agents before invoking an application may semantically or explicitly search for services that implement specific capabilities. Accessing directories where services are catalogued, allows a mark in this criterion as well.

3.2.4 Other Enactment Services Component (Workflow Interoperability Interface)

A fundamental objective of the WF standards and of the WfMC itself is to allow workflow systems produced by different vendors to seamlessly interoperate. There are different levels of interoperation and plenty of connection architectures. We summarized merely two general interoperation utilities:

1) **Common Interpretation of Process Definition:** WfMS may or may not use the same process definition language. In any case, agents are capable of exchanging definitions, while in the case of different languages, they may map the definitions on a common dialect. Agents may request objects and attributes from the process definitions of one system and broadcast them into the WF network as such.

2) **Workflow Data Interchange:** Herein the interchange of both WF control data and of WF relevant data (i.e., state information, recovery points, process state transitions, pre- and post-conditions, assignment messages) are registered. Agents may explicitly transmit these data or they may play a "gateway" role. In addition, any synchronization mechanism is considered as a data interchange technique.

3.2.5 Administration and Monitoring Tools Component

Unless the WfMC standards are followed, there might be confusion between *Workflow Administration* and *Workflow Management* utilities, as "*administration*" and "*management*" do not have always clear boundaries. Nevertheless, complying with the WfMC specifications leads in distinguishing the following criteria:

1) **User / Role Management:** Agents represent individual users or roles. Actions that may be classified as such are user/ role authorization; matching user to roles and vice-versa; personalize system parameters, and agents behaving as proxies.

2) **Audit Management:** In this criterion two types of activities are registered: evaluation and exception handling. These activities are not always separable, thus they are merged into one category. As audit management it is considered the recording of semantic log files; the transformation of log data into semantic ones, and the mining of log data of the workflow instances in order to manipulate exceptions. Additionally, agents that mine audit trails to perform optimization tasks or to account review reports are registered as well.

3) **Resource control:** Agents check for resource conflicts; supervise process concurrency with respect to the resource levels; set access parameters, and define usage parameters.

4) **Process Monitoring:** A rather composite criterion. Herein we classify tasks such as keeping log data (unless semantic ones); process supervising, and querying process status. A single rule is applied to distinguish audit from monitoring: If interpretation of data is required, the case falls to the audit side, else it is classified as a monitoring activity.

3.2.6 Workflow Enactment Service Component

The enactment service supports the runtime environment of a WFMS. The operations listed in this branch are the operations that regularly a WF Engine provides. In certain cases a WF Engine is not present (at least not explicitly), but this does not modify the set of operations that support runtime execution.

1) **Runtime Control Environment:** In this criterion, all approaches that employ agents as runtime control mechanisms are registered. These mechanisms operate as enactment engines. The control refers to a process scope and not to the atomic-task level. Communication among system components and coordination are the most visible runtime control activities.

2) **Definition Interpretation:** The focus is on the cases where agents are able to interpret the process definition language. This criterion concerns just the

interpretation, the other definition-related activities are included in the criteria set of the first component.

3) **Execution of Tasks:** Agents control, and partially or fully execute the atomic tasks that are parts of a WF instance. It is common for agents to wrap other services that finally execute the tasks. This case is indeed considered within this utility.

4) **Scheduling:** Scheduling includes priority assignment, deadline scheduling, routing, creating and supervising synchronization constraints. Agents may perform these activities intelligently or not.

5) **Data Functions:** This is about the general case where agents are responsible for data transactions. Once again, all data types are included, referring however to data handling on the engine side.

6) **Task Assignment:** Agents decide about "*who* is going to do *what*". They have authorities on the global worklist (when such an object exist)and they may edit its content.

7) **Resource Allocation:** Agents decide about "*which* resource should be allocated to *whom*". They implement optimization algorithms. Resources monitoring is an activity registered with a different criterion (see 3.2.4).

3.3 How agents are used? (A survey of the Related Literature)

3.3.1 Process Definition Tools Component

1) **Analyze, model, compose, describe, and document a BP:** In this utility, a great variety of approaches emerges. This diversity probably is a consequence of the low-adaptation of process definition standards. We roughly categorize the approaches into five types:

   a) The agent language is exploited as a (pseudo-)process definition. FIPA protocols are coupled with a process language [21]; Agent Communication Language (ACL) is used to translate the workflow ontology [55], or agent interactions take place on a speech-acts [68] manner.

b) The internal architecture of agents allows the encapsulation of the process definition. Reactive agents anticipate their environment through sophisticated representations like Spheres of Commitment [5], multi-plane state machine [69], or tuples of variables [70, 71]. They often apply *Event-Condition-Action* (ECA) rules that derive either from these representations either directly from the workflow schemes [46]. Agents in [72] act as the transitions in a *Petri Net* (PN) process model, wherein they trigger and they are triggered by the process states (places in the PN). Reflective agents use meta-levels activities to determine their behavior [73]. A Belief-Desire-Intention (BDI) architecture is a case of reflective architecture that is used to model the business process [74]. The workflow definition may also be coupled with a specific role of a workflow participant. This happens in role-based workflow modeling [42], where a role refers to the expected behavior patterns an agent must perform.

c) Migrational Agents. The naming inspiration is after the ability of agents to migrate from one host to another. Agents may be themselves the processes: they may represent the process execution [34], or the process is an object that is enhanced with the properties of agency [17]. Each agent carries the knowledge about how it needs to be processed [41, 75]. A somewhat different approach is when agents are not the entire definition but work-items that are passed to different users and autonomously take care of their current position and further itinerary [76, 77]. An even less complex approach is to model agents as information carriers [78]. Letting agents carry pieces of information while migrating allows (re)configuration of systems. The information as imperative code for host-context exploration/instantiation is a technique used in [79, 80].

d) Service composition. A popular approach, mainly because of the fruitful integration of agents and Web Services. Agents may undertake the realization of an abstract process definition through planning techniques [61], or by providing brokering services [81]. They may also use their interactions protocols as workflow patterns, in order to bind atomic Web services [63, 82].

e) Finally, a multi agent system can be designed to be application specific and to serve specific business processes. There are some fixed components of predefined functionality, but the rest of the functions are either loaded on the fly [83], either designed at build time [15].

2) **Process Definition Write / Edit:** A simple case is to grant permissions to agents to access the definitions repository. Agents may create and delete definitions [59, 84] or create and delete process objects [85]. Changes on processes may be applied either on the static definitions, either dynamically, on the executing instance [86]. A more reflective approach is to let agents modify the dependencies among activities [73]. Sometimes the agent that modifies the definition is instructed by other agents [10], by Remote Procedure Calls (RPC) [87], or even by users and RPC results [29]. In the cases that the definition is encapsulated in the agents, it is obvious that the definitions can be altered by a self-modification of the agent body. When BDI architecture is applied, the agent may determine alternatives situations in which the goals can be achieved [74]. When the process definition is scripted in the body of agents, they can modify the process by inserting their bodies into the run-time environment. Shepherdson et al. [21] encoded the process definition into JAVA classes, so the JAVA agents could modify those classes and re-compile them. A different approach is proposed in [78]. The execution of the processes takes place at distributed processing stations. Agents carry the process-update information while migrating from one station to another. A planning agent is used in [61] to combine the static definition, the user constraints and rules into an executable workflow. A reversed approached is proposed in [69]. The process definition is typed on a blueprint. This blueprint feeds an agent factory to create the corresponded agent.
3) **Definition Retrieval:** Whenever agents are used to model the business process, as described in the first utility of this branch, the retrieval of the process definition is quite straightforward. For the rest of the cases, two similar methods are used: Either a specific agent is charged to retrieve the definition, or a special mechanism fetches definitions to agents. In [24], a process agent is used to get the workflow specification. In [59, 84] the process agent ask the definition from the storage agent, who in its turn, access a database to get the information. The trigger agent, used in [88], acts more or less the same since it transfers the process definition to the other agents. The activity agent suggested in [89], connects likewise the business process model with the system's agent hierarchy. A coordination layer where agents dwell, is proposed in [22, 25] in order to communicate with a workflow management layer to retrieve the definitions.

Object Request Brokers (ORB) is used as a mechanism to allow agents to communicate with the WF Engine [90], while Blake [37, 38, 82, 91] utilizes a representation parser that feeds the Global Workflow Management Agent with the process definition.

3.3.2 Workflow Client Applications Interface

1) **Worklist Handling:** Worklist handling operations are addressed by a variety of methods. The worklist-handler agent proposed in [92] is a visible example of how agents support these operations. It enables work items to be passed from the WFMS to users, and notifications of completion or other work status conditions, to be passed between the user and the WFMS. A popular approach [32, 75, 76, 93, 94] is to let agents communicate directly with the workflow engine or the worklist server. In these cases, agents act simply as data couriers that facilitate information exchange. Personal agents that represent users [85, 95, 96] are a generalization of this case. A different approach is to assign worklist handling operations to control agents [97-99]. These agents have a more coordinative substance and handling worklist is one among their duties. In [17, 100, 101], worklist handling is also a duty for executor agents. Finally, worklist may have a special representation (e.g., tuples [71], or workflow policy rules [101]) which agents may access and interpret.
2) **Process Control:** Usually clients exploit the interface facilities to control the processes. It is very popular for the special interface agents to encapsulate process control abilities. Yanli [99] uses such an Interface Agent to provide clients with process control, while a personal interface agent is also used in [102]. A personal agent carry out the control on behalf of the users in [103] as well. It achieves this by communicating with users and Task Agents. The personal agent of Chang [28], constructs HTML pages and invokes a WWW browser for those pages. Users are able to invoke various tools through those pages. Transforming web pages into a standardized GUI which supports the migration of agents is proposed in [29]. The agents encapsulate all information and code required to allow human users to interact directly with the agent itself or indirectly with a remote service. The web is also the enactment environment of the personal agent in [85]. Treating agents as Web objects allows each agent to have a Web page, which is easily accessed by clients [83]. The web environment allows researchers to follow the client/server architecture, where agents are client-side components

of Web applications while other functional WF components are their servers [104]. More active approaches are also proposed: Clients may interact with agents that execute the processes (Task agents in [46], and Actor agents in [105]). A workflow coordinator in [32], initiates process instances requested by users, by creating proxy agents and dispatching them to workflow engines. Similarly, the users can control through their interface a stationary agent that creates and dispatch a messenger agent into the right server for certain tasks [106]. Agents being e-forms that accept users' invocations are suggested both in [75, 77]. Budimac [76], generalizes this idea by conceptualizing mobile agents as work-items that are circulated among users. In the case that users are related with roles, agents represent them, inherit the permissions and prohibitions governing the creation, usage, and deletion of the processes [42]. Gudes [107] names these agents Alter-Egos.

3) **Data Handling:** As declared in section 3.2.2, this heading includes the transactions that agents may realize in all the three types of WF data: WF control data, WF relevant data, and WF application data. Concerning the control data, one can consider the approaches used here as a spontaneous extension of the approaches described during the previous criterion (*Process Control*). For the rest types of data, there can be enumerated approaches like the Site Manager Agents of Blake [38] who populate a data repository; the storage agent [28] who is responsible for providing a uniform access mechanism (HTTP protocol) to multiple database systems; the Manager Agent [100] who accepts user requests for data, and the Agentboard [104] which is the repository for storing agent properties (relevant data are captured as these properties). Interface agents are commonly used to transfer data within the WFMS [26, 29, 71, 92, 99, 108]. An agent may also be used as a gateway between the client and a legacy database of the system [109], or it can even represent a part of the database itself [107]. Agents can also support the data integration in grid systems [110], where data exchange is intense.

4) **User Interface:** Agents act as effective bridges between users and computers. Such agents can make the human–computer interface more intuitive and encourage types of interactions that might be difficult to evoke with a conventional interface [40]. The simplest shape is agents that provide secretarial functions [96] and act as a "fairly dumb" assistant to support their user [95, 102]. A graphical user interface is often embedded [29, 99]. However, interface agents

can be more sophisticated. In [71], the interface agents are responsible for collecting information about customers and orders. These agents also interact with customers during order execution, informing them about order status and possible problems. In [111], the interface is a mapping from input to output. An agent receives tasks through its input. The output is a set of agents' behaviors. Finally, this criterion's activities may be implemented not by a dedicated agent, but by a more general one. For example, the management agent of [84] provides among else the user interface for the human workflow manager.

3.3.3 Invoked Applications Interface

1) **Worklist Handling:** The approaches proposed for this utility are fairly similar with the ones of the previous interface. Of course, in this case the "*users*" are replaced by "*applications*". Agents act more autonomously in their interactions with applications rather than with humans. The worklist agent proposed in [88, 103, 112] enriches its functionality by exploiting its autonomous collaboration with other agents (register agent, personal agent).
2) **Process Control:** In section 3.2.3, two major approaches were distinguished under this heading. The one is that agents control the applications that they invoke while the other one is that agents are the invoked applications themselves. The latter approach is used for instance in ADEPT [15], wherein agents have control over the tasks that they may perform. The concept of service agent is often applied: In [21], each agent is responsible for one or more service offerings, where a service offering is some combination of workflow activities and the resources that are contingent upon them. The service agent of [113] is an agent on behalf of a service entity that is capable of providing certain facilities, while in [93] service agents run in distributed containers and after receiving a task assignment, they autonomously invoke the required services. The Role Manager Agents [38] play a role in the workflow execution by fulfilling one or more services as defined by the workflow policy in a centralized database. These services may be Web Services or other services encapsulated by other agents. A lot of researchers focus on the integration of Web services: A BPEL4WS specification is used in [43, 45, 72] to allow agents coordinate a set of Web Services. Applications or Web services are captured by resource agents [59], while manager and process agents request task execution from them. In a similar way in [24], agents are utilized to wrap services which are able to execute

workflow tasks. The process agent manages the execution flow of the tasks according to the workflow's Event-Condition-Action (ECA) rules. It can enable, disable, suspend or resume the tasks according to the workflow ECA rules. Once more, the Workflow Provider Agent proposed in [114], controls the execution of atomic processes involved into the business process by invoking, requesting, or informing different Resource Provider Agents. The agents of [115] contain a *WF engine* which calls Web Services where directed by the workflow. Zhao in [110], utilizes Web Services as an interface for controlling legacy workflow engines. Of course, his agents (Scenario managers) may control the legacy engine via different interfaces: Web Services, Socket, or command line. A more general concept is to consider agents as task managers [87]. Each one is implemented as a CORBA object and exports certain public methods as an external interface, including the process control methods. An interesting suggestion is that of [69]: An agent considers the process as a finite state machine, thus it controls it through state transitions – actions. All actions carried out by an agent are the result of the execution of a certain strategy decided when in a specific state. The decision making abilities of the agent and his strategy selection, eventually provide him with the process control.

3) **Data Handling:** The methods proposed for this utility does not differ significantly with those of the second interface criterion: "*Data Handling*". They are rather intuitive techniques of data exchange between agents and other agents, agents and applications, and agent and Web services. In the first case, messages of an agent language are transferred; in the second ad-hoc protocols are used, and in the last one SOAP messages are the most popular approach.
4) **Service Discovery:** In this utility, agents appear to search and advertise services as well. In the frequent case when Web Services are integrated, it is common for a UDDI registry to be maintained. Agents operate on this registry using a semantic tool, like DAML-S [45, 60, 61], and OWL-S [22], or explicitly searching for the desired services. Of course, services are not always web ones. They may refer to the services that a WF engine provides [110], to resources' monitors [100], or to active WF instances [116]. Once again, these services are listed in a registry that agents can access. A third case is when agents are themselves registered in a repository. They can be discovered by a Directory Facilitator Agent [72, 99], by a peer agent through the use of an acquaintance model [16], by a dispatcher agent [85], by a central agent [98], or even by a

special broker agent. Agents may get advertised to the broker by populating their JAVA classes interfaces [49], using FIPA protocols to update broker knowledge [21], or following a special brokering process that the system prescribes [81]. Wang [117] uses a information board to publish agents' beliefs. When a peer agent searches for services, it enters the board and translates the beliefs into capabilities. A negotiation-oriented approach is also proposed [63]: the contract net protocol [118] is used in order to discover which agents can offer the required services.

3.3.4 Other Enactment Services Component (Workflow Interoperability Interface)

1) **Common Interpretation of Process Definition:** The "*common interpretation*" concept in this criterion comes in three versions: the first one adheres to agents that share a centrally-hosted, executing Workflow definition; the second one to agents that are guided by a common definition, and the last one refers to the case that the definition is collectively maintained. Concerning the first version, the definition may be handled by a WFMS while agents execute its partial activities [21]. The notion of a server that maintains the definitions is also supported in [28]. The proposed server is an agent which accepts request from other agents for process definition information retrieval. The model proposed by [41], besides handling centrally the definitions, it segments a workflow definition into blocks, and assigns each of them to a mobile agent. Merz [29, 34] launches the concept of the *Service Representation* (SR). The SR encapsulates the definition while it is developed and provided by a remote server. It is possible to store the SR persistently and to suspend / resume interactions with the remote server. A sub-category of this version is the use of a definition template. The template may be hosted in a server and agents who execute a process instance based on that template, communicate with the server when an exception occurs [75]. Agents that transform definition templates into instances are also suggested in [24, 99]. In a similar way, agents may reason over the meta-model of the definition [10], thus they are able to recognize and manage its variants. The approaches of the second version are quite different. Buhler and Vidal, in a set of their works [45, 60, 72] apply a BPEL4WS definition to express an initial social order on agents. A coordination dialogue among agents is utilized as the process definition in [119]. It is distributed to the interested parties, while the distribution is achieved

by making the dialogue definition publicly available for download through a repository. The methods that use a collective approach exploit the properties of agency: A BDI architecture is used [74, 102] to represent the processes context. Spheres of Commitment [120] and tuple centres [70, 71] are used for the same purpose. A different approach is presented in both [78] and [116]: The workflow object (which carries the definition among others) is moving from node (processing station) to node as its state advances. Nodes are able of course to understand the state of the object, operate on it and perform the required activity, before advancing its state and forwarding it to the next destination.

2) **Workflow Data Interchange:** The use of two dimensions in order to group the approaches is suggested: The first one is to group them along a "*distribution*" criterion and the second one along the *technique* used. For the distribution scale, two options are considered: the central and the distributed one. The former refers to the case that a common point of reference is used to maintain the control information (the point of reference may be a server [38], a special control or monitor agent [24, 90, 121], or a shared repository like tuple centers [70] and information board [117]). After the execution of an activity, an agent leaves its stigma at that reference point, hence the status of the process is updated. This way, the status of every process becomes transparent to all agents, allowing a fair dissemination of the control information. The latter refers to a peer to peer approach, when agents interchange the control data without the intervention of a supervising entity. A peer to peer approach requires a formal interaction protocol among agents. This protocol may be message oriented [63], dialogue-based [119], definition-guided [87, 116], or even based on the mobility of agents [76, 78]. As long as for the second dimension, numerous techniques are used. It is common to allocate the control data interchange to a special agent [22, 24, 90, 97, 99, 103, 114, 121] who is either dedicated, or it has a more general function. No matter if agents are special ones or not, they indeed use messages that contain control data as a communication mean [24, 63, 97, 104, 119]. It is also popular for agents to exchange not just messages but the entire process definitions in order to get synchronized with the process execution [16, 26, 29, 41, 72]. Sometimes, they even use themselves as the communications mean [32, 75, 76, 78, 79, 116]. They migrate from host to host while the control data are embedded in them. Finally, agents may use a reasoning mechanism to communicate the control information. A merging agent who merges the execution plans of other agents [65]; a

backward chaining approach to form a provisioning plan [21], meta-data interpretation [79], or deliberative reasoning over a BDI architecture [74] are listed as such techniques.

3.3.5 Administration and Monitoring Tools Component

1) **User / Role Management:** A popular approach is the design of personal agents. This kind of agents may provide the user interface for humans [76, 77, 86, 105] supporting their communication with the system. Personal agents may also perform more sophisticated actions like customizing the user's working environment [28], filtering and coordinating his/her communication [47, 103], or even managing his/her worklist [97]. Another popular approach that derives from the natural abstraction of agents as autonomous actors is their mapping against roles. A role is usually attributed with capabilities, goals, obligations, permissions over resources, qualifications etc. [41, 98, 117]. Such a role-based conceptualization can be extended to map the workflow of organizations [39], or federations [111] on a multi-agent architecture. This is the case that a role refers to an organic component of a process. Blake suggests that agents should behave likewise, by adopting and fulfilling specific services [37, 38, 91]. Last and actually least, agents are used to undertake user management activities [97]. Researchers seem to prefer to let user management (authorization, authentication) to other technologies than agents.
2) **Audit Management:** Approaches in this utility fall on two broad categories, which indeed overlap in some parts. The first category refers to the evaluation issue, while the second one includes approaches that strive to make the WFMS fault tolerant. In the latter category, there are cases like special diagnostic agents to handle exceptions [108, 117], negotiation [16, 99] or voting [81] protocols . Agents may support the system by re-planning the process [21, 96] or simply by identifying a consistent checkpoint to resume [98]. Concerning the evaluation field, simulation claims as an efficient tool [82]. Performance agents may also be incorporated in the system for evaluation reasons [28, 84, 92, 97, 106, 108]. Sophisticated features for audit, such as learning from previous experiences [77], recommendation for future enactments [94], reputation mechanism [122], and adaptation to modified instances [19], are fairly advantaged by the features of agents. No matter the audit activity (exception handling or evaluation) two basic update mechanisms are distinguished: A

bottom-up one, where agents communicate the error or the performance measure to a central entity [90, 96, 98, 99], and a top-down mechanism, where a central entity inspects the system to identify abnormalities or collect data [25, 84, 92, 97, 105]. An additional interesting feature is the use of agents to agree a specific level of monitoring [40, 122] in order to reduce network traffic.

3) **Resource control:** A visible classification of the approaches in this criterion is to distinguish the distributed from the central ones. The distribution perspective allows agents to communicate each other on a peer-to-peer basis; checking resources availability or priority rules. Resources may be associated directly with agents [26, 96], thus resources' requests correspond to messages among agents, or resources may be associated with static points on a net [116], thus requests are registered there-in. Central approaches implement of course a central entity which supervises resources and controls their conflicts, their availability, and their accessibility. Guidelines for this supervision may be described in the process plan [21, 120], or they can be general rules of the system (e.g. request levels considered as thresholds) that the special entity guards [10, 17].
4) **Process Monitoring:** A typical technique is to dedicate a special agent of the system in monitoring processes [59, 84, 85, 90, 92, 108, 117, 119]. It tracks and monitors the status of all agents and operations of workflow processing, while it is also responsible for the information storage. An analogous approach is to use again a special agent, but not a dedicated one [21, 77, 83, 98, 100, 123]. This kind of agents performs additional activities in parallel, often process management and control activities. An inherent evolution of this technique is to distribute the monitoring process: Agents being capable of reporting their status [17]; migrational agents [24], and agents as log-data carriers [78] are proposed. Finally, less distributed but also collaborative approaches are suggested in [19] and [97]. These approaches decompose the monitoring tasks and assign each of them to a special agent. For instance, in [97], there is one agent to monitor the progress of the workflow while another one focuses on monitoring the exchange of messages.

3.3.6 Workflow Enactment Service Component

1) **Runtime Control Environment:** By definition, the runtime environment in WFMS is provided by the *WF Engine* [1]. Still agents can provide runtime

services that may be used by the system components in an operable assembly. The contribution of agents' technology in this function comes in three shapes:

a. A central agent acts as a *WF Engine* [108, 115] or as a facilitator to the *Engine* [49].
b. Two distinct servers coexist: a server to manipulate agents and a server to support workflow enactment [28, 75, 90, 97, 99]. A major issue in this category is how to synchronize the functions of these two servers. Solutions vary from attaching the agent server into the Workflow Engine [32], up to creating a special interface between agents and the *Engine* [28].
c. A multi – agent architecture. This case is unsurprisingly the most popular one in AWfMS and the one that benefits the most from the agents' paradigm. Three sub-categories can be identified within this case:
    i. Agents use a special representation language that encapsulates the workflow behavior [4, 5, 42, 65, 71, 73, 120, 124].
    ii. Communication and coordination is achieved though messaging and agent communication protocols [10, 15, 17, 21, 46, 77, 81, 85, 96, 98, 113].
    iii. Service-oriented architectures. Agents are not only used to encapsulate (wrap) services, but also advertise, search and coordinate them [24, 38, 82]. An inverse technique is to use a web-services process scheme to coordinate agents [45, 60, 72].

2) **Definition Interpretation:** The interpretation of the process models is by definition a fundamental function of WFMS. Usually, agents understand the language of the process models, where process models exist as bare entities. Nevertheless, agents are used as well when process models are more complex notions. For instance, in Knowledge-driven processes [74, 125] a *Beliefs – Desires – Intentions* (BDI) architecture is employed: agents have explicit goals to achieve (*desires*), or events to handle (*intentions*) in order to carry out a process. Likewise, in [29], the process is represented as an encapsulation of the agent's local state. The WF engine executes operation invocations and passes agents on to other engines. Each invocation advances the local state of an agent until the process goal (the final agent's state) is reached. The concept of embedding the process model into the agents' body, while agents are moving from node to node (e.g., engine to engine, resource to resource) is also followed in [49, 80, 126].

Finally, agents can be used in ad-hoc WFMS, i.e., in systems which do not support every process but just some pre-described, specific ones. In this case, agents do not actually interpret the process definitions, yet they decipher some process parameters [95, 100, 108].

3) **Execution of Tasks:** The automation of the process execution is fairly enhanced with the use of agents. In AWfMS agents appear to autonomously execute workflows: sometimes they undertake the whole process [84, 114], and sometimes just the atomic tasks, according to their expertise and capacity [25, 85, 88, 99]. Besides this typical case, agents can be additionally exploited to control the process [16, 21, 49, 70, 71], even if the actual execution is not their piecework. In a slightly different way, agents may be themselves the subject of work: they travel while they carry the necessary information. Their destinations (either engines, machines, resources in general) act upon agents, so the process steps forward [29, 32, 75, 77, 78]. Agents could even act upon themselves, by executing an internal method or by modifying their state or behavior [73, 106, 111]. In a different method, agents wrap services which do the actual work [24, 38, 60]. The role of agents in this case is to provide a smooth integration framework and a more convenient control mechanism for services.

4) **Scheduling:** By its nature, scheduling is an activity that is seldom individually addressed. Usually it is coupled with task assignment or resource management issues. However, trying to isolate the scheduling activities where agents are involved, three broad categories are outlined: The first one exploits the context of the agents' society. Agents follow some market-based procedure (i.e., negotiation) [20, 96, 127], or some message exchange protocol [26, 63] to mutually agree on a scheduling scheme. The second category includes a more central approach. The *WF Engine* or another central entity applies either special rules [19, 46, 73, 101]; or provides scheduling modules to the agents [32, 80]; or utilizes special techniques and algorithms (e.g., temporal logic [44], AI planning [65]), or finally it follows some prioritization discipline [17, 106]. Some approaches that jointly use methods of both these categories (negotiation together with some optimization method [16, 21]) are also proposed. The last category of scheduling methods in AWfMS relies on the mobility of agents: Agents have a complete knowledge of their itinerary and they schedule themselves to travel from node to node [34, 76, 78]. This category's methods, inherently distributed, do not necessarily yield optimal scheduling efficiency.

5) **Data Functions:** This criterion actually refers to data handling on the engine side. Three major styles can be identified concerning the involvement of agents with *WF relevant* data and *WF application* data (*WF control* data concern more the agents' functions within the "*Runtime Control Environment*" criterion). The first one is to use a special agent (like the Data Management Agent of [90]) or a special kind of agents (Information agents of [40]). A different style is to assign data functions to two or more dedicated collaborating agents. One can recognize this style in [84, 121], where a Storage agent and a Monitor agent work together; in the collaboration of the Trigger agent and the Personal agent in [112, 128], yet in the cooperation of User agents with the WF Execution agent in [97]. Finally, another style is to provide access mechanism to every agent that needs to access external data-spaces [45, 82].
6) **Task Assignment:** Two major approaches appear to address the task allocation issue in AWFMS:
   a. The *negotiation* mode, where an agent negotiation context is applied [15, 20, 114, 127]. The Contract Net Protocol is broadly used [18, 63, 89, 113] to allow agents to negotiate over a set of evaluation criteria. The incorporation of Service Level Agreements (SLA's) to bind the negotiation process is an intuitive way to quantify the evaluation criteria [16, 21].
   b. The use of a *hierarchical* structure to dispatch tasks. The hierarchy may refer to a central entity that is responsible to decide an allocation plan and notify the task executors of it (e.g. a Dispatcher agent [85, 99], a Coordination agent [98], a Decision Making agent [90] or even a Judging machine [97]). The hierarchy may also refer to a brokering architecture [81, 113] or even to special mobile agents [41, 47].

   No matter the mode employed (negotiation or hierarchical), a popular method is to match task demands against agents' abilities or agents' roles abilities [17, 24, 27, 42, 81, 88, 89, 98, 99, 103, 105, 113, 129]. The role acts as a filter to the worklist so that a more efficient matching between agents and tasks is possible. The *Task Assignment* problem within AWFMS is often addressed as an optimization issue [17, 27, 127, 130] while various techniques like Reinforcement Learning [122], Maximal Sequence Model [41], Support Vector Learning [21], UPC theory [27] have been proposed.
7) **Resource Allocation:** An important notice is that the resource allocation decision in AWfMS is a run-time decision and that agents are employed to

contribute to a dynamic allocation of resources. Under this context, the dominant technique is negotiation [10, 16, 18, 20, 125] where agents claim for resources by offering bids. Broker agents, who keep a registry of the available resources and facilitate the negotiation process, are also suggested [22, 84, 121]. By its nature, the *Resource Allocation* problem appears tightly related with scheduling [131] so that scheduling techniques are attached either to negotiation [16] or to brokering [99] to address more efficiently the allocation decision. Other techniques proposed are ECA Rules [73], backward chaining [55], and UPC theory [27] that enhances agents with a self-learning ability in order to avoid resource collision and to allocate resources more efficiently.

Interface	Criterion	References
Process Definition Tools	Analyze, model, compose, describe & document a BP	[4, 5, 15-17, 21, 28, 29, 34, 41, 42, 44, 46, 55, 61, 63, 65, 69-83, 85, 89, 103, 104, 115, 117, 126, 132]
	WF Definition Write/ Edit	[10, 21, 29, 59, 61, 69, 72-74, 78, 79, 83-87, 133]
	Definition Retrieval	[15, 16, 21, 22, 24, 25, 27-29, 32, 34, 37, 38, 41-43, 45, 46, 55, 59-61, 69, 72, 75, 83, 84, 86, 87, 89-91, 93, 96, 99, 103, 104, 112, 113, 116, 119, 120, 123, 132-134]
Client Applications	Worklist Handling	[17, 26, 28, 29, 32, 34, 70, 71, 75, 76, 85, 92-103, 112, 126, 128, 132, 133, 135]
	Process Control	[17, 20, 25, 28, 29, 32, 34, 42, 46, 49, 75-77, 83, 85, 95, 99, 102-107, 123, 126, 132]
	Data Handling	[17, 20, 26, 28, 29, 34, 37, 38, 47, 54, 71, 75, 77, 91, 92, 94-96, 98-100, 102, 104-110, 114, 123, 126, 132-134]
	User Interface	[2, 5, 17, 20, 25, 26, 28, 29, 34, 37-40, 42, 46, 49, 54, 55, 59, 70, 71, 76, 77, 83-86, 91, 92, 95-100, 102-112, 114, 116, 117, 120, 122, 128, 133-135]
Invoked Applications	Worklist Handling	[17, 21, 22, 26, 28, 32, 34, 37, 38, 55, 59, 70, 71, 75, 81, 83-85, 89, 90, 92, 93, 97, 98, 100, 101, 103, 112, 113, 128, 132, 133]
	Process Control	[15-17, 20-22, 24, 25, 29, 32, 34, 37, 38, 42, 43, 45, 46, 49, 54, 55, 59, 60, 63, 64, 69, 72, 75, 80-85, 87,

Interface	Criterion	References
		91, 93, 97, 103, 104, 110, 113-115, 121, 123, 132-134, 136]
	Data Handling	[15-17, 20, 21, 24-26, 29, 34, 37, 38, 43, 47, 54, 55, 64, 65, 69, 72, 75, 80, 81, 83, 85, 87, 91, 92, 98, 100, 104, 109, 110, 116, 123, 126, 132-134, 136]
	Service Discovery	[15-17, 21, 22, 24, 29, 34, 37, 38, 45, 49, 54, 55, 60, 61, 63, 65, 72, 81-83, 85, 91, 93, 98-100, 110, 113-117, 119, 123, 133]
Interoperability	Common Interpretation of Process Definition	[10, 21, 24, 28, 29, 34, 37, 38, 41-43, 45, 46, 54, 55, 60, 63, 64, 70-72, 74, 75, 78, 82, 83, 87, 91, 96, 99, 102, 104, 110, 116, 119, 120, 123, 132-134]
	Workflow Data Interchange	[15, 16, 21, 22, 24, 26, 28, 29, 32, 34, 37, 38, 41, 46, 47, 54, 55, 63-65, 70-72, 74-79, 82, 83, 87, 90, 91, 96, 97, 99, 100, 102-104, 106, 110, 113, 114, 116, 117, 119, 121-123, 126, 132-134]
Administration & Monitoring	User/ Role Management	[2, 5, 17, 28, 37-39, 42, 47, 54, 59, 63, 77, 83-86, 89, 91, 95, 97, 101, 103, 107, 111, 114, 120, 129, 132-134]
	Audit Management	[2, 15-17, 19-21, 25, 28, 40, 43, 54, 59, 73, 77, 78, 81, 82, 84, 90, 92, 94, 96-99, 105, 106, 108, 116, 117, 122, 133]
	Resource Control	[2, 10, 15-17, 20, 21, 26, 39, 42, 55, 59, 83, 84, 96, 98, 116, 120, 129]
	Process Monitoring	[4, 15-17, 19-21, 24-26, 40, 41, 49, 54, 55, 59, 64, 75, 77, 78, 82-85, 87, 90, 92-94, 97-100, 106, 108, 117, 119-123, 126, 132, 133, 136]
WF Enactment	Runtime Control Environment (Communication/ Coordination)	[4, 5, 10, 15-18, 20-22, 24-26, 28, 32, 34, 37-39, 41-47, 49, 55, 61, 63-65, 69-86, 89-92, 95-100, 102-104, 106-108, 110, 113-115, 117, 119-121, 125, 126, 132-134]
	Definition Interpretation	[4, 5, 10, 15-17, 19-22, 24-26, 28, 32, 34, 37, 38, 41-47, 54, 55, 59-61, 64, 69, 72, 73, 75-77, 79, 81-87, 89-

Interface	Criterion	References
		94, 96-99, 102-104, 106, 107, 110, 112-116, 119-123, 128, 132-134, 136]
	WF Instances Control or Execution	[15-17, 20-22, 24-26, 28, 29, 32, 34, 37-39, 41-43, 45-47, 49, 54, 59, 60, 63, 69-75, 77-80, 82-86, 89-91, 94, 97-101, 103, 104, 106, 107, 109, 111-117, 121, 123, 126, 128, 132-134, 136]
	Scheduling	[15-17, 19-21, 26, 27, 32, 34, 41, 44, 46, 55, 61, 63, 65, 69, 73, 75-78, 80, 86, 87, 94, 96, 99, 101, 102, 105, 106, 115, 116, 126, 127, 132]
	Data Functions	[15-17, 20, 21, 25, 26, 28, 29, 32, 34, 37-41, 43, 45, 54, 59, 70-72, 75-78, 82, 84, 86, 87, 90, 91, 94-97, 100, 104, 106, 108, 109, 112, 114, 116, 121, 122, 126, 128, 132-134]
	Task Assignment	[5, 15-18, 20-22, 24-28, 39, 41, 42, 47, 60, 63, 64, 70, 75, 77, 81, 83, 85, 89, 90, 92, 93, 97-101, 103, 105, 106, 110-114, 116, 120, 122, 123, 125, 127-129, 132]
	Resource Allocation	[10, 15-18, 20, 21, 26, 27, 55, 59, 73, 83, 84, 96, 99, 112, 116, 121, 125]

Table 1 Classification of the existing literature in AWfMS.

3.4 Overall Metrics

In order to thoroughly explore the intersection of Workflow Management Systems and agents, a plethora of publications were reviewed. Their types are summarized in Figure 3, the category “other” including technical reports, a PhD Thesis, a patent and an open source development environment. In general, while searching for relative publications, there were no limits about the publication mean or specific journals or conferences since the topic of this study spans across different areas. Hopefully, the variety of 32 distinct journals that were closely examined is a fine account of that endeavor. As long as for the time period of the publications, it is illustrated in Figure 4. The reference period is slightly longer than a decade (1996-2008), while the publications’ chronological distribution is fair enough.

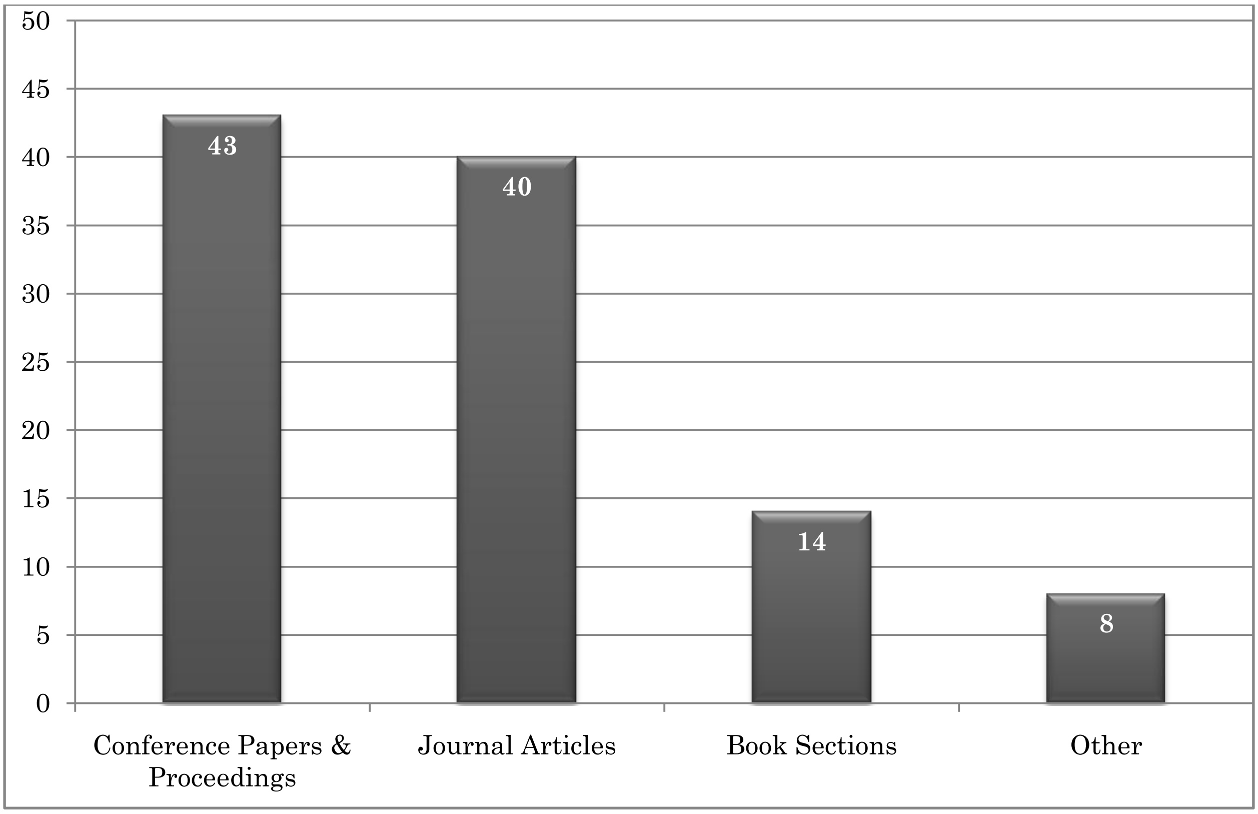


Figure 3 Distribution of the reviewed publications according to their type.

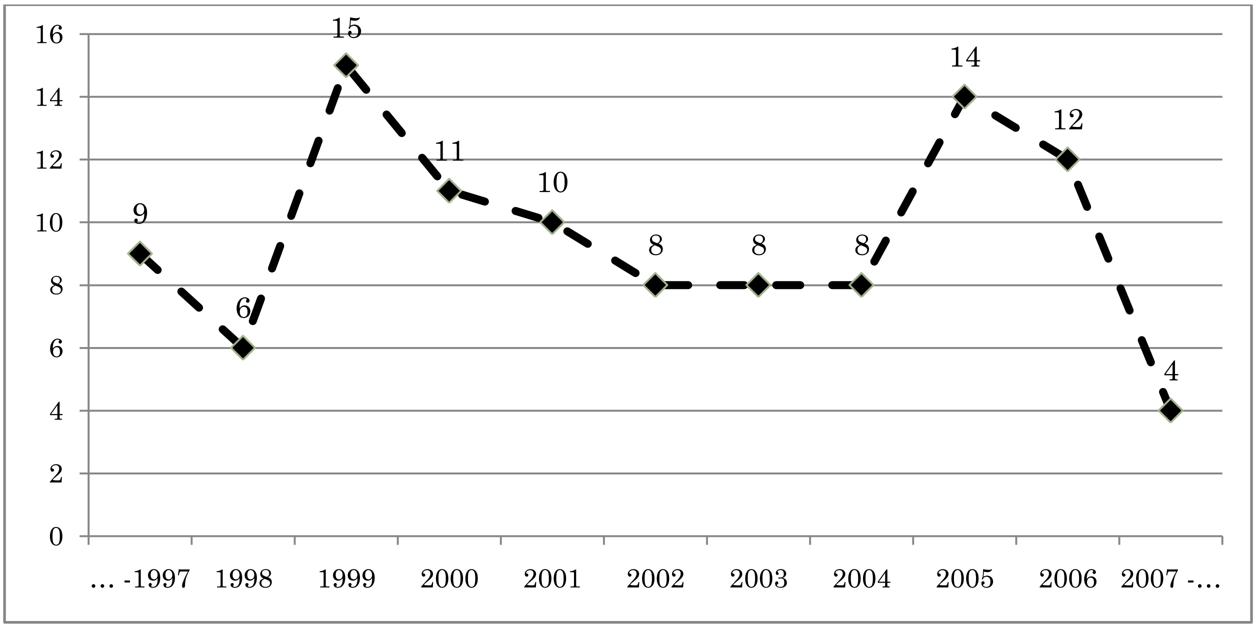
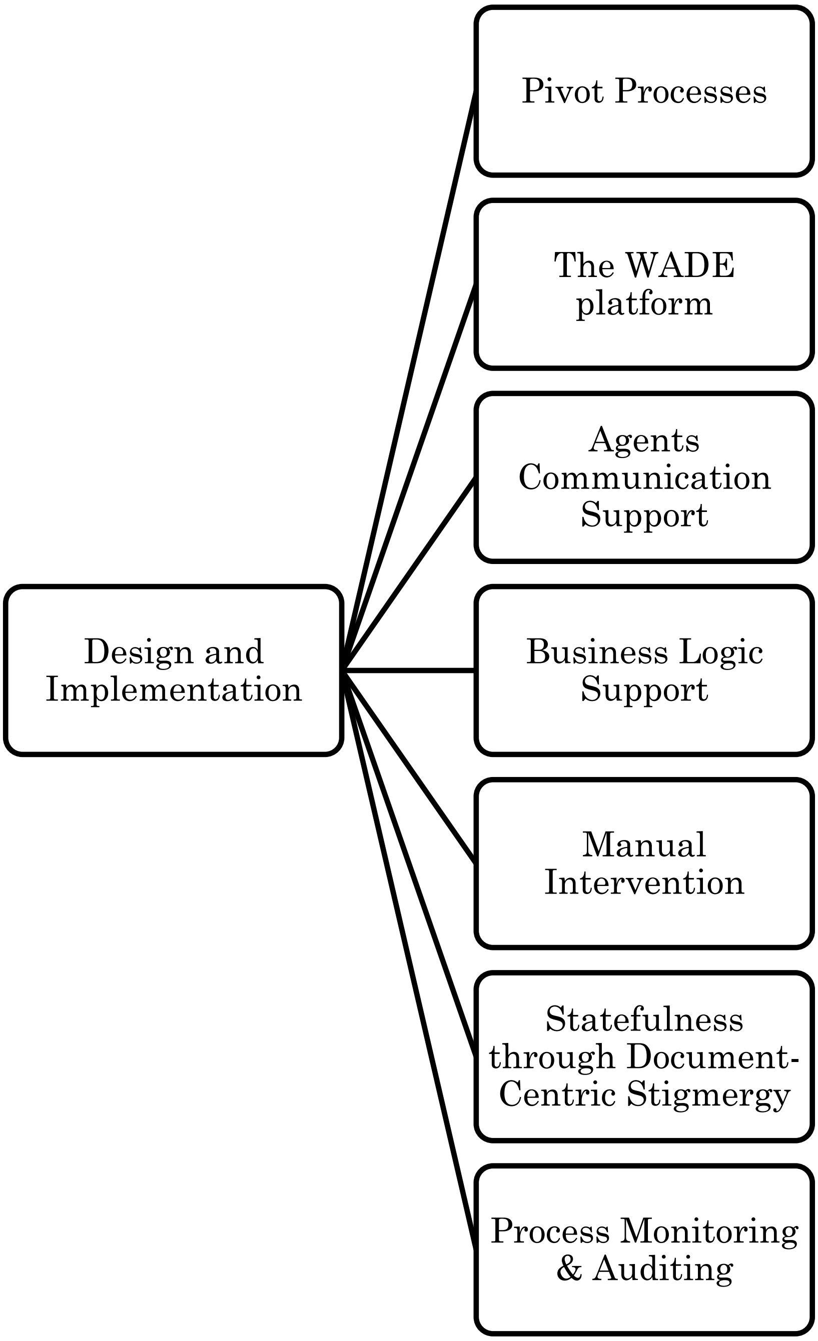


Figure 4 Chronological distribution of the reviewed publications

CHAPTER 4



Design and Implementation
Pivot Processes
The WADE platform
Agents Communication Support
Business Logic Support
Manual Intervention
Statefulness through Document-Centric Stigmergy
Process Monitoring & Auditing

4 Design and Implementation

4.1 Pivot Processes

The proposed agent-involved workflow management system approach is domain abstract, meaning that it could be applied to any domain, as long as the formalization requirements hold. Actually, this is the role of workflow management systems, which are introduced to separate process logic from business logic. However, the thesis theme, as defined by the sponsor program, dictates that the proposed system should be applied to the specific domain of marketing.

In point of fact, marketing is a very convenient domain for workflow management applications: Marketing processes are far more flexible and versatile than production processes since the process flows are not rigidly defined, heterogeneous resources are involved, and high customization per customer is required. However, the regular activities required to carry out a marketing process (e.g., writing a report, extracting data from databases, organizing campaigns, schedule meetings, etc.) have good potentials to be monitored by information systems. To such a context, automation prospects are significant and tightly related with the workflow perspective.

In order to fit the marketing domain, two pivot processes are selected and implemented. During the process selection procedure the following criteria were considered:

- The process is possibly long and comprises rich social interactions among the participants.
- The process is fairly complex and interactions among activities and / or participants are reasonably sophisticated
- The process environment is heterogeneous and demands asynchronous communication
- The process demands extensive human participants integration
- The process has fair automation potentials.

Complying with the above criteria, the two pivot processes which were identified are the *direct mail campaign* and the *customer contact center management*. Since no formal workflow definitions exist in the literature neither it is available by corporate organizations, the workflow definitions were built from scratch. The fundamental base was the generic guidelines that handbooks of marketing provide [3, 137] and published material from vendors where available. Moreover, the partner Next Step Ltd., a company which operates in the marketing business and which is contributing to the sponsor program, acted as a vital catalyst to the refinement of the definitions and to their adjustment into the business reality. Finally, an ultimate filter for the process definitions was the goal to exhibit the system features, i.e., some process elements were regulated in such a way that the AWFMS features were visible.

4.1.1 Direct Mail Campaign Automation

Direct mail marketing refers to sending an advertisement, offer, announcement, reminder or other item to a prospective customer. Kotler [3, p. 536] identifies direct-mail marketing as a major marketing communication mode, and as an important mean to inform, persuade and remind consumers about the brand. In fact, direct-mail campaigns serve multiple communication objectives, such as producing prospect leads, strengthening customer relationships, informing and educating customers, reminding customers of offers, and reinforcing recent customer purchase decisions.

Direct mail marketing (as opposed to mass marketing e.g., advertisement) is a targeted communication and is based on a one-to-one, brand-customer basis. It is becoming increasingly popular, as it can be personalized, a quality of great importance in demassified markets. Direct mail campaigns include a broad mixture of tools and activities such as budgeting, forecasting, managing digital assets, and dealing with complex scheduling requirements. Because of the proliferation of products and brands, even larger number of market segments, fierceness of competition, and overall acceleration of change, direct mail campaigns have become complex and their planning and administrative decisions must be made under increasing time pressure. Indeed, timing and sequencing activities within a campaign is one of the critical decision variables [137].

The rough main activities of a marketing communication process (and thus of a direct mail campaign) have been analytically described in popular handbooks of marketing [3] (see Figure 5). However, it is clear that a campaign could focus on some special steps or it could omit some others, it could execute the steps sequentially or parallelize the process, according to the campaign's special requirements. Moreover, each step may contain different activities in a variety of flows. Because of the above particularities, every campaign may significantly differ from another.

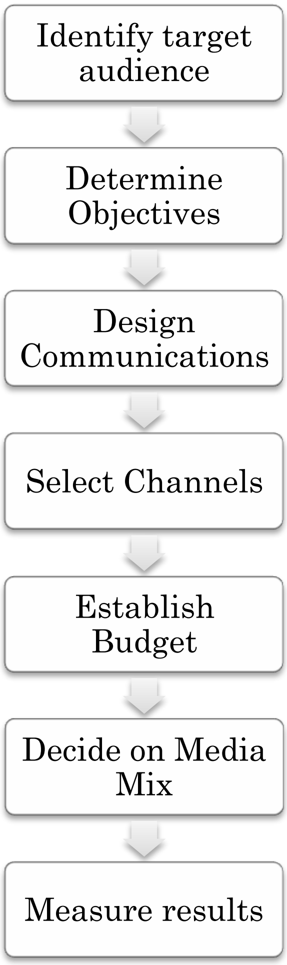


Figure 5 Basic steps in developing effective communications (source: [3, p. 541])

To support the management of direct mail campaigns, and provide organizations with automation potentials, some vendors (SAP [Table 2], Microsoft[3]) provide marketing campaign blueprints so that charting a campaign project and monitoring its workflow is facilitated. In this thesis, the basic outline of a direct mail campaign process is maintained, resulting in the detailed workflows described in the appendix.

Table 2 SAP Business Workflow in Campaign Automation. *Source: http://help.sap.com/ saphelp_crm70/helpdata/EN/45/cbced6f771fae10000000a1553f6/content.htm*

Workflow templates for Campaign Automation
WS14000061 Transfer Target Group to Channel
WS14000062 Create Target Group
WS14000062 Create Target Group and Channel Transfer
WS14000064 Send E-Mail to Employee Responsible

[3] http://ce.microsoft.com/en-us/templates/ TC012330891033.aspx?CategoryID=CT102115851033

WS14000065 Authorization by Employee Responsible
WS14000066 Adding a Business Partner to a Target Group
WS14000067 Deleting a Business Partner from a Target Group
WS14000068 Start Target Group Optimization
WS14000069 Transfer Respondent to Channel
WS14000070 Start Subsequent Step Without Executing
WS15100040 Start Media Campaign

4.1.1.1 Key actors involved

The job roles and the corresponding job titles may vary significantly. In this section, the job roles, which are involved in the direct mail campaign which was implemented, are described:

- ***Marketing Director***: He / She directs the organization's overall marketing and strategic planning programs, and corporate communications. The main responsibilities of the director are to design, implement and facilitate the organization's marketing plan; to support and facilitate the development and implementation of sectional / marginal marketing plans; to plan and administer the marketing operations budget; to oversee marketing development activities; to develop and administer marketing database; supervise the staff of the marketing department.
- ***Product Manager***: The Product Manager is responsible for the product planning and execution throughout the product lifecycle, including: gathering and prioritizing product and customer requirements, defining the product vision, and working closely with engineering, sales, marketing and support to ensure revenue and customer satisfaction goals are met. The Product Manager's job also includes ensuring that the product supports the organization's overall strategy and goals. The Product Manager is expected to: Refine the product strategy according to the business objectives; prioritize the features of a product providing the appropriate justification; be an expert with respect to the competition.
- ***Marketing Communicator***: The marketing communicator (MarCom) supports sales and marketing management with communications media and advertising materials to effectively represent the company's products and services to customers and prospects. He / She reviews literature in the assigned marketing

project, previous marketing materials used in the assignment area, and gathers materials of competitive companies in the field. Additionally, the MarCom researches, writes, develops sketches of supporting graphics, and consults with printing firm representatives on the needs of the particular project; he /she develops draft advertising text and layouts as part of campaign materials and he is involved to the review and approval procedures.

- ***Marketing Assistant:*** The marketing assistant provides administrative support to the staff of the Marketing Department. Duties include general research, clerical, and project based work.

4.1.2 Customer Contact Center Management

A customer contact center is a central point in an enterprise, from which all customer contacts are managed. The traditional contact centers were actually *call* centers, wherein agents were answering phone calls. However, as new communication styles are emerging, this type of contact centers is becoming obsolete. Customers want to reach organizations via e-mails, messages from their cell phones, messages through the organizations website, etc. So, organizations need to reach their customer using the communications channels the customers desire. A major difference between the above mentioned channels and the typical telephone line is that communication is getting asynchronous. This feature alone requires for different management of a contact center.

Although the general business objectives and the performance drivers are independent of the communication style, when an asynchronous mode is employed, a different understanding of resource management tasks and concepts is required. Due to the flexibility and versatility of asynchronous communication channels, answers to the "*who*, *what*, *when*" should be redefined. An important part of managing the contact center is providing schedules that are workable and help achieve business objectives. A contact center is generally part of an organization's overall customer relationship management and its management would usually be provided with special software.

In this thesis, the process described in [138] is adopted, as a typical scenario for contact center management. In particular, the situation addressed is when a batch of customer e-mails arrives to an organization's server, and the organization's staff struggles to process them in a timely manner. E-mails concern one of the following topics: *WARRANTY, INSTALLATION, TROUBLESHOOTING, ERROR, SPECIFICATIONS, and GENERAL*, while the average processing time of serving an e-mail of a specific topic is considered to be known. In addition, the organization has established some quality of service standards, i.e., every mail must be served no later than six hours after its arrival. The abstract phases of the process are illustrated in Figure 6.

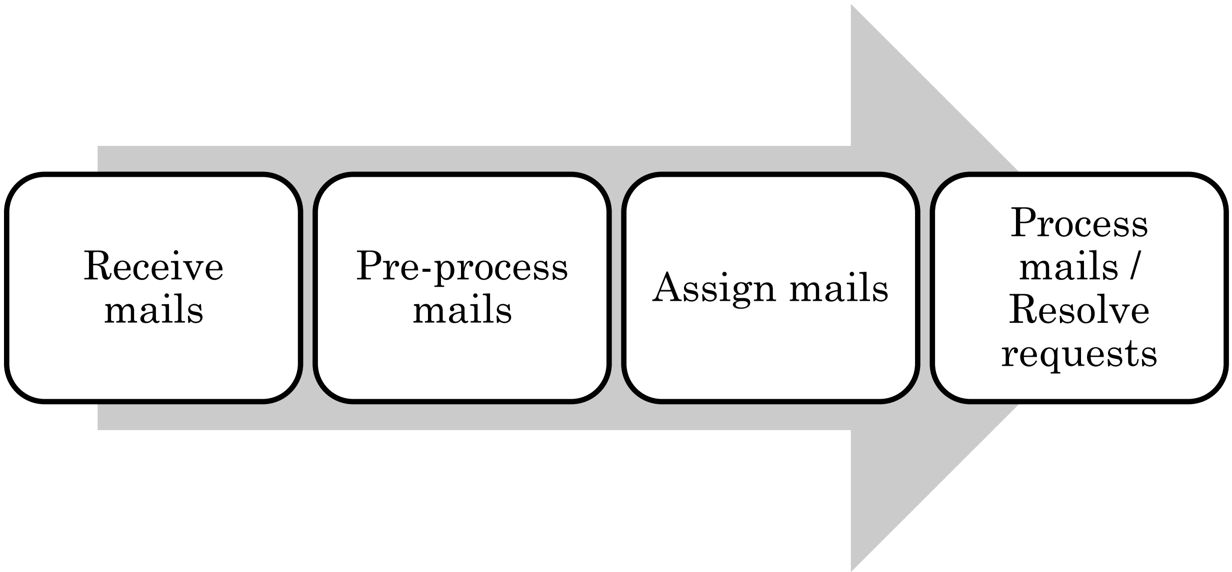


Figure 6 Basic phases of the contact center management process

There are some general business objectives that the management process should consider. These objectives are related to cost control (average cost of putting an agent online, agents' occupancy, non-productive agent time, etc.), customer satisfaction (response time, service level etc.) and employee satisfaction (fairness, supervisor support etc.). These objectives should be translated into specific performance drivers and be subjected to optimization techniques. An analytical application of an optimization algorithm based on this process is presented in section 5.3.

4.1.2.1 Key actors involved

A contact center should have a supervisor, a manager, who normally is an organization's executive. The supervisor of a contact center is responsible for the daily running and management of the center through the effective use of resources with responsibility for meeting, and possibly setting, customer service targets as well as planning areas of improvement or development. Contact center executives ensure that incoming requests are answered by staff within agreed time scales and in an appropriate manner. They coordinate and motivate the center's staff. Typical work activities include defining performance drivers for speed, efficiency, quality and other business objectives; planning and managing the daily running of the center; maintaining up-to-date knowledge of its staff capabilities and performance; organizing staffing, including shift patterns and the number of staff required to meet demand; improving performance by raising efficiency etc.

The other key actor in a contact center is its contact agents. A contact center agent is a person responsible for answering the queries of the customers. They are responsible to satisfy customers and maintain good image for the company. A contact agent must understand the impact of the language he/she uses while he/she should effectively deal with the customers' questions or problems. A contact agent accepts its worklist form his supervisor and he/she should perform his/her assigned tasks with punctuality.

4.2 The WADE platform

WADE (Workflow Agent Development Environment) is a software platform that facilitates the development of distributed multi agent applications where agent tasks can be defined according to the workflow metaphor [4]. WADE is built on top of JADE [139], which provides a distributed runtime environment, the agent and behaviour (a task performed by an agent) abstractions, peer to peer communication between agents and basic agent lifecycle management and discovery mechanisms. An analytical presentation of the WADE platform can be found in the WADE's web site[4]; however, the main elements and features of the platform are explained in the following paragraphs.

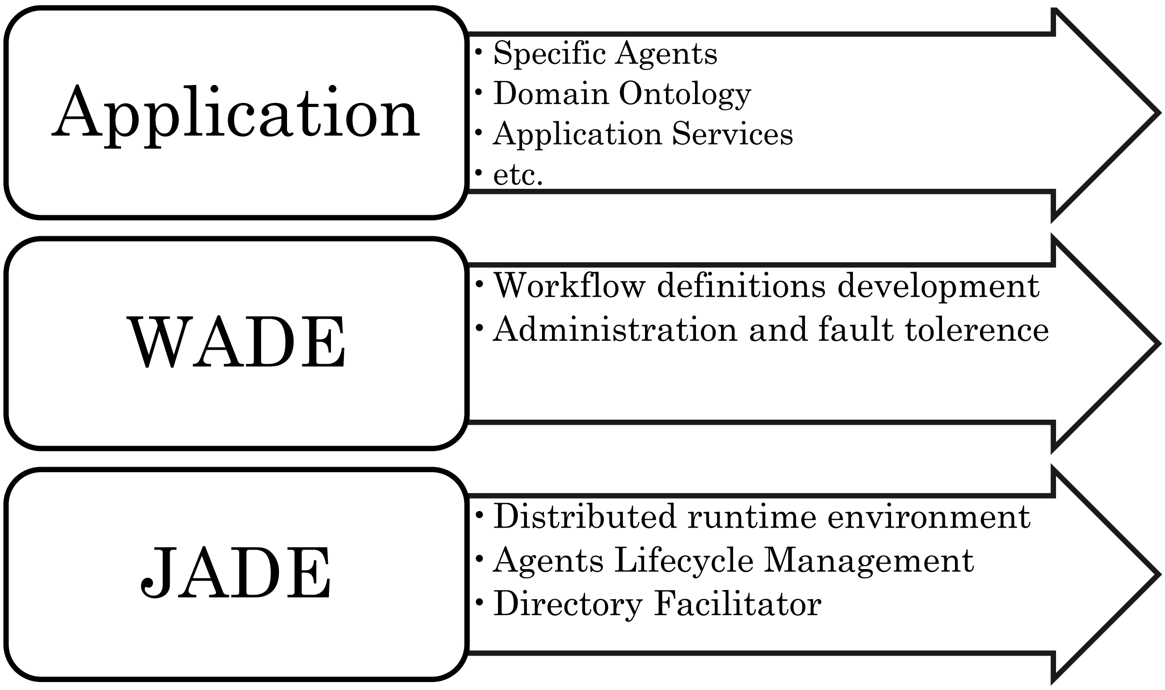


Figure 7 The WADE-based application concept

[4] http://jade.tilab.com/wade/index.html

The abstract idea of a multi-agent application based on WADE is illustrated in Figure 7: At the bottom, there is JADE which provides a FIPA-compliant multi-agent platform that supports agents' creation and lifecycle management, the fundamental constructs of `Agent` and `Behaviour`, yellow pages services and a distributed environment to deploy the application. The next layer is provided by WADE, a tool to enhance with workflow metaphors the JADE platform. Finally, on top of these, the application specific design is set up.

WADE, in contrast with most workflow management systems, does not supply a single workflow engine. It essentially provides an extension of the basic `Agent` class of the JADE library called `WorkflowEngineAgent` that embeds a small and lightweight workflow engine. That is, application specific agents that extend the `WorkflowEngineAgent` class become workflow enabled. A second important point is the workflow definition formalism that WADE uses, and which is the JAVA programming language. However, the WADE view of a workflow class follows the XPDL meta-model, thus building a workflow class turns out to be an ordinary process engineering task.

In order to deploy a multi-agent application on top of WADE, the basic WADE components must be marshaled. The architecture design is illustrated in Figure 8, where the main components are visible.

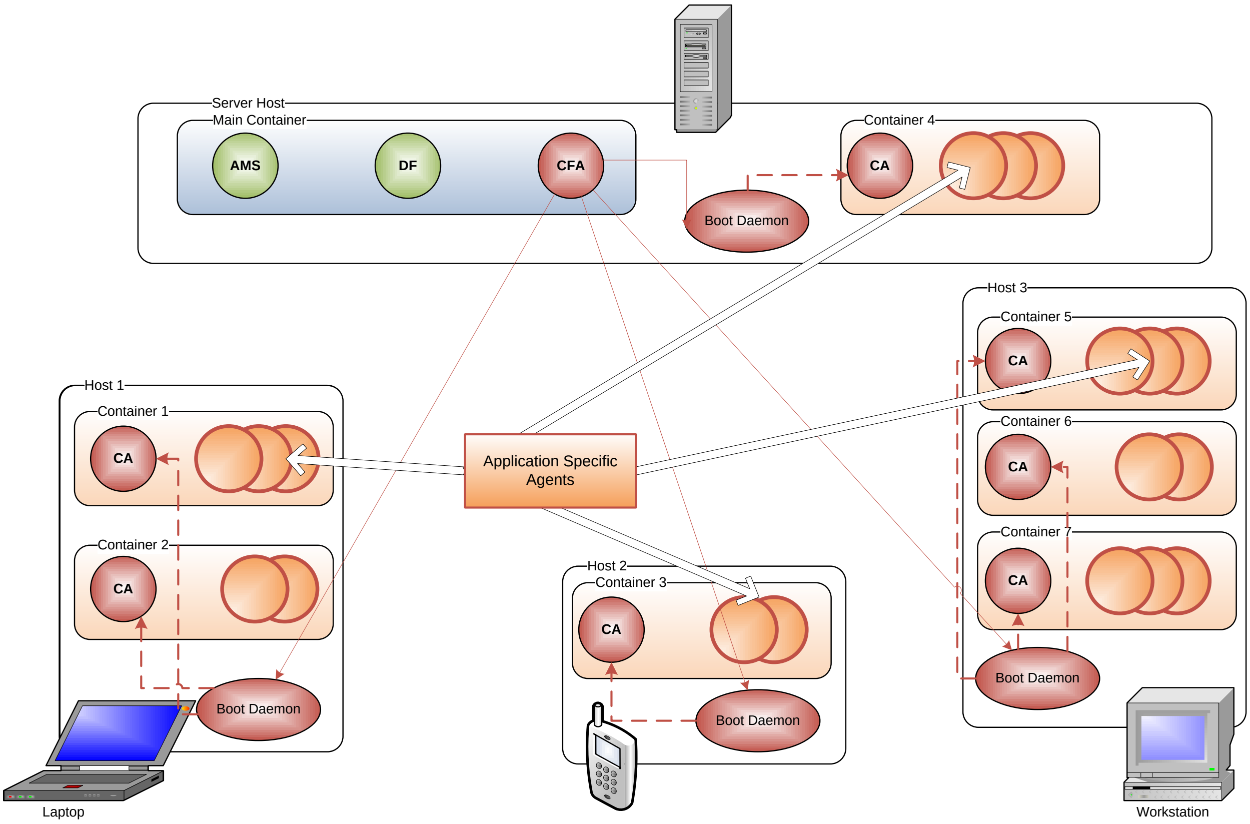


Figure 8 The WADE Architecture

- **The Configuration Agent (CFA).** It is always running in the Main Container (along with the Agent Management System (AMD) and the directory facilitator (DF), and it is responsible for interacting with the boot daemons and controlling the application life cycle.
- **Boot Daemons.** A Boot Daemon is activated at each host. Each daemon is responsible for activating the workflow containers in their local host.
- **Controller Agents.** Every container that needs to be workflow enabled must contain a Controller Agent (CA). The CA is responsible for the supervising activities in the local container and for all the fault tolerance mechanisms provided by WADE.

In order to start a WADE-based application, the Main Container (including AMS, DF, CFA) and the Boot Daemons should be set up and running. The Main Container is launched accepting a property file (`main.properties`) to configure its parameters. An additional file (`types.xml`) is read by the platform to define agent types and roles. Finally, upon application's start-up, an *application configuration* is loaded. An application configuration is a file that specifies, according to an XML based format, which hosts are involved, which containers must be executed in each host and which agents must be activated in each container.

4.3 Agents Communication Support

Agents' communication in the application is inherently message-based, as the application is built on top of JADE. The type of exchanged messages follows the FIPA-ACL specification [140], which in turn is based on the work of [141]. In particular, the agent communication language (ACL) used, stands on the speech act theory which states that messages represent actions or communicative acts (called from this point and on as *performatives*). Some simple and popular examples of such acts (performatives) are the INFORM action, the PROPOSE, the REQUEST, the AGREE etc. In this section, the focus is to present how agents' communication is enhanced by the workflow metaphor. Three different styles are described, each per subsection. Moreover, this section exhibits some workflow cases which agent-involved workflow management systems are particularly suitable to implement and enact.

4.3.1 Interaction Protocols

Usually conversations among agents fall into typical patterns, i.e., they use the same sequences of messages of the same performatives. FIPA has standardized some of the these typical patterns and called them Interaction Protocols (IPs) [142]. FIPA Interaction Protocols specifications deal with pre-agreed message exchange protocols for ACL messages.

In the application developed, the inter-agent communication workflow logic is designed to make agents sufficiently aware of the meanings and the goals of the messages exchanged, so that an IP can instinctively implement the agent's planning process. This design is inherently favored by the specified interaction protocols; as the planning process frequently matches a sequence of communicate acts.

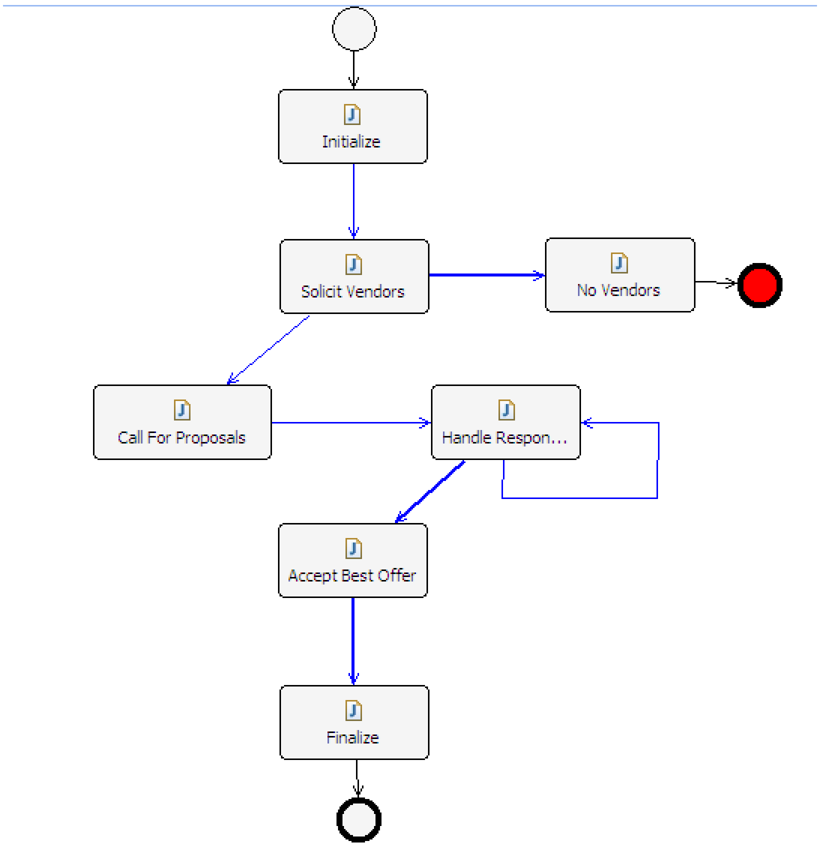


Figure 9 The workflow of the `SolicitDesign` class

To clarify the above statement, an illustrative example is presented. This example concerns the `SolicitDesign` workflow class (see Figure 9). The general objective of this process is to select a vendor who will produce the marketing piece at the most low price, holding of course the specified requirements. The process accepts the piece requirements as input, while at the output, it returns the name of the winning vendor (actually it returns the identifier of the agent that represents the vendor). Vendor agents calculate the offer that they might make (they may of course refuse to make any offer) by calling a web service. The web service itself is called through another workflow class (`VendorOffer`, see Appendix). The whole process (save the initialization & the finalization code) can be mapped on the contract net interaction protocol specification [143]. Figure 10 demonstrates the sequence diagram that implements the contract net interaction protocol and derives from the run of an instance of the `SolicitDesign` workflow class, during a sample case of three available media vendors.

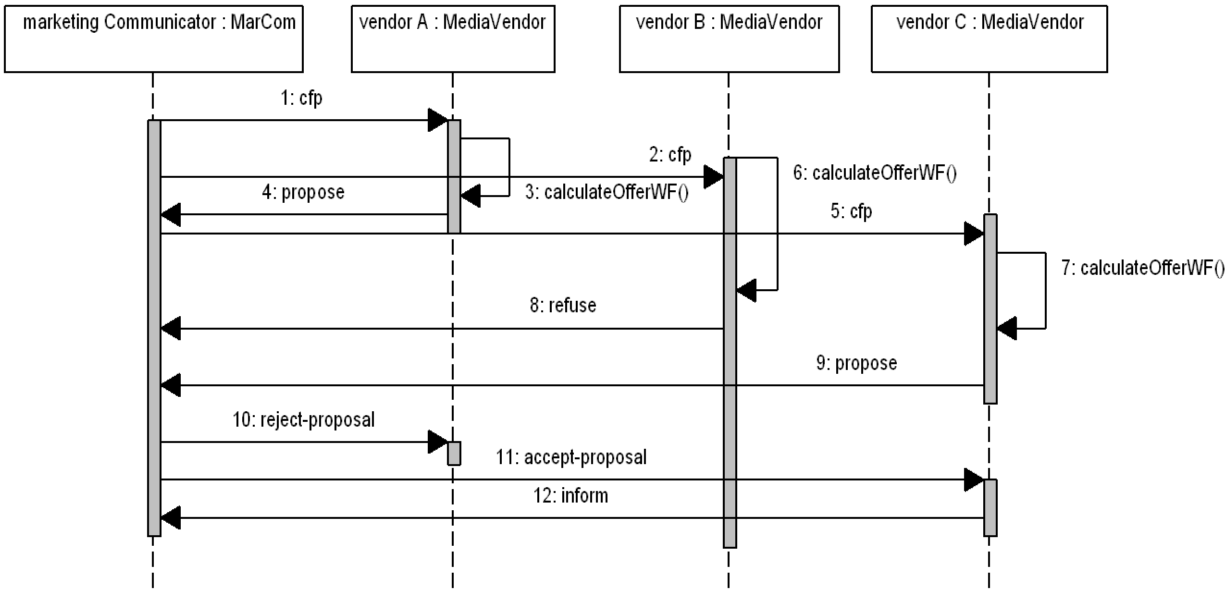


Figure 10 The contract net protocol implemented during an instance of the `SolicitiDesign` workflow

In the above case a workflow class coincides with an interaction protocol. Definitely, the same interaction protocol can be implemented outside a workflow class; however exploiting the workflow metaphor facilitates the whole procedure, since the graphical representation of a workflow allows smooth integration of interaction protocols with external tools and activities.

4.3.2 Joined Interaction Protocols

A different case is when inside the scope of a process, two or more interaction protocols must take place so that the process logic is realized. For instance, during the `EstablishTargetMarkets` process, the Marketing Director communicates a checklist to the Product Manager, requesting him to fill / refine the document. The product manager replies either negatively (refuse) or positively (agree). In the latter case, he sends an additional informative message at a later time notifying the results. These actions are exactly described by the FIPA REQUEST Interaction Protocol, so the "Communicate List" activity within the `EstablishTargetMarkets` process implements it, carrying out a piece of the process logic. However, the process logic requires that next, during the "Arrange meeting" activity, the Director propose a date to the Manager in order to arrange a bilateral meeting. The Manager can either accept or not. This interaction is prescribed by the FIPA PROPOSE Interaction Protocol, which is implemented by the "Arrange meeting" activity (see Figure 11). What is ultimately achieved is to join two interaction protocols under a special workflow logic (herein a sequence). This style represents the modeling of IPs as individual activities, as distinct puzzle pieces that can be combined with other activities or tools to form a process according to the business needs. An emerging advantage of this style is the reuse of the activities that implement an IP, into different, potentially more complex, processes.

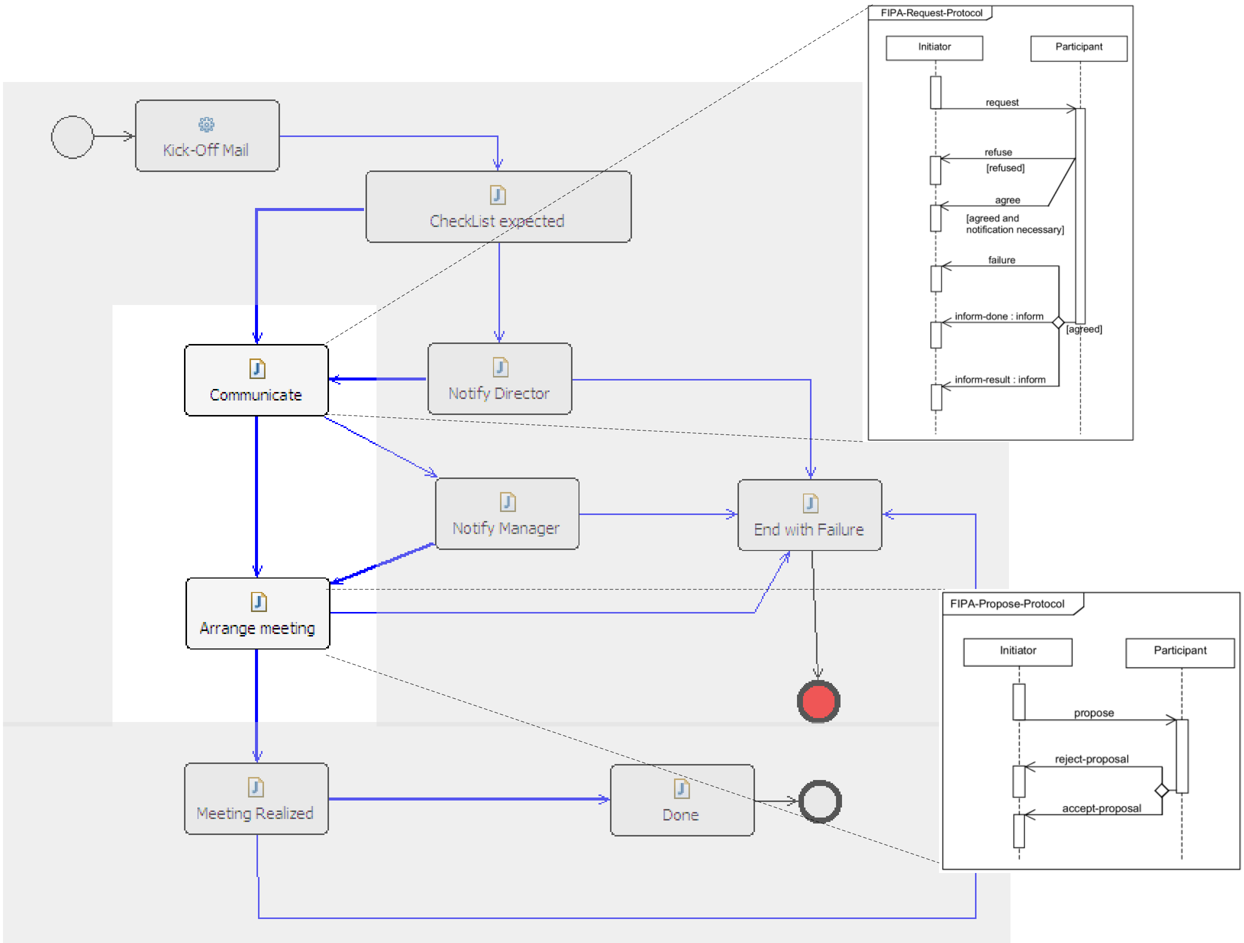


Figure 11 Join two interaction protocols during one process

4.3.3 Unspecified Interactions following a workflow logic

FIPA has specified eleven (11) typical patterns of messages exchange (Interaction Protocols). Although these eleven protocols address the most popular interactions, it is quite possible for an interaction pattern to happen following a different logic, not specified in any FIPA protocol. In such a case, the workflow metaphor provides a good mean to control the messages exchange under a well-structured marshal. Consider the example of the `ReviewDrafts` workflow class, illustrated in Figure 12.

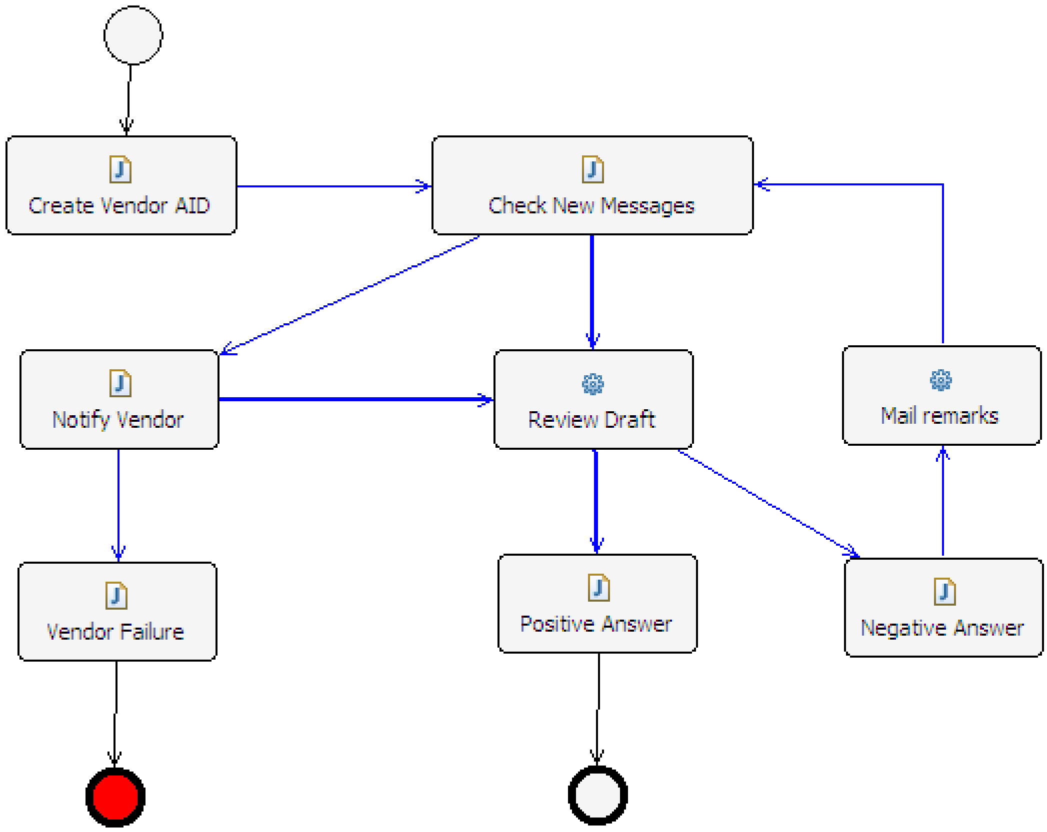


Figure 12 The ReviewDrafts workflow diagram

According to the `ReviewDrafts` process, the Marketing Communicator waits for the vendor to send a draft of the illustrations needed for the marketing piece. The marketing Communicator shall review the draft and reply positively or negatively to the vendor. The positive answer means that the drafts are accepted without any changes, and that the vendor shall go into the production phase. A negative answer includes a document explaining the modifications that are necessary. When the proposal is finally accepted, the vendor notifies the communicator that it enters the production phase. The whole process seems like the PROPOSE interaction protocol: The arrival of the draft is announced through an ACL message of the PROPOSE perfomative while the answer is another message either of the REJECT_PROPOSAL or the ACCEPT_PROPOSAL perfomative. Nevertheless, there are two important differences that do not allow the FIPA specified PROPOSE Interaction Protocol to be applied as it is. The first one is the cardinality of the protocols occurrences. The propose – decision – counter propose – decision pattern may be repeated over and over again until a positive answer takes place. The second difference refers to the final action of this interaction, that is, the notification (INFORM message) the vendor sends to the communicator when it is entering the production phase.

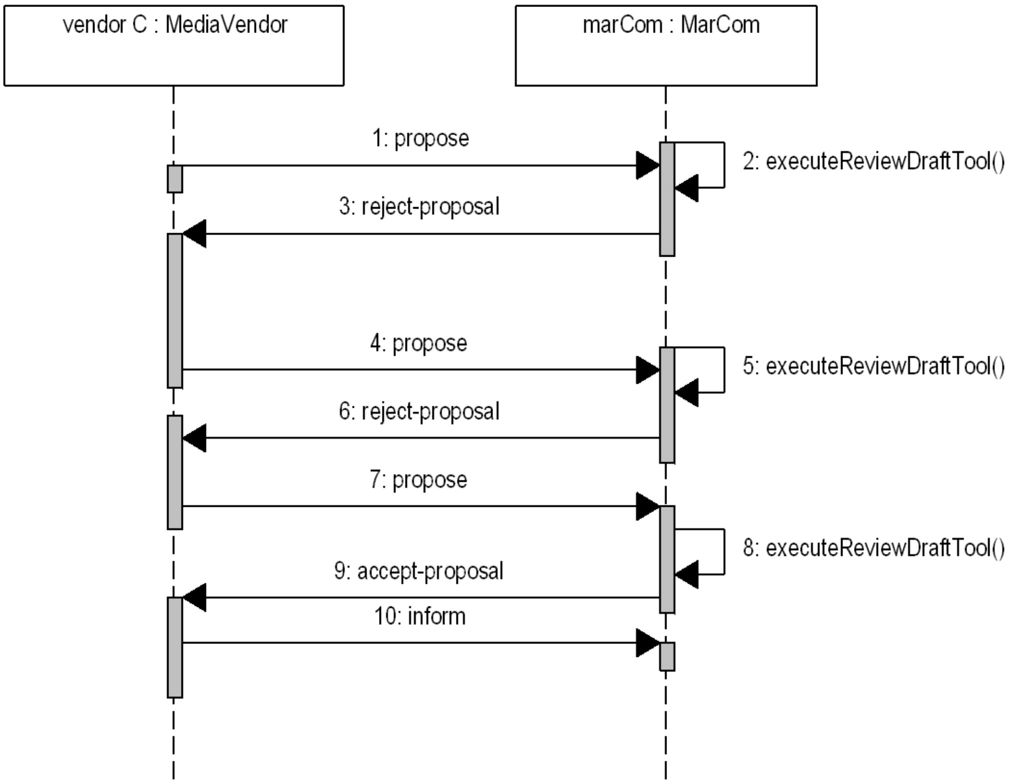


Figure 13 Main interactions within a sample instance of the ReviewDrafts workflow process.

So, in such a case the workflow metaphor can be exploited to specify a new ad-hoc interaction protocol. Figure 12 depicts the workflow diagram of the `ReviewDrafts` class which eventually produces an exchange of messages that follows the sequence pattern presented in Figure 13. In details, Figure 13 presents an iteration of the PROPOSE IP for 3 times (actually until a positive answer happens) and a final informative communication act. Apparently, by introducing a workflow class to represent the interaction protocol, an effortless yet exact mapping is possible. Figure 14 demonstrates how this mapping is achieved.

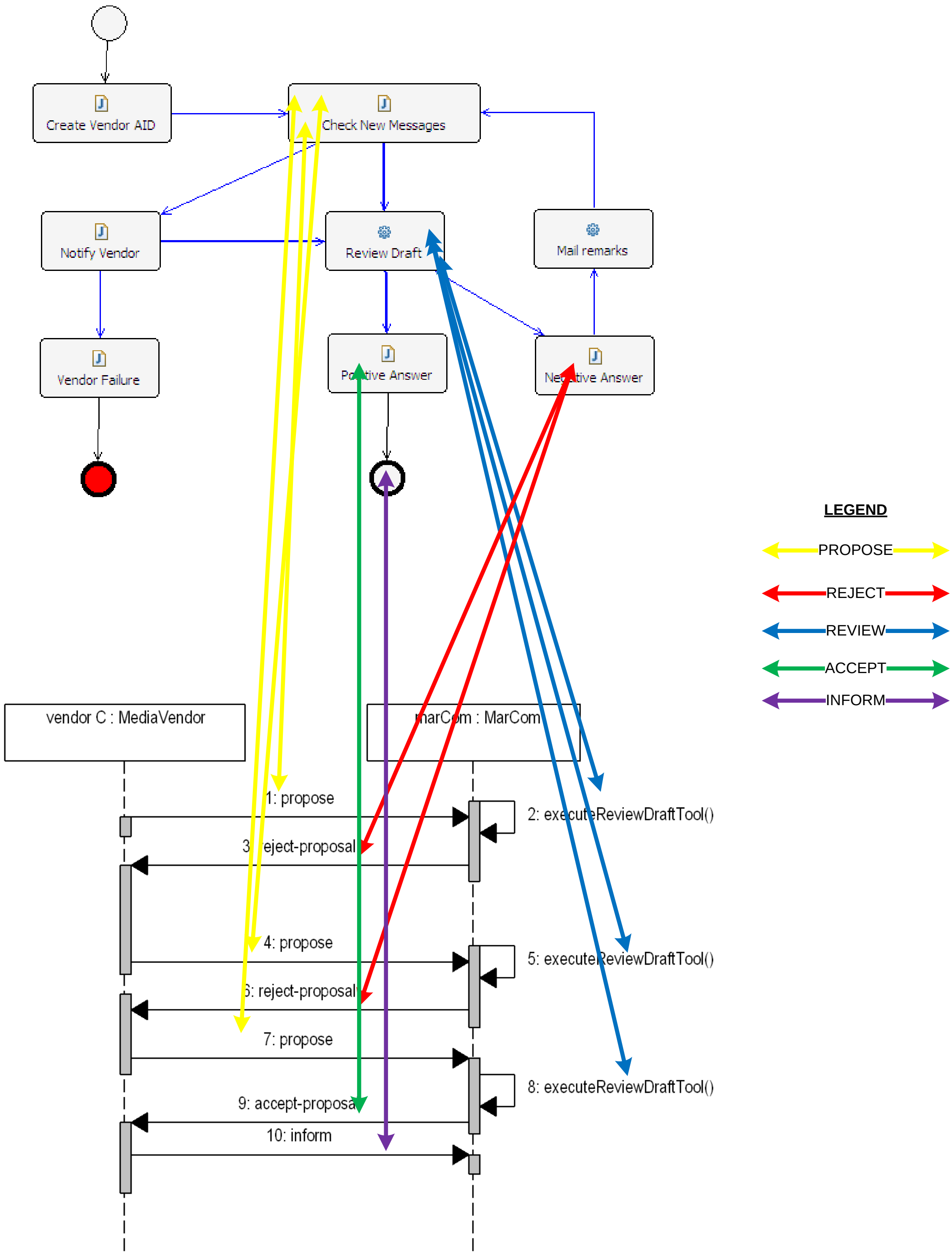


Figure 14 Mapping an ad-hoc message exchange pattern to a workflow class

4.4 Business Logic Support

The role of WFMS is not just to support the enactment of business processes but to support the definition of the workflows as well. Agent-involved workflow management systems inherent this role along with others process definition related features. During the previous chapters, the various techniques that are used in existing approaches for defining the workflow processes were described. In this section, the business logic support framework of the proposed application is presented. Two different approaches are proposed in order to better address the wide-ranging field of AWFMS. Their goal is to allow a concise business logic representation that will yield rapid and predictable development of workflow process models. The two approaches, although conceptually different, they are not mutually exclusive and can be used in combination as it demonstrated in subsection 4.4.3.

4.4.1 Rely on the Workflow Definition

This approach proclaims that the business logic is fully described in the workflow definition, which orders agents to perform any necessary actions. This is probably the most intuitive approach, which assigns every logical piece of work to an atomic activity of the definition. Every activity is related with a performer, which takes over the responsibility to carry out the task. This way, a workflow definition exploits the natural distribution of agents. In this thesis, this approach is implemented by utilizing the mechanisms provided by WADE.

4.4.1.1 Importing an XPDL document

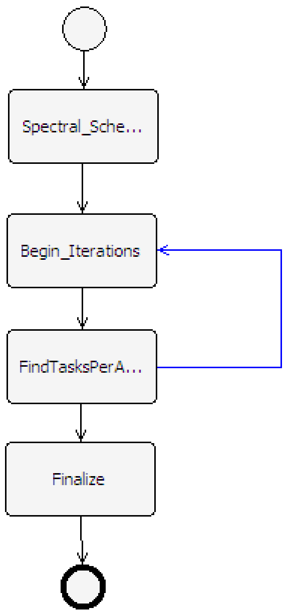
XML Process Definition Language (XPDL) is actually a process definition meta-model which provides a common method to access and describe process definitions. XPDL is an open standard [144], which enables a process definition, generated by one modeling tool, to be used as input to a number of different run-time products. So, XPDL is a format for process definition interchange - it does not force a particular process model on the execution environment. The real benefit of XPDL comes from the exchange of the design of the process. XPDL is used today by more than 80 different products today to exchange process definitions, and it is emerging as a de facto industry standard [145]. Concluding, for a workflow management system that visions to be interoperable, XPDL support is a recommended feature.

Table 3 Importing a XPDL definition

XPDL code	Resulting workflow class

```
<?xml version="1.0" encoding="UTF-8"
standalone="no"?>
<Package xmlns="http://www.wfmc.org/2002/XPDL1.0"
xmlns:xpdl="http://www.wfmc.org/2002/XPDL1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" Id="auxiliary" Name="auxiliary"
xsi:schemaLocation="http://www.wfmc.org/2002/XPDL1.0
http://wfmc.org/standards/docs/TC-
1025_schema_10_xpdl.xsd">
    <PackageHeader>
        <XPDLVersion>1.0</XPDLVersion>
        </PackageHeader>
    <RedefinableHeader
PublicationStatus="UNDER_REVISION">
        <Author>Pavlos Delias</Author>
        <Version>0.8</Version>
    </RedefinableHeader>

    <WorkflowProcesses>
        <WorkflowProcess Id="SpectralScheduling"
Name="SpectralScheduling">
            <ProcessHeader>
                <Created>2009-07-30
09:02:19</Created>
            </ProcessHeader>
            <RedefinableHeader>
                <Author>Pavlos Delias</Author>
            </RedefinableHeader>
 <Participants>
        <Participant Id="Assigner" Name="Assigner">
            <ParticipantType Type="ROLE"/>
        </Participant>
    </Participants>
            <Activities>
                <Activity Id="Spectral_Scheduling"
Name="Spectral Scheduling">
                    <Implementation>
                        <No/>
                    </Implementation>
                    <Performer>Assigner</Performer>
                </Activity>
                <Activity Id="Begin_Iterations"
Name="Begin Iterations">
                    <Implementation>
                        <No/>
                    </Implementation>
                    <Performer>Assigner</Performer>
                    <TransitionRestrictions>
                        <TransitionRestriction>
                            <Join Type="XOR"/>
                        </TransitionRestriction>
                    </TransitionRestrictions>
                </Activity>
                <Activity Id="FindTasksPerAgent"
Name="Find Tasks per Agent">
                    <Implementation>
                        <No/>
                    </Implementation>
                    <Performer>Assigner</Performer>
                    <TransitionRestrictions>
                        <TransitionRestriction>
                            <Split Type="XOR">
                                <TransitionRefs>
```



```
                                    <TransitionRef
Id="SpectralScheduling_tra4"/>
                                    <TransitionRef
Id="SpectralScheduling_tra3"/>
                                </TransitionRefs>
                            </Split>
                        </TransitionRestriction>
                    </TransitionRestrictions>
                </Activity>
                <Activity Id="Finalize"
Name="Finalize">
                    <Implementation>
                        <No/>
                    </Implementation>
                    <Performer>Assigner</Performer>
                </Activity>
            </Activities>
            <Transitions>
                <Transition
From="Spectral_Scheduling"
Id="SpectralScheduling_tra1" To="Begin_Iterations"/>
                <Transition From="Begin_Iterations"
Id="SpectralScheduling_tra2"
To="FindTasksPerAgent"/>
                <Transition From="FindTasksPerAgent"
Id="SpectralScheduling_tra3" To="Begin_Iterations">
                    <Condition Type="OTHERWISE"/>
                </Transition>
                <Transition From="FindTasksPerAgent"
Id="SpectralScheduling_tra4" To="Finalize">
                    <Condition Type="CONDITION"/>
                </Transition>
            </Transitions>
<ExtendedAttributes>
                <ExtendedAttribute
Name="StartOfWorkflow"
Value="Executor;Activity_1;100;50;NOROUTING"/>
                <ExtendedAttribute
Name="EndOfWorkflow"
Value="Executor;Activity_6;110;100;NOROUTING"/>
                <ExtendedAttribute
Name="ParticipantVisualOrder" Value="Executor;"/>
            </ExtendedAttributes>
        </WorkflowProcess>
    </WorkflowProcesses>
</Package>
```

Table 3 presents how a process definition, created as an XPDL document, can be imported to the system, and result in a workflow class. In fact, what is presented is the resulting workflow diagram. An important notice is that what is transferred from the XPDL document to the system is the process flow (activities, transitions, conditions, joins etc.). The actual implementation of the activities, transition conditions etc. shall of course be defined in the system's language. Yet, using an XPDL definition allows the system to interoperate with vendor specific tools or platforms by transferring process models via a common exchange format.

4.4.1.2 Construct a JAVA class containing the definition

Since the proposed system is a software piece, written using a programming language (JAVA), a simple way to communicate the business logic is to translate business logic into the same programming language. This way has two major drawbacks:

1. The process designer must be familiar with JAVA programming or he/ she shall work in tandem with a software developer.
2. The JAVA class developed, must adhere to a specific formalization, imposed by the underlying software (in this case WADE)

In spite of these counterarguments, constructing a JAVA class to represent the business logic is a very rich, powerful and efficient way to express business logic. In the next paragraphs, the basic steps that should be followed in order to construct a JAVA class that symbolizes a business process are explained:

Ultimately, what has to be done to create a workflow class according to the WADE formalism is to build a finite state machine (FSM) model. A finite state machine is a model of behavior composed of a finite number of states, transitions between those states, and actions. Within the JADE concept, FSMs are used to describe complex agent behaviors, defining states not necessarily as agents' internal states, but also as activities (JAVA code pieces) that the agent should implement. A WADE workflow class is an extension of a FSM, and from an UML perspective is similar to an activity diagram. Activity diagrams themselves are used to show the flow of activities through the process. Diagrams have branches and forks to describe conditions and parallel activities.

So, the first and fundamental step in constructing a workflow class is to express the business logic into activity diagram concepts, i.e., activities and transitions. Process designers are facilitated by a graphical editor so that they can visualize the mental picture of the process that they hold, and get immediate feedback on the screen of this visualization. In Figure 15, such a visualization of the "`PreparePiece`" process is depicted.

As it can be seen, the `PreparePiece` process declares that the initial activity is to take some media decisions about the marketing piece (e.g., the format of the piece – Brochure, Flyer, Catalog etc.-, the amount of the pieces that will be produced, etc.). These decisions are articulated in a document which is read during the second activity of the model ("*Read Media Decisions File*"). Next, a preparing activity transforms the articulated data into distinct requirements for every cluster of customers, and a loop begins. For each cluster, the business logic orders to solicit potential vendors that could produce the marketing pieces according to the specified requirements and after selecting one of them, to review their production iteratively until the piece artwork is approved. Finally, some mandatory tasks (such as updating the database of the system, or cleaning data) take place. In the `PreparePiece` process, the `SolicitDesign` and the `Review` activities are composite activities, containing other workflow processes. The first two activities ("*Media Decisions*" and "*Read Media Decisions File*") are realized by invoking external tools.

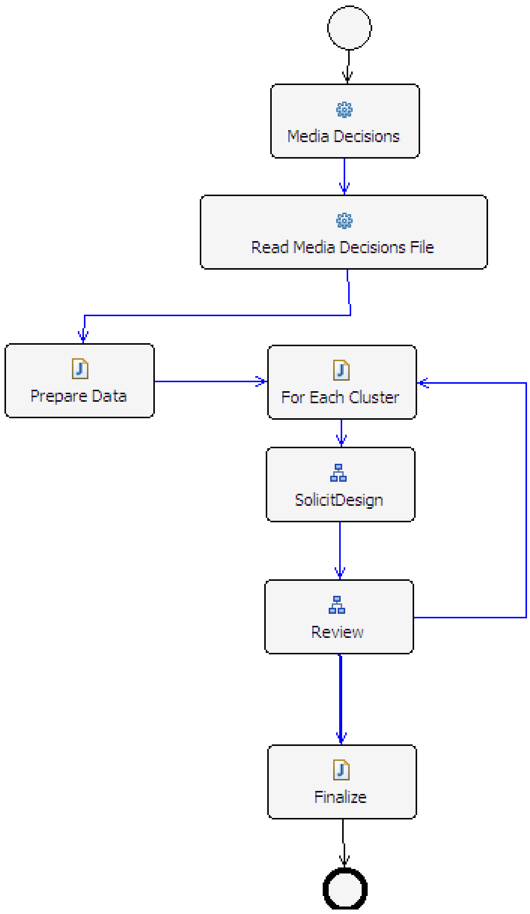


Figure 15 Workflow diagram of the PreparePiece process

The second step to construct the workflow class is to define the parameters that are exchanged between this process and the external tools or other workflow processes. The final step is to build the necessary classes for the relevant tools and workflows, so that the business logic is fully represented. Figure 16 illustrates the resulting class diagram for the `PreparePiece.java` class and the related class (tools and joined workflows). An advantage of using JAVA classes to represent the business logic is that a typical feature of object orientation, inheritance, can be exploited to create new process definitions by extending the classes of the existing ones.

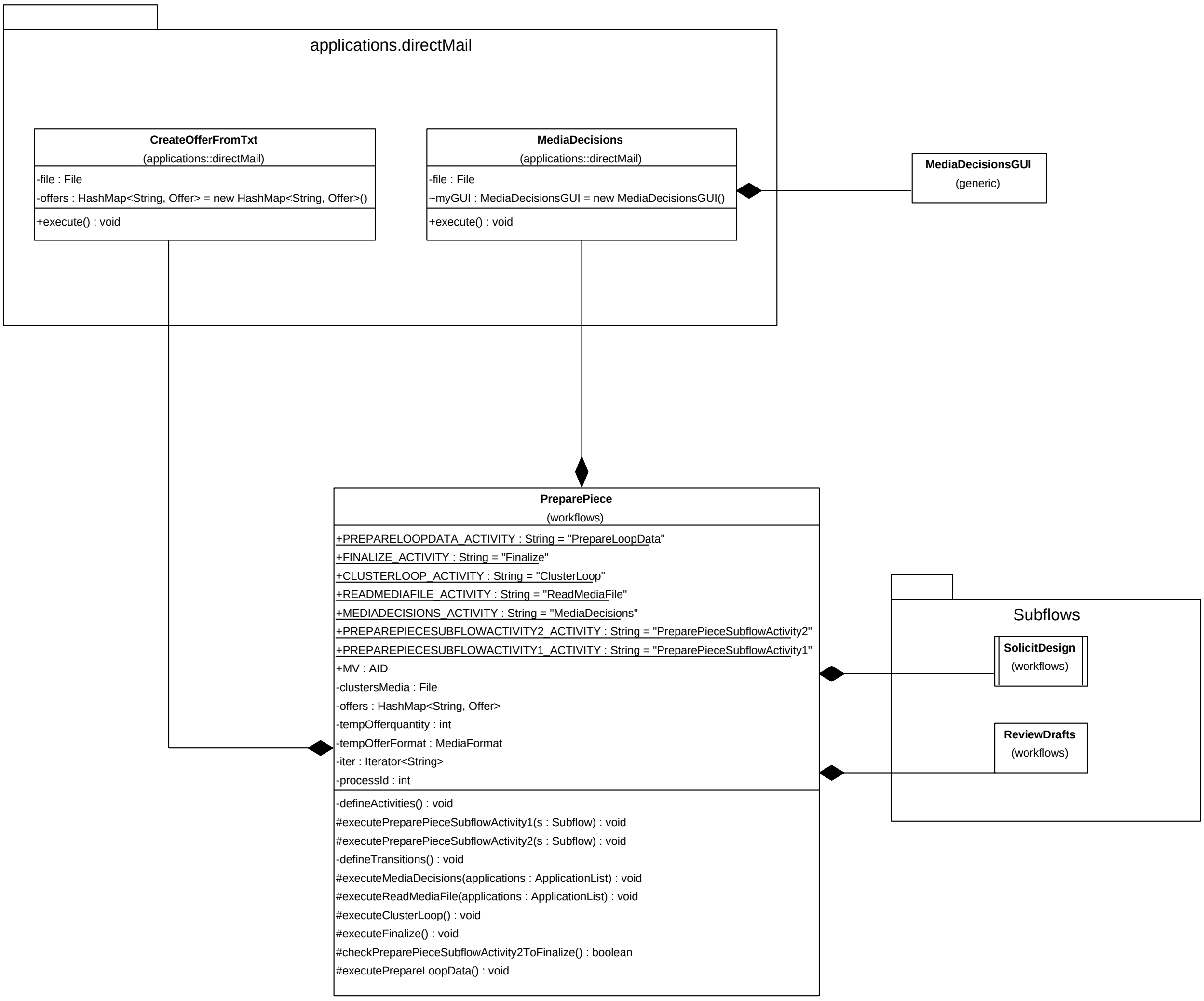


Figure 16 Class Diagram for the PreparePiece process and related tools

4.4.2 Use an Application Engine and an application specific ontology

Workflow processes are needed to be described formally and their models shall not let any room for ambiguity, subjectivity or inaccuracy. Formal process languages can achieve the above by providing a workflow definition. However, a different way to achieve these goals is to use ontology to eliminate conceptual and terminological confusion. Ontology is a representation vocabulary, often specialized to some domain or subject matter. In other words, the representation vocabulary provides a set of terms with which to describe the facts in some domain [146]. Of course, building ontology requires an additional effort, in terms of profound analysis of the kind of objects and relations that exist in the domain, but one can afford this effort by saving time from building a workflow definition using a formal process language.

Within the agent-involved workflow management systems context, if we can manage to use a domain-specific ontology to represent the aspects of a specific process, then i) we can build formal descriptions of the business logic and ii) we can support the workflow execution by feeding agents' communication and reasoning functions with the ontology concepts. In this section, the above claim is supported by an example, the `ContactCenterOntology` ontology which was used in the application developed to support the process described in section 4.1.2.

Once again the first and fundamental step is to express the business logic with the basic elements of the ontology, i.e., objects and relations among these objects. To comply with JADE formalism, objects can be one out of the following types:

- *Concepts*, which are entities with a complex or simple structure that "exist" in the world that the domain refers.
- *Agent Actions*, which are special *Concepts* pointing to actions that can be performed by agents.
- *Predicates,* which are expressions that are evaluated and can result in either true or false.

The contact center domain ontology is presented in the class diagram of Figure 17. Some explanations for this ontology follow:

- Concepts:
- `Mail`: Represents an e-mail that arrived at the system. Each mail has a specific type (available types are enumerated in the `MailType` class), an estimated duration based on its type, a timestamp denoting when it arrived and a second one denoting until when it should be served. Finally, every mail has of course its actual content.
- `MailBatch`: Actually a collection of `Mail` objects. It contains also a reference to the file where the mail elements are saved.
- `Sender` & `Receiver`: These two entities are used to declare agents that have exchanged messages. They are used for audit purposes and they are general entities (not directly connected to the contact center domain)

- `Task`: This entity represents an atomic task that has to be carried out by an employee of the center. In essence, this task is to read an e-mail and reply according to its request.
- `Worklist`: Actually a collection of tasks. The entity contains also a reference to the file where the `Task` objects are saved.
- Agent Actions:
- `ReceiveMails`: This action orders the performer agent to connect to a POP3 mail server and get the mails that have arrived. The connection attributes (username, password, and server) are also attributes of the class.
- `SendMailBatch`: This action specifies a list of mails and a receiver agent. The receiver agent gets informed about all the mails that arrived during the current time window.
- `Read`: The performer agent reads the file which is specified by this action
- `AddWorklist`: This action has two attributes, an agent and a worklist. The performer agent publishes the worklist and announces the agent that should perform it.
- `Todo`: This is an action of assignment. The performer agent assigns to another to do a specific task (reply to a batch of mails) specified in the `item` attribute.

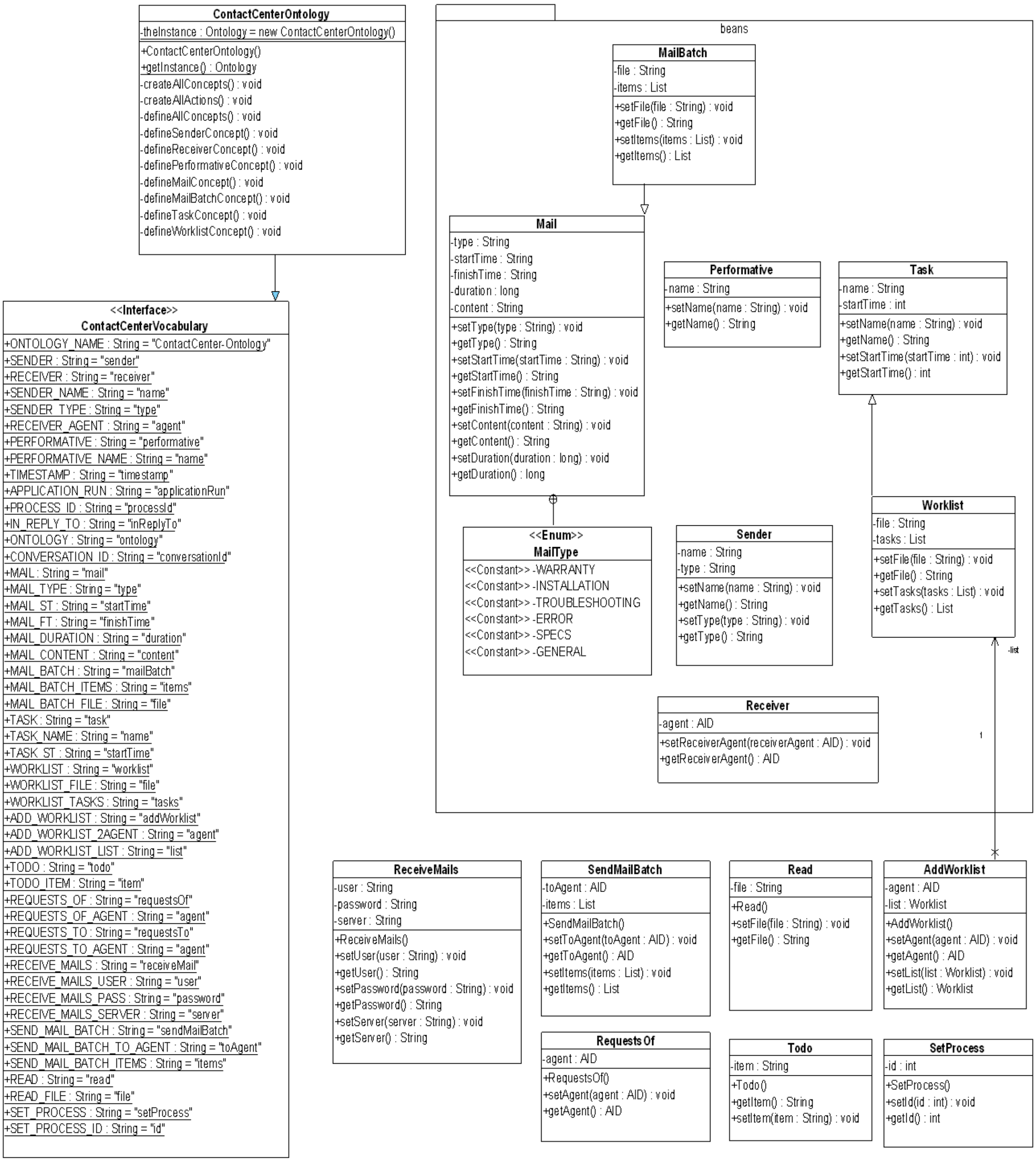


Figure 17 The Contact Center Ontology

Still, how does this ontology support the actual execution of the business process in an AWFMS context? The truth is that the ontology alone is not capable of such a thing. It has to be combined by another feature of agenthood: communication. Through message-based communication an agent can include in the content-slot of the message an agent action, so that the receiver agent, by receiving the message is ordered to perform that action. A basic prerequisite for this is that both agents do understand, and are able to interpret the same ontology.

In the application developed the following pattern is adopted: A central agent, called `ApplicationEngineAgent` is responsible for hearing requests that concern actions related to the contact center ontology. Such requests can be sent by any agent of the system (e.g., often the `GUIAgent`). After receiving a message of the `ContactCenterOntology`, the application engine agent serves the requested action. Serving an action for the application engine means that either it performs it by its own, or it delegates it to another more appropriate agent. The whole procedure is based on FIPA interaction protocols (see Section 4.3.1). An example of ontology-based workflow execution is illustrated in Figure 18.

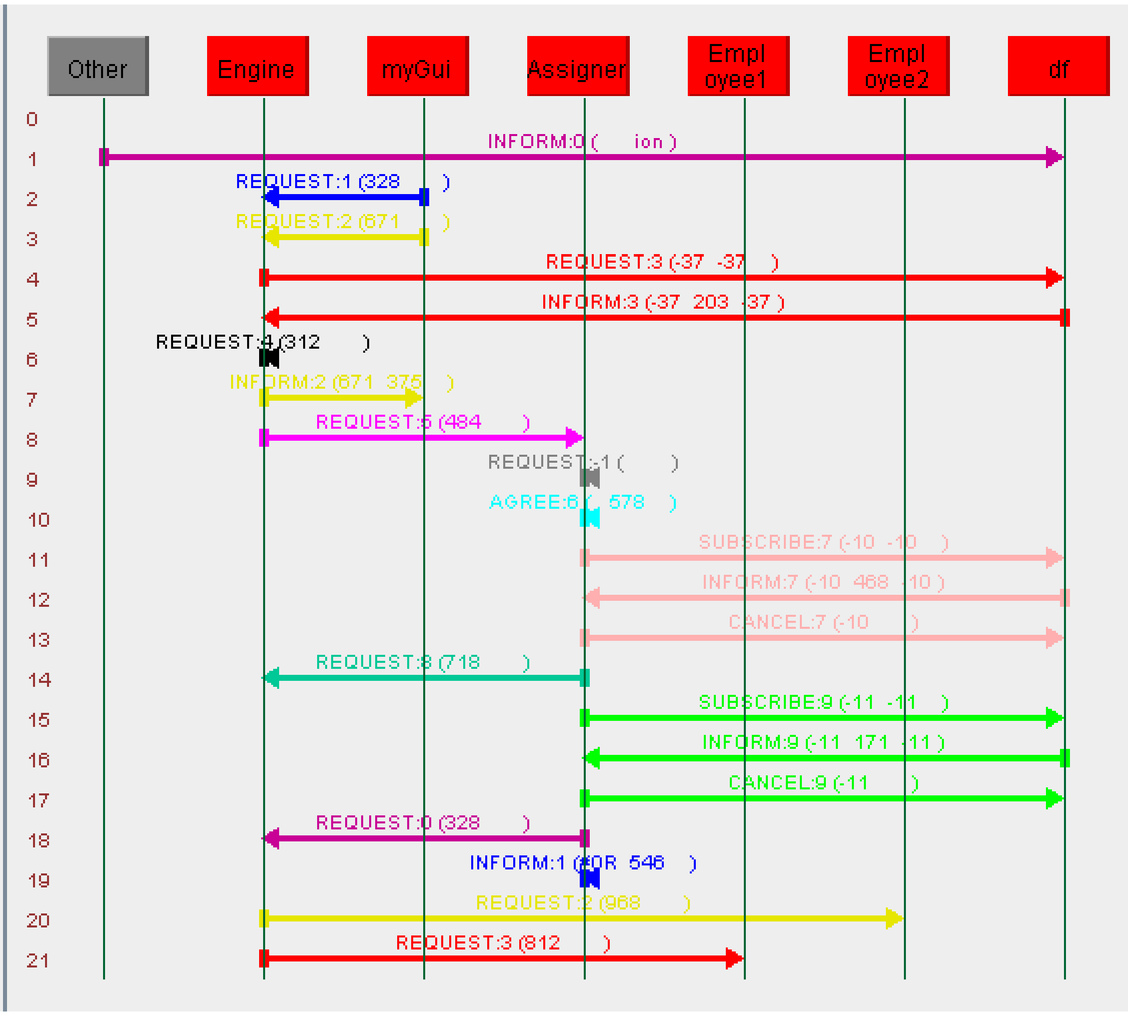


Figure 18 Messages exchanged during the ontology-based workflow execution ***(Source: Application runtime – JADE Sniffer Agent).***

In Figure 18, a total of 20 messages are exchanged to achieve a single iteration of the workflow process (not counting the 1st message which is irrelevant with the process). More specifically:

- Message 2: The `GUIAgent` (myGui) request from the `ApplicationEngineAgent` (Engine) to perform an action (set the process Id to the current process' id).
- Message 3: Like message 2, but the action this time is the `ReceiveMails`.
- Messages 4 & 5: The engine talks to the Directory Facilitator (df) to get the address for the `AssignmentAgent` (Assigner).
- Message 6: According to the business logic, the engine asks from itself to self-perform an action (prepare a mail batch to be sent)
- Message 7: The Engine sends a notification to myGui to inform him that his request is served. This notification is send to keep accordance with the FIPA interaction protocols. Due to agents' autonomy, there is no exact schedule of when this kind of messages are sent.
- Message 8: The engine requests from the assigner to read the file he prepared (serve the action `Read`).
- Message 9 & 10: The inner logic of the `Read` action is to execute another workflow process. So, the assigner requests from himself to perform another workflow and gets a positive reply (Once again, the reply is used to comply with FIPA protocols)
- Messages 10-13: The assigner talks with the df to get informed about the address of the `ApplicationEngineAgent`. A similar procedure was followed during the initialization of the assigner to get informed about the available employees. This procedure is recommended, because agents may move to different nodes during the workflow execution.
- Message 14: The assigner requests from the engine to add a worklist to an employee
- Message 20: The engine, following the previous request, asks from the employee (Employee2) to carry out the specific worklist (`Todo` action).
- Messages 15-17, 18 and 21: Similar with the exact three previous steps, just altering the name of the employee and its assigned tasks.

4.4.3 Business logic support using both methods in combination

In the previous paragraphs two different approaches to support business logic in AWFMS were presented: Relying on the workflow definition and ontology-based workflow execution. Both ways are accurate and powerful, and it depends on the business logic needs to choose which one to implement. Yet, there is a possibility to use both in combination, in order to tackle any special process needs. Actually, an example of this case was described in the previous paragraph (messages 9&10 of the ontology-based workflow execution - Figure 18). In that case, there is a workflow class (`SpectralScheduling`) which defines the workflow logic of a specific sub-process (provide a schedule for the tasks, considering the available resources). This workflow class makes no use of the contact center ontology, however while serving an action of the ontology, the workflow class is invoked by the agent who performs the ontology specified action. So, ontology is used to achieve high-level coordination and business logic support, while low-level operations are prescribed within workflow definitions, which in turn are attached as ontology actions' components.

4.5 Manual Intervention

The term workflow signifies the automation of a business process which is defined within a process definition. Workflow management systems are supposed to guarantee that during run time, every process is executed according to its definition, typically with little or no human intervention. Nevertheless, there are circumstances that a strict, automatic execution of the definition does not produce the desired outcome. There are some exceptional circumstances that the user needs to override the initial definition and manually change the execution path of the process. For instance, the user may detect invalid data in the process input data, or new information may have became available, so the process needs to rewind and resume execution from a previous step. Moreover, in a business environment, special events emerge (e.g., an ad-hoc agreement with a special customer) that may lead to different process rules (e.g., a document is not delivering or a deadline is getting loose). Ideally, the workflow administrator should have some tools to handle these exceptional circumstances, and manually specify the activity node that the system should execute next.

This lack of flexibility and the non existence of manual intervention support has been early identified as a limitation of workflow management systems [30]. Systems that didn't provide this functionality were noticed to irritate end users, who felt that the systems were merely enforcing rigid rules [53]. Manual intervention can be expressed by many ways: performing the tasks manually, skipping some tasks, modifying the control flow, rewinding and repeating some tasks, providing manually values to evaluate conditions etc.

In this thesis, manual intervention implies that a user can choose a specific point of a process, and start execution from that point. Moreover, he/she can also choose to execute just a special part of the process and not the entire workflow. To succeed in allowing this, the notion of "*state*" is incorporated. The concept of "*state*" is analogous to a milestone within a workflow. Typically, a milestone indicates the end of a stage and it goes together with some specific deliverables. Thus, if there is a need to check if the milestone is reached, it is sufficient to check if the deliverables are okay. This abstract idea is adopted in the proposed system. In particular, the process designer indicates a limited number of states that roughly split the workflow process into phases. A state is actually the interval between two milestones: one indicating the starting point and the other the finishing point. Often the finishing point is the process end. Following the procedure, the designer associates a set of "*requirements*" with every state. If the requirements are indeed accomplished, the user may begin workflow execution from that particular state. As it will be described in section 4.6, a "*requirement*" is a synonym for file. This technique allows end users to:

- Skip any number of activities, by providing manually the expected deliverables
- Rewind workflow execution to a previous step and repeat process execution for a number of times
- Intervene to the outcomes of the workflow without obstructing the process execution, by manually modifying the requirements' files.
- Execute just a part of the workflow, asynchronously if allowed by the business logic

Consider for example the "`directMail`" workflow, described in section 4.1.1. The states identified are:

- "`NOT_STARTED`". The process instance has been created but it hasn't started execution yet. It may be used to signify that a process id has been assigned to the instance but no other action has been performed (e.g., workflow assignment)
- "`ESTABLISH_MARKETS`". This is the initial state of the workflow. The workflow has been assigned and it is ready to start execution. The whole process will be executed.
- "`SEGMENTATION`". The process instance will start execution from the segmentation point, that is, it skips the "`EstablishTargetMarkets`" step.

- "QUANTIFY_TAM". Starts the process from the quantification of the total available market point. The steps of "EstablishTargetMarkets" and "Segmentation" are skipped.
- "BUDGET_RF". Begins executing the budgeting of response factor. All the previous steps are skipped.
- "PREPARE_PIECE". This state refers to the second phase of the process and if selected, it orders to skip the entire marketing research phase (which includes the states described previously).
- "LAUNCH_CAMPAIGN". This state orders that the two first phases (marketing research and prepare piece) should be both skipped.
- "SINGLE_SOLICIT_DESIGN". While all the previous states indicate that the process instances should start execution from a specific point and continue until the whole workflow is completed, this state (along with others that hold a prefix "SINGLE_") indicate that just a part of the work should be executed. This particular state refers to soliciting vendors to design the artwork for one marketing piece.
- "SINGLE_REVIEW_DRAFT". A state that applies the reviewing of the artwork of one marketing piece and then terminates.
- "SINGLE_CREATE_JOB_SCHEDULE". This state refers to the CreateJobSchedules class that the product manager implements to create work schedules for every group of assistants.
- "SINGLE_ASSISTANT_LAUNCHING". This state is about the execution of a task by one assistant. The reason to create such a state is that assistants may execute their assigned task at a different time, and asynchronously publish the results of their work.

When a state is selected as the starting point of a workflow execution, a requirements check is performed. If this check returns a positive answer, then the user is able to intervene to the process by altering the process starting point. This procedure is explained in greater detail in section 4.6. The system assures that all states are related to the correct process instances through a process id, which is passed as a formal parameter to all the workflows and sub-workflows that correspond to a state.

Manual intervention may provide the AWFMS with flexibility, but it incurs an added risk and cost. The risk associated with manual intervention is that when you override the process definition with a subjective – manual manner, there is no guarantee that the resulting process will be valid and sound. Moreover, when the requirements are fulfilled manually, there is also no guarantee that they have the appropriate content format or that they comply with the specified business rules. These factors make more error-prone the process instances which were manually mediated. The additional cost is related with the poor logging of manual activities. Since manual actions escape the system monitoring, auditing and backtracking become no longer possible for those particular instances.

4.6 Statefulness through Document-Centric Stigmergy

Statefulness refers to the capability of maintaining the status of a process, recognizing at any moment what has been accomplished and what is yet to come, or at least what is coming next. In the workflow management context, wrapping stateful behavior is an innate requirement, which becomes crucial in case of long lasting workflows.

Two general modes to integrate this workflow functionality are popular [147]:

- The system determines the next task by querying the data contained in the process instance itself. The system is unaware of the tasks that are already realized and of the tasks that may follow. All state information is contained within the process instance. Thus, the instance's data needs to indicate who is assigned to that unit of work, and all history information about what happened in the past. Examples of this style of implementation in an AWFMS context can be found in [76, 78, 93]
- The system knows everything about the process instance, and the instance itself doesn't contain any history or "stateful" information. In [54, 96] this general implementation style is followed.

However in this thesis a different approach is proposed. This approach, presented in the following paragraphs, can be characterized as a "*document-centric stigmergy*", a novel term, introduced here. Firstly, the use of "*stigmergy*" is explained:

Stigmergy is formed from the Greek words "*στίγμα*" (stigma – sign) and "*έργον*" (ergon – action), and it was coined in the 1950's by Grassé, a French entomologist who used the term to describe the indirect communication taking place among individuals in social insect societies [148]. Stigmergy captures the notion that agents' actions leave signs in the environment. Thus, if all agents are capable to understand and interpret these signs, they will determine their subsequent actions in such a way that the emergent behavior of the system is the desired one. Stigmergy has been used as an optimization tool by a plethora of researchers [149], exploited mainly as a simple yet effective mechanism for agents' coordination. Nevertheless, the approach proposed here does not follow the strict formulation, as described in [149]. It rather uses the conceptual initiative of stigmergy to construct an organic design for workflow management. Actually, although the mechanism of stigmergy is mostly popular in insects societies, its original concept has indeed been analyzed as a coordination framework for collaborative activities in other environments as well [150] (e.g., humans [151] or software agents [152]).

In general, in order to apply a stigmergy mechanism the following elements should be considered [151]:

- An *environment*, which is described by a *state*
- The *dynamics* of the environment, which governs the evolution of its state over time
- The agents' *sensors* that allow agents to interpret the state of the environment
- The agents' *actuators* that allow agents to modify the environment
- A *method* that configures agents' actions based on the sensed state of the environment.

In the proposed document-centric approach, these elements are defined as following:

- **Environment:** The environment should be directly related with the process instance, and its state shall exhibit the current execution state. By setting the environment to the process instance itself, a milestone in the process definition can be used to declare the environment's state. For this purpose, the notion of "*state*" which was described in section 4.5 can be exploited.
- **Dynamics:** States follow one another according to the process definition. Yet, a state can not begin unless its requirements are fulfilled. These requirements are the core of the document-centric approach. More specifically, a document (or file in general) is an atomic piece of work of a process. Every document corresponds to the results of one (or more) atomic activity, but the inverse does not necessarily happen, since there may be some intermediate activities which do not need to be stored to a file. However, storing results in a document is the only way of saving process instances' data permanently. Documents are saved

during runtime (process execution) and usually they follow a particular template. Thus, every document is a partial deliverable of a process instance and has a specific time point when it is delivered. Each state comprises a set of documents as it prerequisites. These documents are state requirements, and they are specified by the process designer during build-time.

- **Sensors:** Documents' paths are stored to a database. Agents (workflow performers) query the database to learn which requirements are fulfilled for a particular process instance
- **Actuators:** When an agent performs a workflow, upon successful implementation of some work units, it updates the database.
- **Method:** Agents perform a workflow according to its definition. They sense the environment, interpret the signs and begin execution from a particular point (state). They know what they should execute next since the can interpret the process definition and realize the point at which the process instance exists.

4.6.1 A supportive database schema

An important capability of workflows is that they can be persisted (saved and reloaded at a later time). Workflow persistence is especially important when developing applications that coordinate human interactions, since those interactions could take a long period of time. But persistence is also applicable to other types of applications. Without persistence, the lifetime of workflows is limited. When the application is eventually shut down, any workflow instances simply cease to exist. Workflow persistence means to save the complete state of a workflow to a durable store such as a database or SQL file.

Nevertheless, the database schema is an important aspect of the application. In this section, a schema that is capable to support the *document-centric stigmergy* approach is proposed (Figure 19). Save the "`monitor_details`" table which is used for monitoring reasons (see Section 4.7), the rest seven tables are exactly the tables that are needed to store workflows according to the document-centric approach. In particular, each workflow model has a specific *process type*, which corresponds to its definition. Process types are stored in the `process_type` table which needs to contain just the name of the process type (and maybe a short textual description). As discussed in section 4.5, for every process type, the process designer indicates a few "milestones" within its definition. Each milestone corresponds to a "*state*". Thus, the `state` table is incorporated. Every state is related with a specific process type and a workflow class that should be initiated upon the state's activation. Workflow classes are actually the process definitions and they are stored to the `workflows` table, along with a hint of what is the appropriate performer type. An important notice is that the database needs not to store any additional information (e.g., regarding the flow of the activities, or the performers' types hierarchy) since this piece of information is hard-copied either into the body of the agents, or into the modular components of the application (e.g., workflow classes maybe deployed by their .jar files).

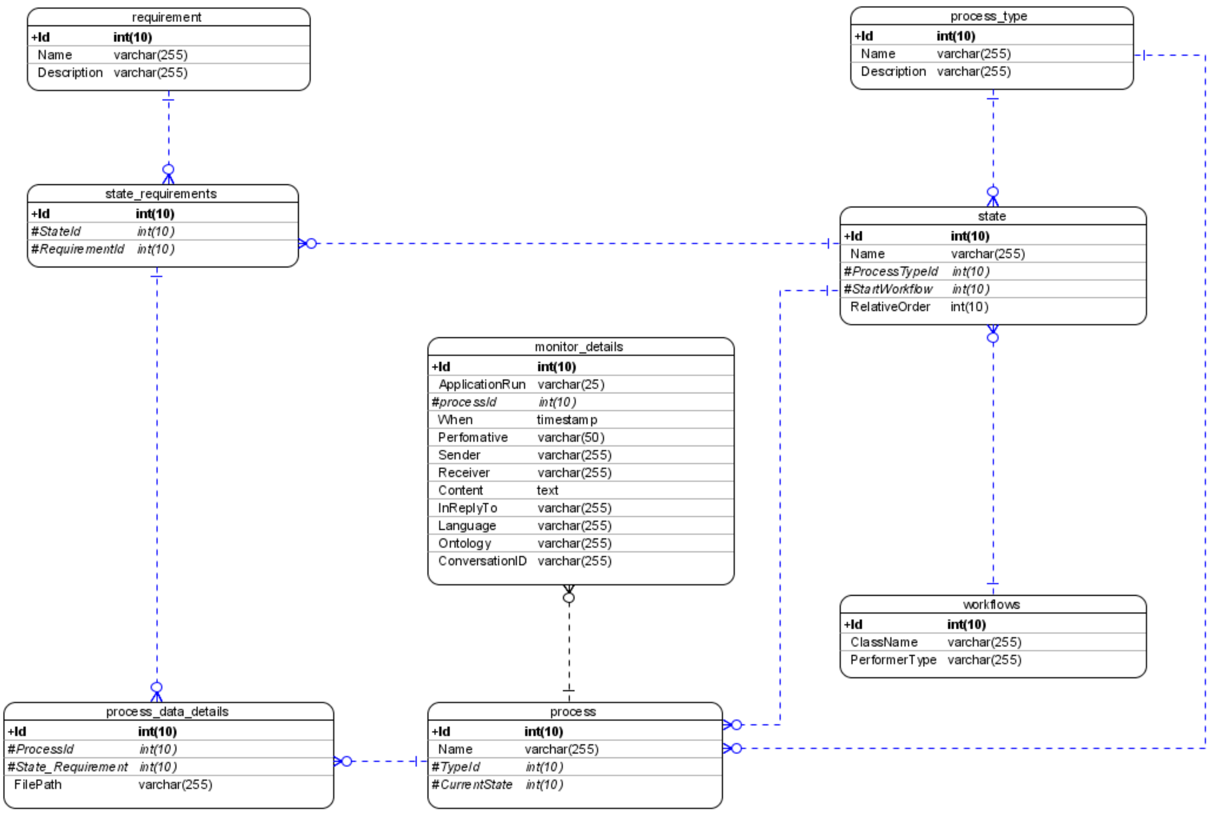


Figure 19 The proposed database schema.

The `process` table refers to the process instance and it is used to track its execution details, which are actually stored in the `process_data_details` table. As mentioned in the previous section (4.6), the execution details (not referring to the monitored elements) are documents (files) that are delivered during the runtime. The `process_data_details` table is used to store the relative file paths. Every file is a "*requirement*", and as such it is defined within the `requirement` table. Finally, the `state_requirements` table is used to model an `m-to-n` relationship between the requirements and the states, that is every state may have zero or more requirements while a requirement may belong to one or more states.

The great advantage of this schema is that is minimal respective to the application needs. It fully exploits agents' statefulness and the application's programming language to avoid storing large volume of data. Agents (as workflow performers) are fully conscious of what is the workflow they are executing, which activity follows next, what conditions will allow the transition to which activities, to whom they may delegate a piece of work, what is their type and role and where they should address in order to get informed about other agents or process related data.

For this advantage to become more evident, Figure 20 illustrates a database schema that would be needed if the agents awareness was not exploited and process definition were not hard-copied as JAVA classes, but they were stored to the database. The tables shaded in blue are the tables used also in the minimal schema. Although the schema of Figure 20 is not the only one that can respond to the issues mentioned in the previous paragraph, it becomes apparent that unless we exploit agenthood and a stigmergy approach, a significant overhead is added to the database, regarding process definition data, execution auditing activities, participants' hierarchy and workflow implementation details.

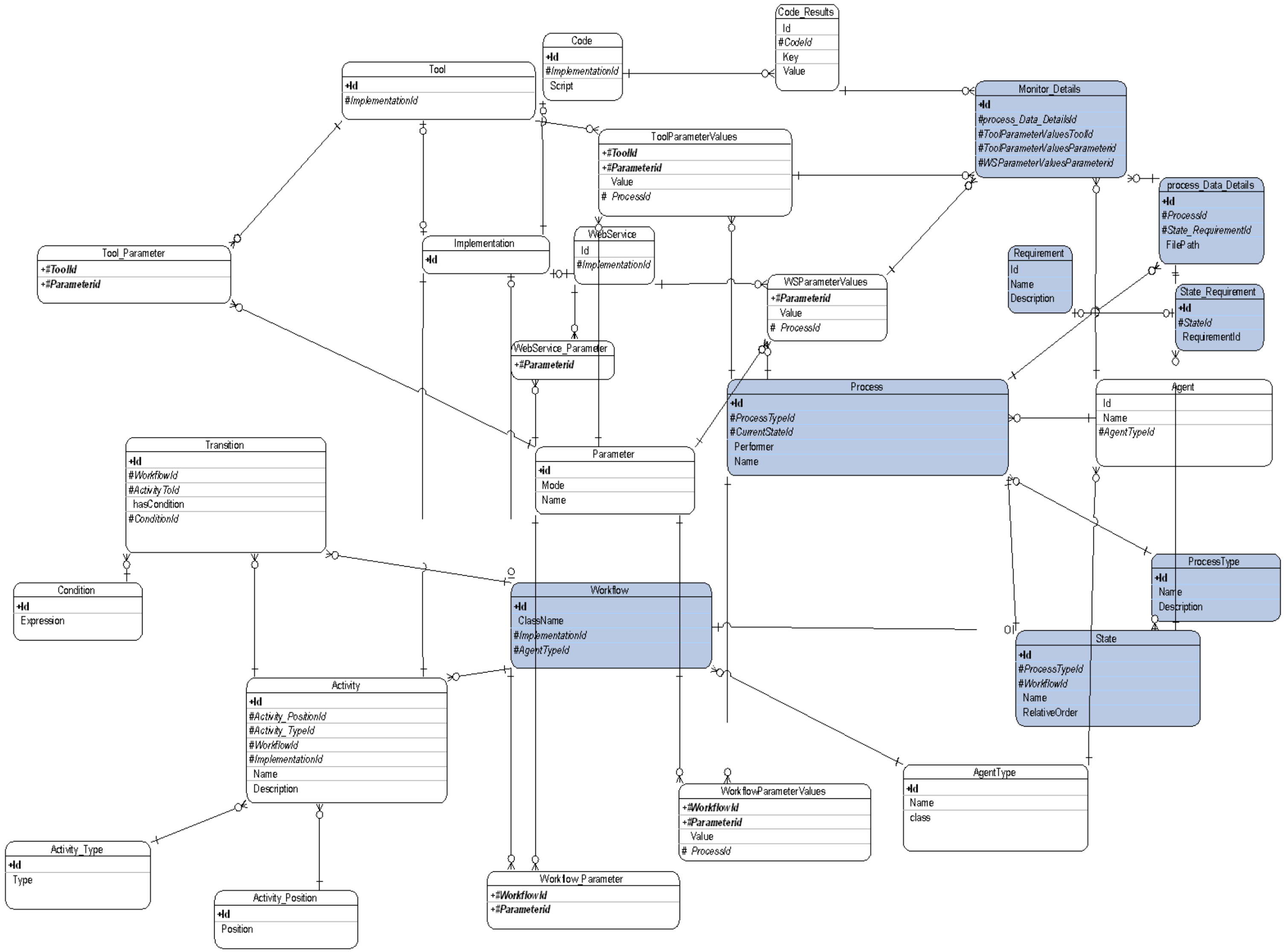


Figure 20 A database schema which does not exploit application's features.

4.7 Process Monitoring & Auditing

4.7.1 Why is it important?

Process monitoring and auditing in agent involved workflow management systems include different tasks as described in section 3.2.5. In this thesis, these activities are considered to be related with the tracking and the recording of log files, semantic and not semantic. Logging provides a way to capture information about all the operations that take place within the application. Once captured, the information can be used for many purposes, but it is particularly useful for evaluating the application logic, auditing its statistics and solving problematic issues.

4.7.2 Implementing the monitoring component as a kernel service

The monitoring component should be developed as a distinct manageable and comprehensible module, adhering to the *separation of concerns* concept. In order to comply with the approach of separation of concerns supported by JADE, the composition filters approach is adopted [153]. The general idea of composition filters is that each object is provided with two filter chains: an incoming and an outgoing. The incoming chain uses the filters on the incoming messages while every outgoing message is filtered before entering the outgoing queue.

The way that JADE uses to implement this approach is through a *Service Manager*. A Service Manager resides in every node of the Platform (actually the Service Manager is inherently present in the node that hosts the Main Container, while in the other nodes there are Service Manager proxies), and it manages the activation of all the possible services that are registered to the platform. Therefore, following this principle, the monitoring component is developed as a special service (`MonitoringWFService`), so that it can be smoothly integrated into the platform architecture.

The `MonitoringWFService` has an ultimate goal of recording the semantics of every message exchanged in the platform. In order to support the debugging and the auditing of a process, the elements that are recorder are:

- A unique id of the application thread, in which the message is exchanged. This parameter express a single run of the application, and it is of course the same for all the messages created during that run

- An id of the process during which the message is created. If the message does not concern a specific process (e.g., concerns the platform initiation or the import of a configuration) then this field is set to 0.
- A timestamp of the moment that the message is exchanged
- The performative of the message
- The sender agent
- The receiver agent
- The actual content in string format
- The "`inReplyTo`" element which indicates if the message is a reply to another one
- The language used to encode the message
- The ontology based on which the message is created
- The conversation id, which indicates if the message is a part of a particular conversation.

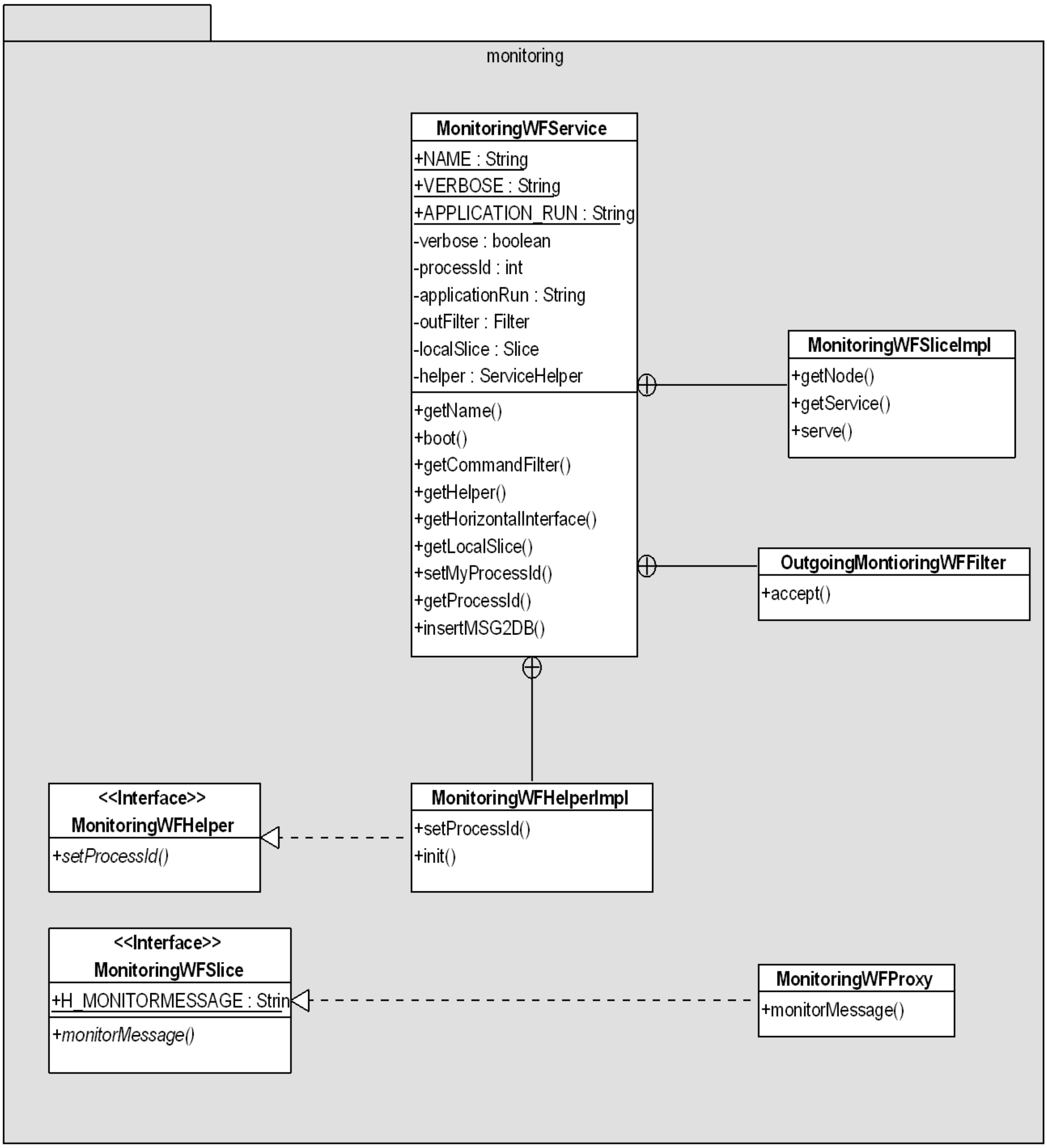


Figure 21 Class Diagram of the monitoring package

The `MonitoringWFService`, upon initialization, it accepts the parameters of the running platform profile. As mentioned earlier (Section 4.2), the configuration parameters are set in a properties file, during the build time. The parameters related to the monitoring component are:

- The VERBOSE parameter (`marketingWF_monitoring_MonitoringWFService_verbose`), which configures the logging level. VERBOSE can be true or false, when it is true, all messages are also printed to the standard output of the application.

- The APPLICATION_RUN parameter (`marketingWF_monitoring_MonitoringWFService_applicationRun`), which associates every run of the application with a unique id, so that log data can be grouped along this variable as well.

The `MonitoringWFService` contains also an outgoing Filter as an inner class, which specifies the `accept()` method as it can be seen in Figure 21. The `accept()` method employs all the service logic, i.e., records every exchanged message to the application's database, according to a predefined schema. The service is accessible to agents by a special Helper (`MonitoringWFHelperImpl`) which is used by agents to indicate the id of the current process. Figure 21 also depicts some additional classes: `MonitoringWFProxy`, `MonitoringWFSliceImpl`, `MonitoringWFSlice` (interface). These classes are used to capture every message exchanged, regardless of the container in which the agents that generated them live. More particularly, when the monitoring service needs to interact with a remote container, it previously retrieves a proxy of the service slice in that container and then it calls the required methods. An illustrated example of how the monitoring service behaves is presented in Figure 22. When an agent sends a message, a (defined by JADE) command `SEND_MESSAGE` is generated. This command is transferred vertically to all the outgoing service filters. Every filter invokes its `accept()` method and if it returns a positive answer, the command is forwarded to the outgoing sink where it is further processed. If the operation is to be performed by a remote agent, the outgoing sink delegates the command as an `HORIZONTAL_COMMAND` to the service slice proxy at the remote node, which in turn delegates the command to a target sink.

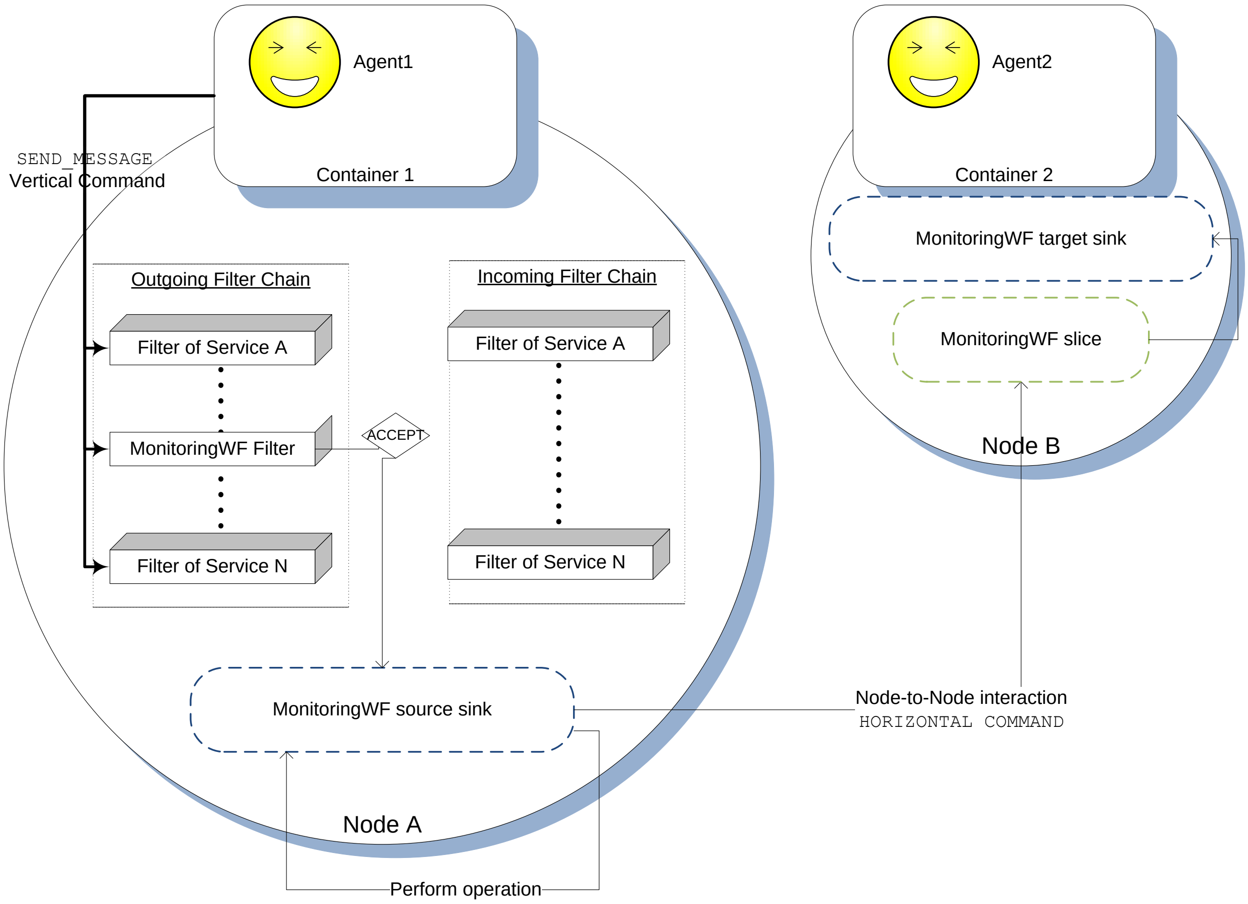


Figure 22 Basic behaviour of the monitoringWF service

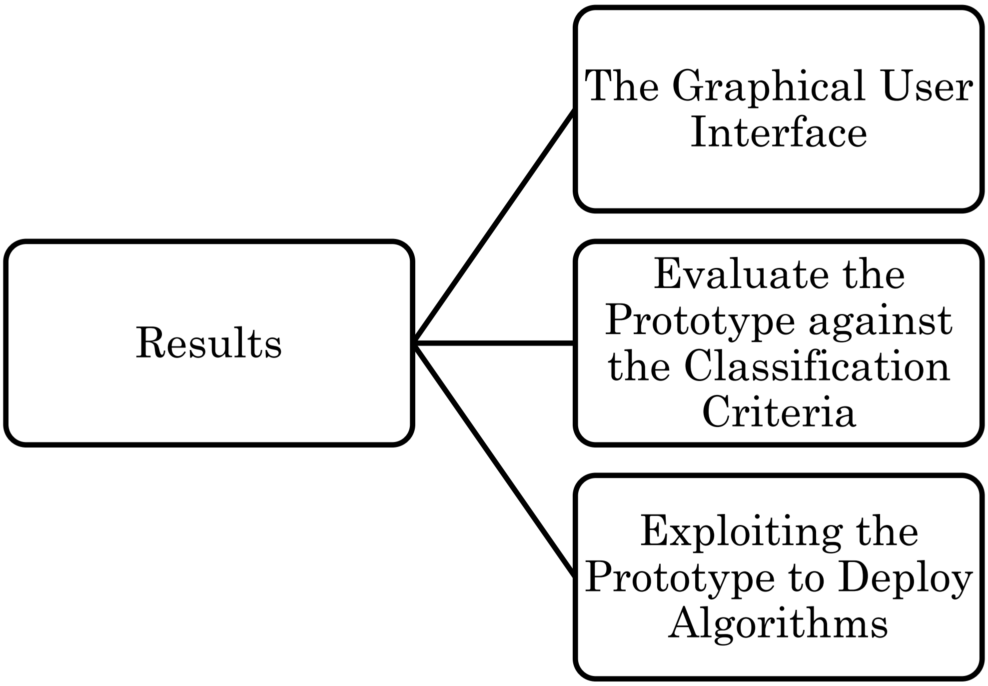
An extra tool which is also related with the monitoring function is the facility to save a text file containing the log data produced during the runtime and printed to the standard output device. Yet, log data use mostly human-interpreted expressions of no formal semantics, so the produced file can only be read by humans and does not allow directly any kind of automatic evaluation. This function will be presented in greater extend in the next section (see 5.1.5).

4.7.3 Benefits and Cost

Monitoring, implemented as a kernel service can generate and record detailed information of semantic essence about the operation of the application. It is a totally automated procedure that requires no human intervention (agents undertake the whole effort). The information is stored to a database, so it can be evaluated on a later time. The evaluation is supported by the audit trails, as the elements recorded are detailed and properly formatted. Moreover, since errors in the workflow execution are also announced by messages, the monitoring component can capture this kind of errors, supporting the troubleshooting of the application. Debugging is supplied with an extra tool as the monitoring component addresses the multi-threaded and distributed nature of the application, a nature that is not often addressed by debuggers. Finally, as the monitoring service records the messages, it does need any maintenance with the surrounding code and does not need any adjustments when the agents' code is modified.

Of course, the above benefits come at a cost: The monitoring adds runtime overhead, from capturing every message and from registering it at the database. This limitation can be critical if resources are limited. This is why the `MonitoringWFService` can be de-activated before launching the platform if one just removes the relative line from the configuration file. De-activating the `MonitoringWFService` has no other effect in the application besides the lack of recording the messages to the database.

CHAPTER 5


Results
The Graphical User Interface
Evaluate the Prototype against the Classification Criteria
Exploiting the Prototype to Deploy Algorithms

5 Results

The features discussed in the previous section reveal some of the advantages of mixing software agents and workflow management systems. To support these features' elicitation a prototype system has been developed. It is an incipient version of a workflow management system and it can be used for any of the following reasons:

- Developers and end-users experimenting with the prototype to see how the system supports their work.
- Developers and end-users acquiring a concrete impression of the system's capabilities.
- The prototype may serve as a basis for deriving a system specification.
- Facilitate rapid software development to validate business logic requirements.
- Operate as an experimental test-bed to test specific algorithms or/and provide the general context to test the integration of supplementary modules and services.

The prototype application is presented analytically in the following subsections.

5.1 The Graphical User Interface

5.1.1 Starting the application

Upon starting the prototype application, a graphical user interface becomes visible to the user (see Figure 23). The application uses the tabbed pane philosophy, that is, it employs a distinct windowpane for each type of actions that are needed to be performed. In the prototype version three conceptual sections are identified: Platform related actions, Workflows related actions and management actions. An additional menu is available at the top of the window, to control some general actions.

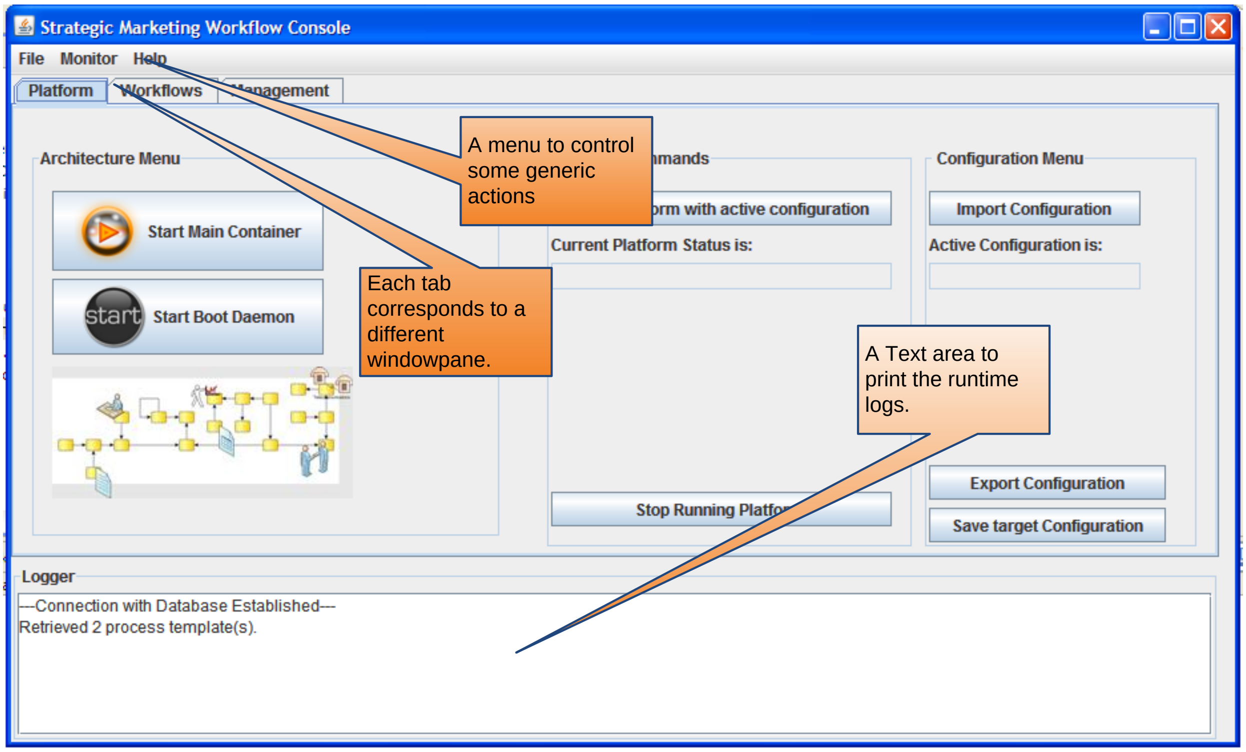


Figure 23 The application's starting screen

The application window is split in two horizontal parts (see Figure 23): A windowpane that contains all the necessary controls for each tab (buttons, textboxes etc.) and a quite large white text area, called "*Logger*" at the bottom of the window. The *Logger* is visible at every tab and it is used to capture the application's standard output, i.e., to print to the screen the runtime logs. One can control what is and what is not printed to the *Logger* by adjusting the log commands of the application's code.

5.1.2 Platform related actions

Although the prototype is a standalone application, in order to operate as a workflow management system, it is necessary to activate the multi-agent platform. As described in section 4.2, the underlying multi-agent platform is WADE. Thus, there are some standards actions, prescribed by WADE, that are needed to be performed. These are the following:

- **Start the Boot Daemon.** A single button to perform this command in provided within the platform pane. The daemon is activated taking as arguments the agents types file (types.xml) and the root configuration directory. Once the Daemon is started, the button is disabled.

- **Start the Main Container.** The main container is the core container of JADE (see section 4.2) which contains the AMS (agent management system) agent, the directory facilitator and the configuration agent. Additionally, when the relative button of the platform pane is pressed, besides the JADE platform that is being initiated according to a configuration file, an agent that accompanies the graphical interface (`GUIAgent`, see Appendix) is started as well in another container.

After performing the two above actions, the platform should be on and working. However, in order to start a specific workflow claim, a domain application, some supplementary actions are required:

- **Importing a platform's configuration.** A platform's configuration is a file that indicates how many and which containers are active in the platform, where each one resides (in what host), and what agents they contain. This functionality is also provided by a single button, which upon clicked it opens a dialog to prompt the user to select a configuration among the available ones (see Figure 24).
- **Starting the platform.** Having imported a configuration, the multi-agent platform is now ready to be deployed. Another button is provided for this action ("*Start the platform with active configuration*" button).

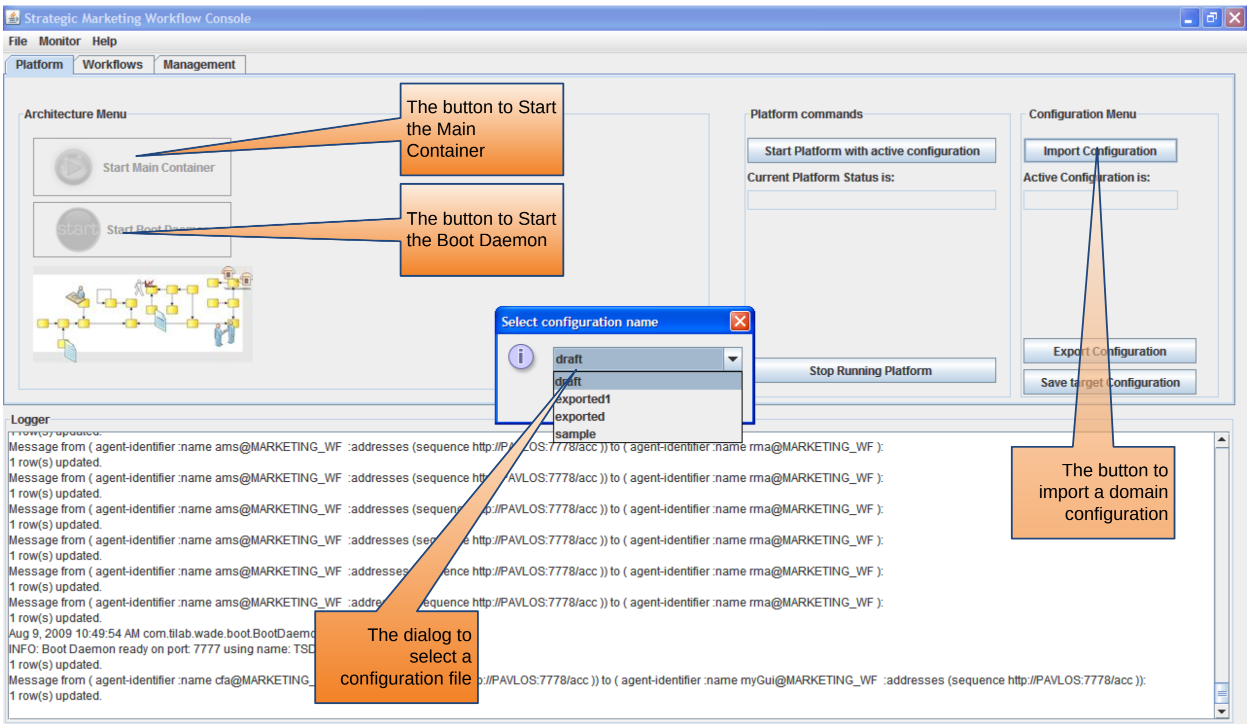


Figure 24 Starting the multi agent platform and providing domain information.

Besides the necessary actions, some extra facilities are provided such as exporting current configuration (the current configuration may differ from the one imported, as new agents may have been added or some agents may have been killed during the application's runtime), saving the configuration that exists in the "target" slot, stopping the platform, a label to show the name of the active configuration and another label to show the current platform status.

Having started both the *Boot Daemon* and the multi-agent platform with a specific configuration, the *Platform* windowpane shall look like the one depicted in Figure 25. The active configuration name will be visible in the text box under the "*Import Configuration*" button, and the platform's status will be visible in the textbox underneath the "*Start Platform*" button, while the *Logger* will contain all the logs that will have been printed during the platform initialization. Notice that the button to start the *Main Container* and the *Boot Daemon* are both disabled.

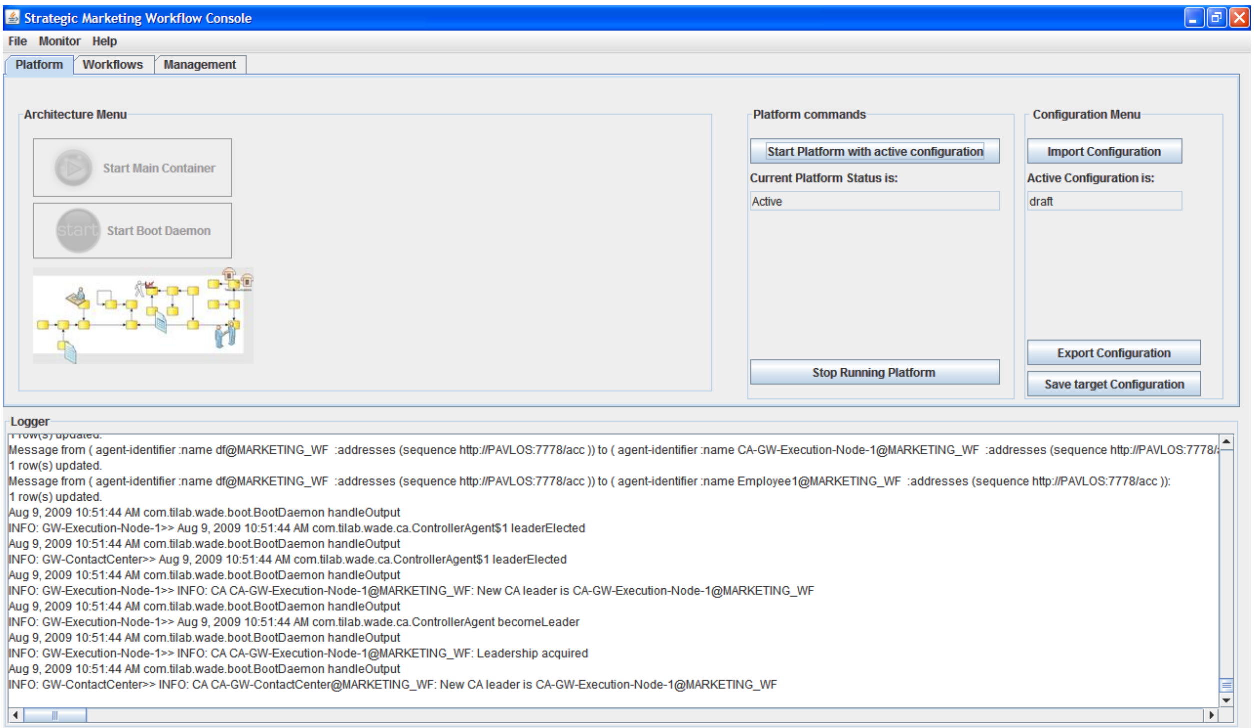


Figure 25 The Platform pane after the initialization of the platform with a specific configuration.

5.1.3 Workflow related actions

The workflows windowpane contains three vertical sub-panels. The leftmost one concerns process data actions and controls, the middle one the performers' control and actions and the rightmost one contains the necessary control to handle the actual workflow class execution.

As illustrated in Figure 26, the "*Process Data*" panel allows the user to select a process type among the available one (the combo box at the top of the panel), and to choose if he/she will begin a new process instance or he/she rather resume an existing one that has been suspended. According to that choice (starting a new instance or continuing an existing one) the respective controls group is activated. In case of a new instance, just the name of instance and a button to submit it are needed. In case of resuming an existing instance, more information is needed in order to implement the features of manual intervention and statefulness described in sections 4.5 and 4.6 respectively.

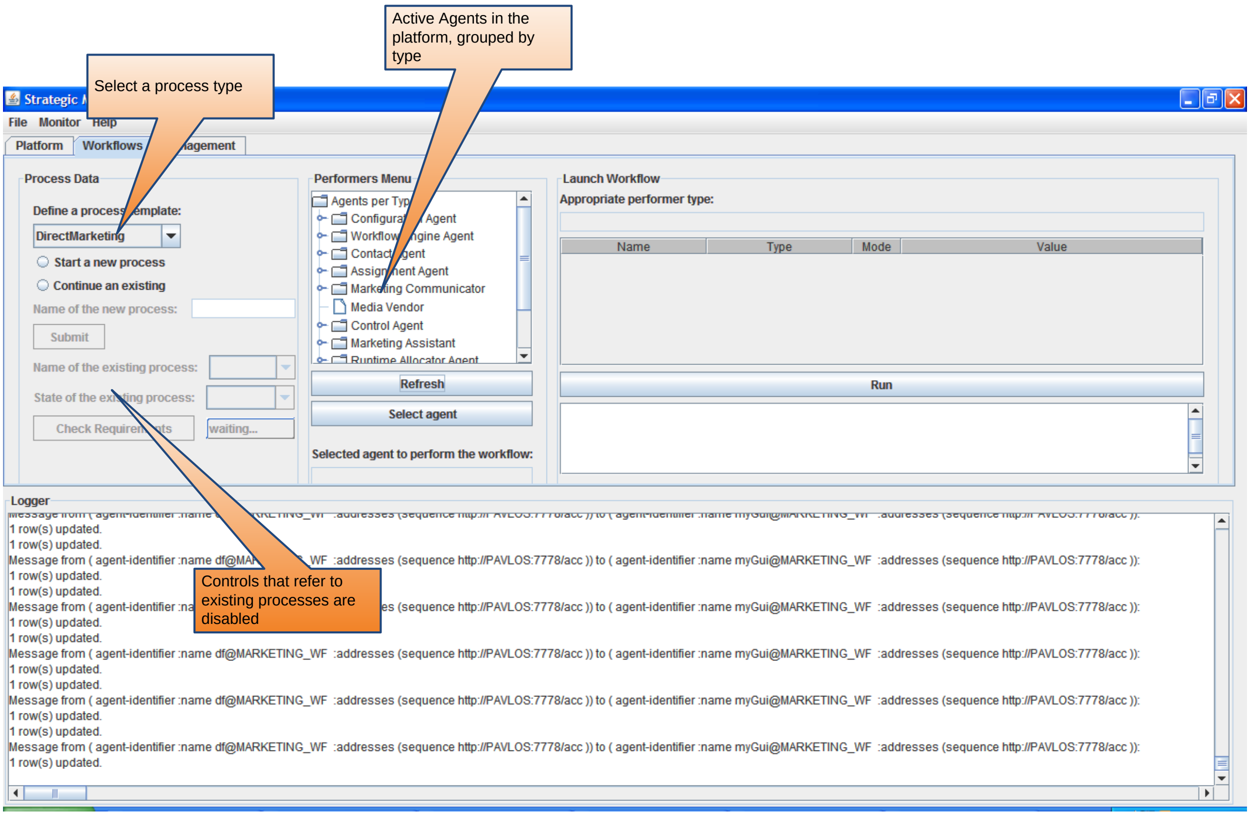


Figure 26 The workflow pane

In particular, based on the selected instance's process type, the available "*states*", that a process instance could be in, are identified. For example, in Figure 27, the selected process instance's type is "*Direct Mail Campaign*". As a result, twelve (12) possible states are identified and are published to the respective combo box at the bottom of the "*Process Data*" panel. This information is retrieved from the application's database. Let's suppose that the user selects to resume execution from the "*SEGMENTATION*" state.

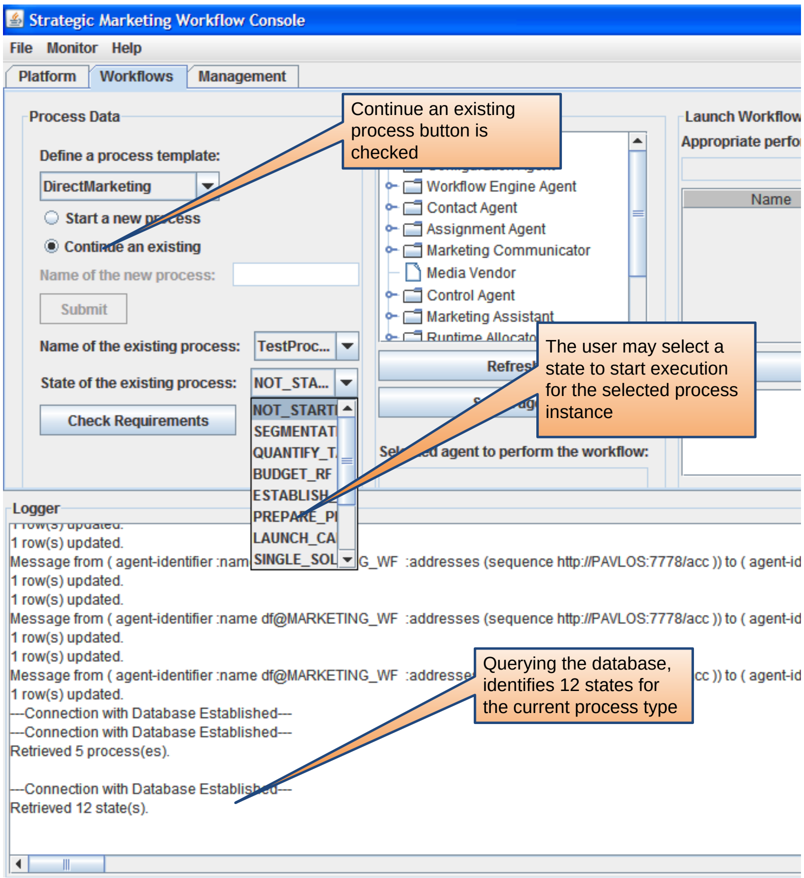


Figure 27 Choosing to continue an existing instance

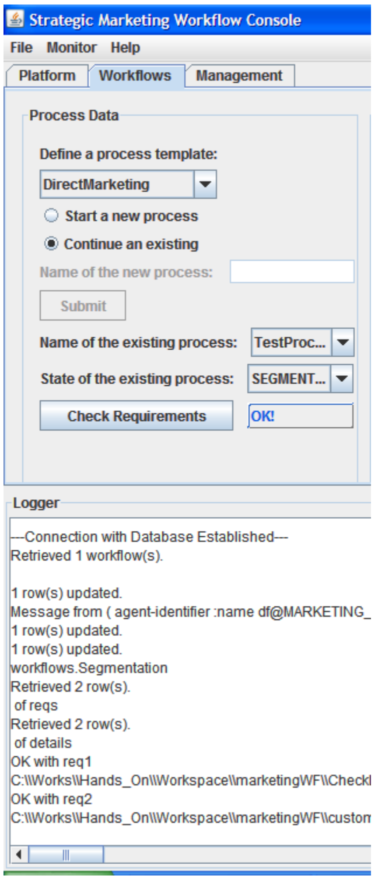


Figure 28 Checking instance's requirements

Then, by pressing the button "*Check Requirements*" (see Figure 28), the application checks if the requirements that are related with the specific state exist for the particular process instance. If yes, the file paths are printed to the *Logger*, and an "*OK*" label becomes visible. If at least one requirement does not exist, then a message declaring the problem is printed.

In addition, when a state is selected, the rightmost panel, the "*Launch Workflow*" panel is activated. Since every state is related with a workflow class (see section 4.6), the application can easily understand which workflow class needs to be executed. Thus, it

provides an indication of what is the appropriate agent type that it could execute the respective workflow (see Figure 29, the red oval shape at the top). Then, the user can focus on the performers' panel (in the middle of the windowpane) and in specific on the tree list. By expanding an agent's type, users can see the available agents of that type in the platform. By selecting one and pressing the "*Select agent*" button, the name of the agent that has been selected to perform the workflow becomes visible (see Figure 29). Next, the user shall fill any parameters that a workflow class may need.

For example, in Figure 29, the workflow class needs only an integer value to be specified. Having selected the performer agent and having provided the necessary parameters, the user can start the workflow execution by pressing the "*Run*" button. Then, it is up to the business logic to call external applications, opens supplementary graphical interfaces etc. in order to properly execute the workflow. During workflow execution, logs are printed to the *Logger*.

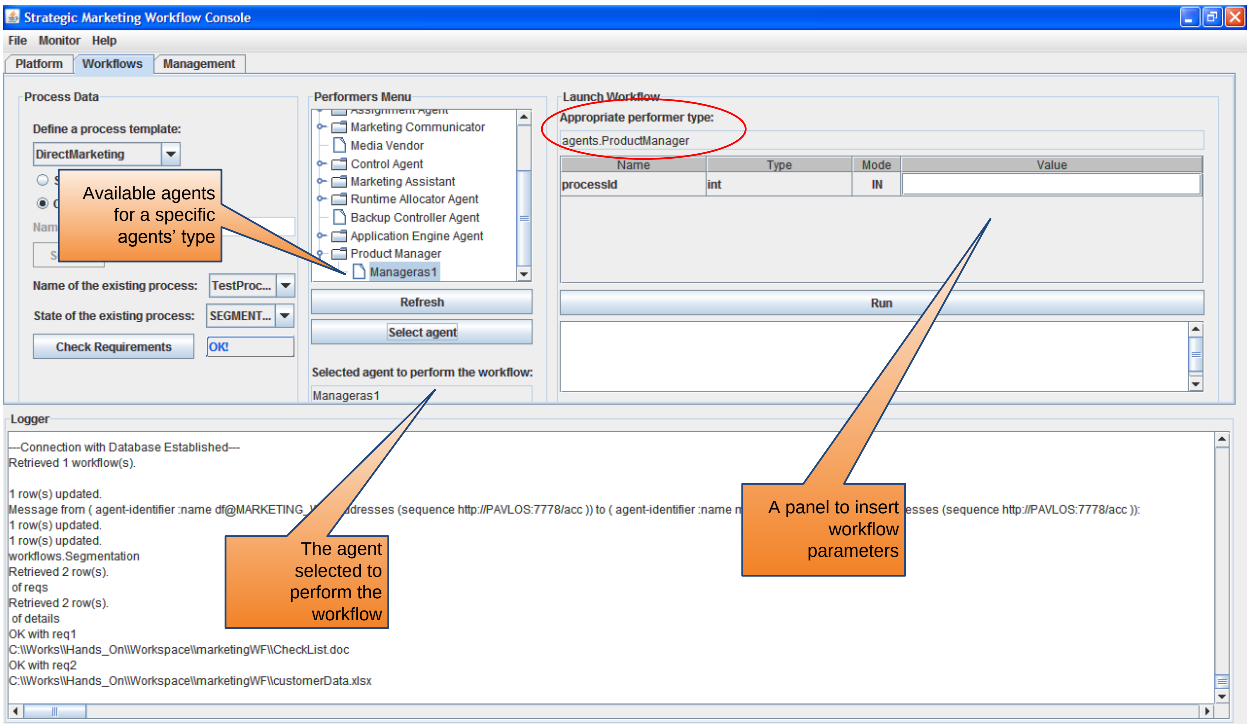


Figure 29 Providing workflow parameters

5.1.4 Application configuration and management related actions

The third windowpane allows users to tune and manage some application's configuration parameters. In particular, the "*Management*" windowpane contains a text editor where users can edit the three most important configuration files of the application. Each file opens when its dedicated button is pressed. The "*Open JADE configuration*" button opens the `main.properties` file, which contains the multi-agent's platform parameters (e.g., the name, the port, the services initiated etc.), the "*Open WADE types file*" button opens the `types.xml` file which contains information about the role and the types of agents, and finally the "*Open a configuration file*" button opens a file dialog to prompt the user to select a platform's configuration file (see section 5.1.2). Users can save the edited files by pressing the "*Save File*" button.

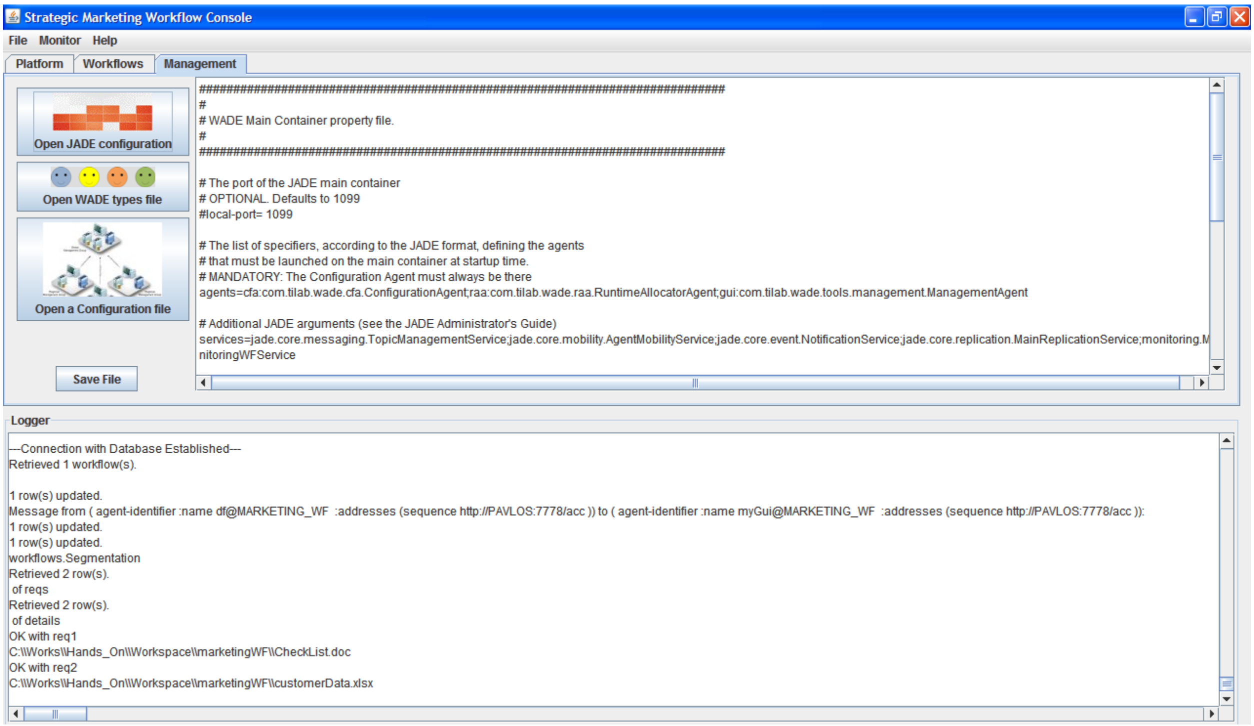


Figure 30 Editing the configurations' files.

5.1.5 Other actions

In addition to the platform, workflows and management actions described in the previous sections, the prototype provides some supplementary facilities. These facilities can be found in the application's menu and they are the following:

- Saving the log data to a file
- Retrieving the platform's current status and printing it to the Logger
- Open the application's documentation file
- Open a message dialog which contains general information.

One may have notice that the application does not contain any controls to handle agent-related actions (start a new agent, kill an existing one, create a container etc.). As such features are fully provided by JADE, the prototype encapsulates them by opening the JADE graphical interface in a separate window. This is achieved by declaring the `-gui` option in the `main.properties` file.

5.2 Evaluate the Prototype against the Classification Criteria

5.2.1 Process Definition Tools

5.2.1.1 Analyze, model, compose, describe, and document a Business Process

The ways that the application uses to model business processes are described in section 4.4. Two are the possible modes: either built a workflow class in JAVA language, or create a domain-specific ontology and exploit agents' communication to impose workflow logic through interaction protocols.

5.2.1.2 Process Definition Write / Edit

The process definition (defined as a workflow class) results in an agent's behavior. Agents are allowed to add a workflow to their behaviors' pool, but in general they are not able to edit the process definition.

5.2.1.3 Definition retrieval

The application uses a special mechanism to fetch definitions to agents. This mechanism accept as an input variable a state of a process type and returns the definition itself, the requirements needed to allow its execution, the process parameters, and the appropriate performer type (see sections 4.4.1.2 and 5.1.3)

5.2.2 Workflow Client Applications

5.2.2.1 Worklist Handling

The notion of worklist in not strictly defined inside the application. There may be ad-hoc worklists, related to the special activities that the processes describe, but their handling is also special and can not be criticized as an application feature.

5.2.2.2 Process control

Agents supervise workflow execution and they are authorized to start, suspend, resume or even terminate an instance's execution. Since a workflow is an agent's behavior, an instance can not be executed without the agent's support.

5.2.2.3 Data Handling

Agents handle application data as workflow class parameters, workflow relevant data as their behaviors fields and workflow control data by notifying the Controller Agents and other system components (e.g., Application Engine, Boot Daemons) about the status of the workflow execution.

5.2.2.4 User Interface

The main application's interface is associated with a GUI agent which takes over the user – platform communication. It is actually a bridge between the reactive graphical interface and the proactive nature of agents. Application specific agents may also have their own custom interface to communicate with business actors.

5.2.3 Invoked Applications

To realize the reference marketing business processes, numerous external applications are invoked, like mail clients (MS Outlook), Office applications (MS Excel, MS Word), technical computing software (MATLAB), databases (MySQL), operating system's runtime, and Web Services.

5.2.3.1 Worklist Handling

Similarly with 5.2.2.1, no formal worklist handling is defined.

5.2.3.2 Process Control

Invoked applications, according to the WADE formalism and the XPDL meta-model, are invoked through a workflow class as atomic activities (tool activities). Thus, the workflow performers (agents) are responsible for the synchronization of the invoked application and the rest workflow activities.

5.2.3.3 Data Handling

Similarly with 5.2.2.3, agents are responsible for all three types of data (Application, Control and Workflow Relevant data).

5.2.3.4 Service Discovery

Discovery services are provided by the platform's directory facilitator (DF). The DF maintains the services descriptions for all available agents in the platform. In addition, for the agents that are workflow-enabled, properties like their type or their role are also maintained and can be used as search filters.

5.2.4 Workflow Interoperability

5.2.4.1 Common Interpretation of Process Definition

The "common interpretation" concept is applied by means of a single process definition that guides multiple agents and that imposes an ordering on agents' behaviors. In addition, the ontology approach is actually a collective preservation of a process model, which ubiquitously exists in the agents behaviors.

5.2.4.2 Workflow Data Interchange

Agents interchange workflow data based on message-oriented formal interaction protocols. As the system is built on top of JADE [139], which is a FIPA compliant platform, agents may or may not reside to the same host or platform.

5.2.5 Administration and Monitoring Tools

5.2.5.1 User / Role Management

The system exploits the natural abstraction of agents as autonomous actors to map them against business roles. The `types.xml` configuration file is used to declare during build time the types and the roles to which each workflow-enabled agent class corresponds. More in details an agent type has a name, a corresponding class and possibly a set of properties that will apply to all agents of that type. Type management is provided by WADE through the `TypeManager` class.

5.2.5.2 Audit Management

Audit management takes place in the system by semantically decoding the messages that agents exchange during workflow execution, and by registering them into the system's database (correlating them with the application runtime and the process instance that they refer). Moreover, the audit trail, printed to the application's screen during runtime can be saved and evaluated at a later time. Finally, the fault tolerance mechanism provided by WADE is always present to handle any exceptions caught.

5.2.5.3 Resource Control

Although resource conflicts are avoided by allowing multithreaded workflow execution, no additional formal resource control mechanism is designed.

5.2.5.4 Process Monitoring

Although the system does not record additional log data (except the ones referred in 5.2.5.2), it does supervise processes through Boot Daemons and Controller Agents, and it does query platform status through the actions specified in the Configuration Ontology, provided by WADE.

5.2.6 Workflow Enactment Service

5.2.6.1 Runtime Control Environment

Agents' communication and coordination are achieved through messaging, interaction protocols and proper workflow ontology. Although there is not central workflow engine, agents encapsulate workflow logic by executing workflows as their behaviours.

5.2.6.2 Definition Interpretation

Agents are able to interpret the workflow definitions as the workflows are ultimately agent behaviours.

5.2.6.3 Execution of Tasks

Agents control the atomic tasks that are parts of a WF instance, and they execute tasks themselves or they delegate them to other agents. They may wrap other tools that finally realize tasks' workload.

5.2.6.4 Scheduling

Workflows are added to agents' behaviours pool, each running in a different thread, and they are executed preemptively, according to the default JAVA threads scheduling model.

5.2.6.5 Data Functions

Agents are responsible for a plethora of data transactions, including querying the database, reading files, getting workflow results, saving them to files etc. The system uses a mixed style to handle data functions: Each agent has its own specified data handling methods, but there also common access mechanisms that provide data access utilities, like the `marketing.wf.gui.DBGUIUtils` class (see Appendix).

5.2.6.6 Task Assignment

Agents often decide "who is going to do what" according to the general guidelines of the business logic, specified in the workflow definition. In particular, they are able to execute a task by themselves, or they can delegate it to another agent. Delegations are usually decided based on agents' types or roles. However, although task assignment takes place during runtime, task assignment decisions follow strictly the process definition guidelines, and agents are not able to dynamically modify them.

5.2.6.7 Resource Allocation

Concerning domain-specific resources, allocation algorithms can be developed and applied (see Section 5.3). Nevertheless, concerning resource application in platform-level, the resource allocation facilities are provided be WADE and refer to policies that allocate agents to containers.

5.3 Exploiting the Prototype to Deploy Algorithms. The Case of a Scheduling Algorithm.

In this section the prototype is utilized as a test-bed to design and apply effective algorithms. The domain background is supplied by the "Customer Contact Center Management" business process, described in section 4.1.2. In particular, the focus is in designing an algorithm that will allow the supervisor of the contact center (the AssignmentAgent – see Appendix) to dispatch the tasks to available contact agents, in such a way that the derived schedule will optimally exploit the available resources (agents). A preliminary version of this algorithm is presented in [130], while in [131] a more analytical version, yet under a different modeling perspective, is described.

5.3.1 The algorithm's context and similar works

The notion of resource is a fundamental concept in Workflow Management. It is a resource (a human or a machine) that supports each execution of a workflow activity [1], and imposes its execution constraints and limitations. Likewise, finding the most appropriate resources is probably the most significant function of a workflow management system (WfMS) [66]. Proper resource management should match each atomic task with an allocation principle, and ultimately with the most suitable resource. An allocation principle should support two decisions; the first refers to the execution order of the tasks, while the second to the assignment of the tasks to the most appropriate resources among the available ones [52]. The need for an appropriate execution order of the tasks causes the resource allocation decision problem to become tightly related with workflow scheduling. Although a large research effort has been made to workflow scheduling [154-157], the methods proposed pay most attention to the validation of the temporal constraints of the workflows while they hardly tackle the resource constraints. On the other hand, when the focus is on the resources, most attention is paid to modeling issues [158-161] while workflow scheduling is barely addressed. The innovative approach proposed in this section simultaneously tackles both the resource allocation and the workflow scheduling problem.

The combined problem, mentioned above, can be addressed either in build-time or in run-time (in [162] this classification is mentioned as business process modeling issues and implementation issues). Addressing the problem during build time allows a more intent validation of the process model and a fair identification of the conflicts. Build-time approaches are most appropriate for optimizing the workflows over their control constraints. Nevertheless, they only use static information to schedule the tasks and to allocate the resources. However, in workflow management systems, there are some real-time issues (such as resource utilization, resources unavailability due to failures, actual throughput etc) that should be considered. Not surprisingly, real-time issues can only be tackled by run-time methodologies. In general, build-time methods optimize process models to eliminate resource conflicts while run-time methods optimize workflow scheduling and resource allocation respective to conflicts constraints.

As far as the build-time methods are concerned, a popular approach is to use a sound process modeling approach -such as Petri-Nets- to model the workflows [154, 163], and incorporate the allocation principles into the static process models. Researchers following this approach, rely on the soundness of the process model to guide the enactment of the process instances while they follow some common queuing disciplines (First-In First-Out; Last-In First-Out; Shortest Processing Time; Earliest Due Date [164]) for selecting the execution order of the tasks. A more sophisticated approach is to use mining techniques to address the structural aspects of the workflows [165] and to facilitate the automation of the execution (e.g., ECA rules used in [166]). A variation of Petri-Nets, the so-called Resource-Constrained Workflow Nets, is introduced in [167] to deal with resource conflicts. The authors of [167] present a method to assess the sufficient amount of initial resources that guarantees successful termination of the process. They indeed claim that the amount of resources calculated this way is sufficient, no matter the scheduling policy used. The calculation of the sufficient amount of resources is an important factor during the design process of the information system, since overestimated piles of resources would eliminate resource conflicts but they will also result in a wasteful architecture.

The above approaches, strive to verify the workflow specification during build-time by checking the process model for inconsistencies and by optimizing the model's design. However, optimizing the process design and minimizing resource conflicts, does not routinely yield optimal resource management. There is a supplementary need to balance resources utilization in order to maximize the benefit per resource ratio. Besides being a matter of cost, a balanced workload may also result in better system performance. Considering these additional issues, a stochastic Workflow Net modeling approach is applied in [168] to optimize the process throughput. The optimization function considers the execution time of the atomic tasks and the resource utilization in order to allocate the available resources to bottleneck-prone tasks. Nevertheless, this algorithm needs a special modeling of the processes, so that they can hardly be applied in the case of multiple interoperating workflow management systems, each of them complying with workflow specification language standards.

A different approach is to address the problem during run-time instead of build time. In the WorkWeb system [96] resources are associated with agents. These agents mutually communicate to reserve office resources and to check their availability. In [27], resource allocation agents are employed to manage resource collisions and to optimize resource utilization. Broker agents, which keep a registry of the available resources and communicate with the runtime control environment, are also a common approach for tackling the resource allocation problem in WfMS [22, 84, 99, 121]. However, in agent-involved WfMS the dominant technique to dynamically assign resources is the "negotiation" [10, 16, 18, 20, 26, 125]. The allocation procedure is optimized through market mechanisms, since the negotiating agents accept the most profitable bid. Negotiation is indeed a flexible mechanism, but one should ensure that human resources would be able to keep in line with negotiating machines (e.g., broker agent) and that the bilateral negotiations do not obstruct system scalability.

In essence, effective resource management in WfMS should examine resource allocation together with task scheduling since these problems impose mutual constraints. Optimizing the one factor subject to the other one constraints (e.g., minimizing resource conflicts subject to temporal constraints or optimizing throughput or utilization subject to resources constraints) is an admissible strategy, but ideally, there should be an algorithm that would jointly optimize both. Coupled with an effective algorithm, a WfMS should support an efficient control mechanism to ensure that the system will not fail in case that any conflict occurs. Also, a WfMS should consider that it should be functional and operable in an open and ubiquitous environment.

All the above considered, the target is to propose an effective algorithm within the framework of a WfMS. Previous research in these critical workflow decision problems is advanced with a threefold contribution: Firstly, the resource allocation problem is addressed in tandem with workflow scheduling since the final output is both a process scheduling plan and a resources reservation arrangement. Secondly, the two critical factors of resource management, resource conflicts and resources utilization are jointly optimized. A consistent modeling approach allows the transformation of data of both these factors into a matrix format so that exploitation if the notion of generalized eigenvalues and the Ky-Fan theorem [169] becomes available. Finally, the proposed method can be exploited to assess the minimum amount of resources needed for proper workflow enactment, namely to support the system design phase. However, the method's primary goal is to be applied as a run-time mechanism, through the multi-agent platform that supports the workflow management of the "Customer Contact Center Management" business process. In particular, to support the supervisor agents to manage the allocation decisions for their registered resources.

5.3.2 The resource allocation decision

A *Time Window* TW is considered when N tasks demand for execution. This time window can be considered as a time interval after which a new allocation procedure is activated. In the "Customer Contact Center Management" (CCCM) case, the time window equals the period of checking the incoming e-mails while a task corresponds to serving a single mail. These tasks are denoted as T_i, $i = 1,2 \dots, N$. Variable N denotes the overall number of tasks. A resource may be a machine; a human; or even a composite resource (e.g., a human together with a machine). Nevertheless, in the CCCM case a resource is equivalent with a contact agent. Atomic tasks do not request for specific resources yet the demand to be timely served by anyone who is capable of serving them (i.e., contact agents can serve incoming mails in contrast with other agent types –e.g., the AssignmentAgent – who can't).

A task's start time is denoted as ST_i and signifies the e-mail's arrival time. Six hours later (see 4.1.2) is the task's deadline, called the finish time (FT_i). The necessary time to serve an e-mail, i.e., its execution duration is symbolized with d_i and as described in section 4.1.2, it depends on the e-mail topic. Tasks are assumed to be assigned in a *non-preemptable*, non-interruptible way. A task is said to be *non-preemptable* if once it begins execution by an agent, it has to be completed by that agent. Additionally, a task is said to be *non-interruptible* if once it starts execution it cannot be interrupted by other tasks and resume execution later. Under this assumption, once a task has been assigned to an agent for execution and another task requests for service during the execution time interval, then, the latter task should be assigned either to another agent (which is not reserved at the requested time interval) or undergo violation of its quality requirement, i.e., its deadline.

To prevent this from happening, we define as z_{ij} the non-overlapping measure between tasks T_i and T_j. Since non-overlapping is the desired situation, we define z_{ij} as

$$z_{ij} = \begin{cases} \alpha, & T_i, T_j \text{ do not overlap} \\ 0, & T_i, T_j \text{ overlap} \end{cases} \tag{1}$$

where $\alpha > 0$ any positive non-zero value.

Finally, we need to denote as A_m the set of all tasks executed by the m^{th} agent. Sets A_m, for different agents m, $m=1,2,\ldots,M$, are mutually exclusive, meaning that a task cannot be split and executed collectively by different agents, assuming a non-interruptible scheduling scenario.

5.3.3 Optimization Criteria

Recalling from section 5.3.1, an efficient allocation policy is the one that maximizes i) the percentage of the active agents (optimizes the workload balancing) while ii) simultaneously minimizes the distortion of the tasks' quality requirements. The first condition is of critical importance for the system performance, since, otherwise, resources are wasted (agent idleness) or not properly used (agent overloading). The second condition states that the allocation policy should respect user's quality parameters as much as possible. We evaluate violation of deadlines and non-dedicated execution of tasks as quality metrics. When an agent executes at the same time more than one activity, it will inevitably split his capacity across the activities. This will lead to broken deadlines and potentially to reduced quality of the deliverable.

Based on the above mentioned requirements, we infer two optimization criteria:

- *Workload balancing* as the minimization of the non-overlapping measure among tasks of different agents and
- *Quality of Service* (*QoS*) as the maximization of the same non-overlapping measure among all the tasks dispatched to a specific agent.

Using equation (1), one can express the non-overlapping degree among tasks of different agents as the sum of the non-overlapping degrees of all tasks assigned to the m^{th} agent with the rest ones, normalized over the sum of non-overlapping degrees between tasks in the m^{th} and all tasks, pending in the system. The corresponding equation is:

$$W_m = \frac{\sum_{i \in A_m, j \notin A_m} z_{ij}}{\sum z_{ij}} \tag{2}$$

where V is the set of the pending tasks (mails).

Low values of W_m mean that many other agents in the system are concurrently active with the m^{th} agent. On the contrary, as W_m increases, the number of concurrently active agents with the m^{th} one decreases. In the same way, we can express QoS as:

$$Q_m = \frac{\sum_{i \in A_m, j \in A_m} z_{ij}}{\sum_{i \in A_m, j \in V} z_{ij}} \tag{3}$$

The numerator of (3) expresses the sum of the non-overlapping degrees for all tasks of the m^{th} agent. The denominator of equations (2) and (3) expresses the non-overlapping values of the tasks executed by agent m with all the N tasks including the ones that are executed by the m^{th} agent. The denominator is used in (2) and (3) for normalization purposes. Instead, optimizing only the numerator of (3) would favor the trivial solution of one task per agent. The Q_m expresses a measure of the QoS violation for the tasks' assigned to the m^{th} agent. As Q_m increases, tasks' overlapping, thus QoS violation decreases for the m^{th} agent.

It is quite straightforward to see that $W_m + Q_m = 1$. Thus, taking into account all the M available resources, the above optimization metrics can be generalized by defining a measure W for the overall workload balancing and a measure Q for the overall QoS violation as:

$$W = \sum_{m=1}^{M} W_m \qquad Q = \sum_{m=1}^{M} Q_m \tag{4}$$

The additive formula that is introduced for the generalization of the optimization metrics does not distort at all the optimization algorithm, since W_m and Q_m are themselves additive formulas of positive values. The ultimate goal of the allocation policy will be to maximize Q while simultaneously minimize$W$. Combining equations (2), (3) and (4), we get

$$W + Q = M \tag{5}$$

recalling from section 5.3.2 that M stands for the number of the available agents.

Since M is a constant number, equation (5) means that maximization of Q simultaneously yields a minimization of W and vice versa. Hence, in the specific context, the two aforementioned optimization requirements are in fact identical and they can be satisfied in parallel. Therefore, it is sufficient to optimize only one of the two criteria. In this case, and without loss of generality, the choice is to minimize W, estimating an optimal task assignment to the M agents, that is a scheduling policy which minimizes the following equation:

$$\widehat{A_m}: minW = min \sum_{m=1}^{M} \frac{\sum_{i \in A_m, j \notin A_m} z_{ij}}{\sum_{i \in A_m, j \in V} z_{ij}}, \forall m \quad (6)$$

where $\widehat{A_m}$, is the estimated set of tasks executed by the m^{th} agent.

5.3.4 The scheduling algorithm

The general idea behind the proposed algorithm is to treat the scheduling problem as a clustering one. In particular, if the M agents are assumed to be M clusters (one cluster per agent) then clustering the N tasks to those clusters will be equivalent to assigning these tasks to the agents. Moreover, the ordering of tasks derives from their start times, so the results are a valid scheduling scheme.

Optimization of equation (6) is a NP-complete problem. Even for the sample case of two agents, (M=2), the optimization of (6) is practically impossible to be implemented for large number of tasks. For this reason, an effective methodology is necessary. Spectral clustering [170], appears to be a compelling algorithm for clustering approaches. An overview of the basic steps of a spectral clustering algorithm is depicted in Figure 31. The analytical mathematical formulation is explained in the next paragraph.

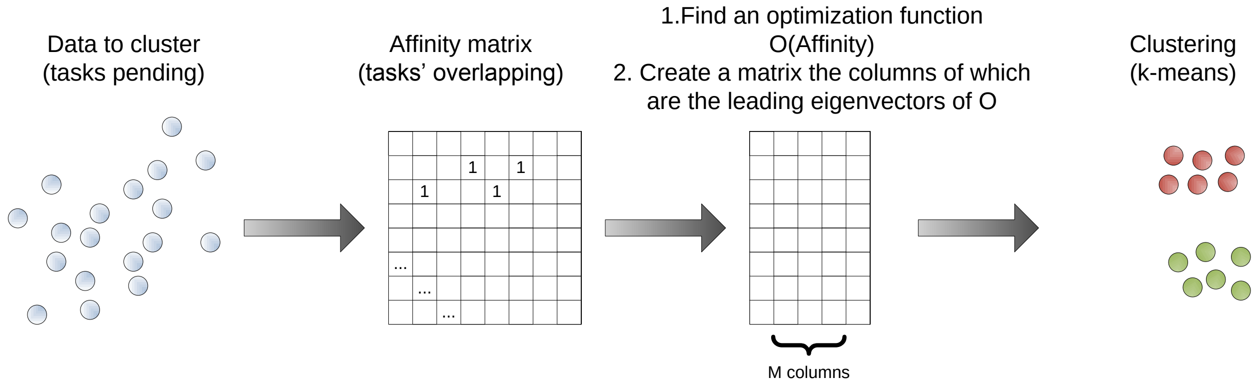


Figure 31 The basic steps of spectral clustering

5.3.4.1 Expressing the optimization metric with a matrix representation

At the beginning, a matrix $\boldsymbol{Z} = [z_{ij}]$ is denoted. Matrix $\boldsymbol{Z}$ contains the values of the non-overlapping measure z_{ij} for all tasks T_i and T_j. Next, an indicator vector $\boldsymbol{e}_m = [\cdots e_m^u \dots]^T$ of size *Nx1* is denoted. The elements e_m^u of this vector are given by

$$e_m^u = \begin{cases} 1, & \text{if task } T_u \text{ is executed by agent } m \\ 0, & \text{Otherwise} \end{cases} \tag{7}$$

The indicator vector $\boldsymbol{e}_m$ points out which tasks are allocated to whom. M different indicator vectors exist, one per agent. Therefore, the optimization problem of (6) corresponds to the estimation of the optimal indicator vectors $\widehat{\boldsymbol{e}_m}, \forall m$, which minimize equation (6). Consequently, equation (6) can be written as

$$\widehat{\boldsymbol{e}_m}, \forall m: minW = min \sum_{m=1}^{M} \frac{\sum_{i \in A_m, j \notin A_m} z_{ij}}{\sum_{i \in A_m, j \in V} z_{ij}} \tag{8}$$

The main difficulty in (8) us that its right part is *not* expressed as a function of the indicator vectors $\boldsymbol{e}_m$. Therefore, there is a need to re-write the right part of equation (8) in a form of vector $\boldsymbol{e}_m$. For this reason, a diagonal matrix $\boldsymbol{L}$ is introduced as $\boldsymbol{L} = diag(\cdots l_i \cdots)$. Elements $l_i, i = 1,2,\dots,N$ express the cumulative non-overlapping degree of the task T_i with all the remaining tasks. That is

$$l_i = \sum_j z_{ij} \tag{9}$$

Using matrices $\boldsymbol{L}$ and $\boldsymbol{Z}$, the numerator of (8) can be expressed as a function of vectors $\boldsymbol{e}_m$. In particular,

$$\boldsymbol{e}_m^T(\boldsymbol{L} - \boldsymbol{Z})\boldsymbol{e}_m = \sum_{i \in A_m, j \notin A_m} z_{ij} \tag{10}$$

In a similar way, the denominator of (8) is related to the indicator vectors $\boldsymbol{e}_m$ as follows

$$\boldsymbol{e}_m^T \boldsymbol{L} \boldsymbol{e}_m = \sum_{i \in A_m, j \in V} z_{ij} \tag{11}$$

Thus, we can re-write (8) as

$$\widehat{\boldsymbol{e}_m}, \forall m: \min W = \min \sum_{m=1}^{M} \frac{\boldsymbol{e}_m^T(\boldsymbol{L} - \boldsymbol{Z})\boldsymbol{e}_m}{\boldsymbol{e}_m^T \boldsymbol{L} \boldsymbol{e}_m} \tag{12}$$

5.3.4.2 Optimization in the Continuous Domain

Assuming *non-interruptible* tasks, we allow agents either to undertake the whole task; or let another agent do the work. That means that the coordinates of vectors $\boldsymbol{e}_m$ take binary values (1 for assignment, 0 otherwise). In other words, we can form the *indicator matrix* $= [\boldsymbol{e}_1 \cdots \boldsymbol{e}_M]$, the columns of which refer to the *M* system agents, while the rows to the *N* tasks. Then, the rows of $\boldsymbol{E}$ have only one value equal to one while all the rest values are zero. Optimization of (12) under the binary representation of the indicator matrix $\boldsymbol{E}$ is still a NP hard problem. However, if we relax the indicator matrix $\boldsymbol{E}$ to take values in continuous domain, then we can solve the problem in polynomial time. We call $\boldsymbol{E}_M$ the *relaxed* version of the *indicator matrix* $\boldsymbol{E}$. The elements of the relaxed matrix take real values.

It can be proven [131] that in the continuous domain the right part of (12) can be written as

$$W = M - trace\left(\boldsymbol{Y}^T \boldsymbol{L}^{-1/2} \boldsymbol{Z} \boldsymbol{L}^{-1/2} \boldsymbol{Y}\right) \tag{13}$$

subject to $\boldsymbol{Y}^T\boldsymbol{Y} = \boldsymbol{I}$ where $\boldsymbol{Y}$ is a matrix which is related to the matrix $\boldsymbol{E}_M$ through the following equation

$$\boldsymbol{L}^{-1/2}\boldsymbol{Y} = \boldsymbol{E}_M \boldsymbol{\Lambda} \tag{14}$$

and $\boldsymbol{\Lambda}$ any $M \times M$ matrix. By selecting $\boldsymbol{\Lambda}$ to be equal το the identity matrix $\boldsymbol{I}$, the *relaxed indicator matrix* $\boldsymbol{E}_M$ (the matrix we are looking for) is calculated as

$$\boldsymbol{E}_M = \boldsymbol{L}^{-1/2}\boldsymbol{Y} \tag{15}$$

Minimization of the problem (13) is obtained through the Ky-Fan theorem [169]. The Ky-Fan theorem states that the maximum value of the $trace\left(\boldsymbol{Y}^T \boldsymbol{L}^{-1/2} \boldsymbol{Z} \boldsymbol{L}^{-1/2} \boldsymbol{Y}\right)$ subject to the constraint $\boldsymbol{Y}^T\boldsymbol{Y} = \boldsymbol{I}$ is equal to the sum of the *M (M<N) largest eigenvalues* of matrix $\boldsymbol{L}^{-1/2}\boldsymbol{Z}\boldsymbol{L}^{-1/2}$. Consequently,

$$\max\left\{trace\left(\boldsymbol{Y}^T \boldsymbol{L}^{-1/2} \boldsymbol{Z} \boldsymbol{L}^{-1/2} \boldsymbol{Y}\right)\right\} = \sum_{i=1}^{M} \lambda_i \tag{16}$$

where $\lambda_i$refers to the i^{th} large eigenvalue of matrix $\boldsymbol{L}^{-1/2}\boldsymbol{Z}\boldsymbol{L}^{-1/2}$. However, maximization of (16) leads to minimization of W in (13). Thus, it is clear that the *minimum* value of W will be

$$\min W = M - \sum_{i=1}^{M} \lambda_i \tag{17}$$

The Ky-Fan theorem also states that this minimum value of (17) is obtained through the matrix

$$\boldsymbol{Y} = \boldsymbol{U} \cdot \boldsymbol{R} \tag{18}$$

where $\boldsymbol{U}$ is a $N \times M$ matrix the columns of which are the *eigenvectors* of the M largest eigenvalues of matrix $\boldsymbol{L}^{-1/2}\boldsymbol{Z}\boldsymbol{L}^{-1/2}$ and $\boldsymbol{R}$ an arbitrary *rotation matrix*. Again, a simple approach to select matrix $\boldsymbol{R}$ is to select the identity matrix (i.e., $\boldsymbol{R}$=$\boldsymbol{I}$) so that $\boldsymbol{Y} = \boldsymbol{U}$. Finally, the optimal relaxed indicator matrix $\widehat{\boldsymbol{E}_M}$ is calculated in the continuous domain as

$$\widehat{\boldsymbol{E}_M} = \boldsymbol{L}^{-1/2}\boldsymbol{U} \tag{19}$$

5.3.4.3 Discrete approximation of the results

The optimal matrix $\widehat{\boldsymbol{E}_M}$ of (19) has not the form of the indicator matrix $\boldsymbol{E}$ since its values are continuous, while the elements of $\boldsymbol{E}$ are binary. Recalling that since a non-interruptible, non-preemptable scheduling policy has been assumed, binary values are the desired format. Consequently, in order to accept the optimal solution of (19) as a solution, the continuous values of $\widehat{\boldsymbol{E}_M}$ should be rounded in a discrete form that approximates matrix $\boldsymbol{E}$.

One simple solution, regarding the rounding process, is to set the maximum value of each row of matrix $\widehat{\boldsymbol{E}_M}$ to be equal to 1 and let the remaining values to be zeros. However, such an approach yields unsatisfactory performance in case that there is not any dominant maximum value at every row of $\widehat{\boldsymbol{E}_M}$. Furthermore, it handles the rounding process as N independent problems, implying that each task is delegated without regarding the allocation of the others. An alternative approach, which is actually adopted here, is to treat the N rows of matrix $\widehat{\boldsymbol{E}_M}$ as *M-dimensional* feature vectors. Each one of these feature vectors indicates the association degree of each task and the respective m^{th} system's agent.

More specifically, having normalized the rows of $\widehat{\boldsymbol{E}_M}$, the *k-means* clustering algorithm is applied, considering the rows of $\widehat{\boldsymbol{E}_M}$ as the population to be clustered in M classes. The *k-means* algorithm comprises three phases, the initialization; the clustering construction; and the updating phase.

- **Initialization**: In this phase, the algorithm arbitrarily selects a set of $\widehat{\boldsymbol{E}_M}$'s rows as centers of the classes that are to be constructed. The number of selected rows equals *M*. That means that each class will contain the tasks assigned to one agent.
- **Clustering Construction**: In this phase, the remaining rows of $\widehat{\boldsymbol{E}_M}$ are clustered to the *M* classes using a metric distance. In particular, a row (namely a task) is assigned to a class by comparing its vector with the class centers and selecting as the appropriate class, the one with the most proximate center.
- **Updating**: After the classification, new centers are created as the means of all vectors belonging to a class. In case that these centers are different from the previous ones, a new process takes place and the algorithm moves on to the clustering construction phase for further processing. On the contrary, if the new centers are exactly the same with the previous ones, meaning that the same task assignment have been concluded, no further processing is required and the clustering is terminated.

The performance of the *k-means* algorithm highly depends on the initial selection of the class centers. Thus, the effectiveness of the scheduling policy is actually influenced by the selection of the initial matrix rows. In the proposed algorithm, to overcome such a drawback and simultaneously to search for new possible solutions that will yield, in relatively small time, a satisfactory approximation of the optimal solution in the discrete domain, the experiment is repeated by selecting each time different rows for the initialization, which in turn, will provide different solutions. Among all selections, the minimum is returned as the finest approximation. To put things into perspective, the execution of the algorithm assuming a set of 2000 tasks, when 50 iterations of the *k-means* are used, takes around 40 seconds on a 2.00 GHz duo core processor.

5.3.5 Evaluating the algorithm's performance

To evaluate the performance of the proposed algorithm, objective criteria should be introduced. The evaluation criteria should be able to (a) compare the proposed strategy with other techniques and (b) measure the algorithm effectiveness under different load conditions. Cascading this evaluation need, parameters which characterize the system's load condition should be introduced as well.

5.3.5.1 Defining parameters for the system's load condition

An important parameter for characterizing the load of tasks requesting to be executed is *granularity* (the quality of being composed of large or small grains – particles). This is defined as the ratio of the average tasks' duration over the time window TW on which a task allocation mechanism is activated.

$$g = \frac{D}{TW} \tag{20}$$

In the previous equation D is is the average duration of all tasks requesting for service, i.e.,

$$D = \sum_{i=1}^{N} \frac{d_i}{N} \tag{21}$$

Granularity g is a measure of how demanding the pending tasks are in terms of execution service time compared to the time window TW. High values of g, mean that the pending tasks occupy a significant amount of the time and thus, tasks' overlapping is more probable. On the contrary, low values of g indicate that the execution demands of the arrived tasks are small compared to the time window TW and thus, a better allocation plan can be achieved. For instance, in the special case of $g = 0.5$, corresponding to the fact that the average tasks' duration is the half of the time window TW, tasks' overlapping is certain, save the extreme case that all tasks arrive sequentially one after the other, and thus no gaps or overlapping are encountered.

Granularity is independent from the number N of the arrived tasks, which is also a significant parameter that characterizes the load. Multiplying the number of the arrived tasks N by the granularity g, we can derive a measure for system characterization as

$$B = \frac{ND}{TW} = N \cdot g = \lambda \cdot D \tag{22}$$

where λ denotes the tasks' arrival rate defined as the ratio of the number of tasks, say N arrived within a time window TW over this window, i.e., $\lambda = N/TW$.

Parameter B is a *low bound* of number of the resources needed to achieve no task's overlapping. This low bound B is smaller than the minimum number of resources M_{opt} required for no tasks' overlapping even using an exhaustive allocation strategy. It should be mentioned that M_{opt} cannot be reached in real life scenarios, since the exhaustive search algorithm is a *NP-hard* problem. That is,

$$B \leq M_{opt} \tag{23}$$

The low bound B reaches the optimal value M_{opt} in the extreme case that the tasks arrive one right after the other within the time horizon *TW*. For example, if *g = 50%,* (i.e., the arrived tasks occupy for execution half of the window time) and *N = 2*, the low bound of resources equals one, which coincides with the optimal value for the number of resources, only in the extreme case of a sequential arrival of all the tasks. Thus, B is an indicator for the number of resources required, which can be estimated before the arrival of the tasks, i.e., during the design phase of the system without being necessary to know the time constraints of the tasks, their arrival model, and particular realization. It is clear that the performance of any applicable task allocation scheme would yield higher values for the number of resources needed for no tasks' overlapping thanM_{opt}.

5.3.5.2 Efficiency criteria for evaluating the execution case

The "*execution case*" refers to a test methodology that considers a constant number of agents and assigns the pending tasks to the available agents using a task allocation strategy. Thus, this methodology is proper for dynamic allocation schemes during tasks execution. In this case one objective criterion would be the percentage of the number of tasks that undergo overlapping over the total number of tasks *N* involved in the task allocation process. However, such a metric has the drawback that it depends on the granularity values. More specifically, small granularity values result in very small percentage values. To address this difficulty, the objective criterion created is the ratio of the maximum number of overlapped tasks achieved through the application of a task allocation strategy when a fixed number of agents is used, over the maximum number of overlapped tasks that are generated from the specific simulation (specific time constraints of the tasks) during the time window *TW*. That is

$$F(S, M) = \frac{H(S, M)}{H(EX)} \tag{24}$$

where S denotes the applied task allocation strategy, M the number of available agents, $H(S, M)$ the maximum number of overlapped tasks in case of the task allocation strategy S with M agents and $H(EX)$ denotes the maximum number of the overlapped tasks that have been generated from the experiment during the time window TW. It is clear that as the number of M increases, the ratio $F(S, M)$ decreases regardless of the strategy used, since more agents are available to satisfy the tasks time constraints. In the special case that $M = 1, H(S, M) \equiv H(EX)$ since all the tasks are assigned to the only one available agent. Thus, $F(S, M) \leq 1$ for all M. Nevertheless, $F(S, M)$ expresses the amount of violation of tasks' constraints regardless of the degree of such violation. That is, an instant overlapping between two tasks is handled with the same way as a complete overlapping. To obtain a measure that also accounts for the extent of overlapping, an alternative criterion is defined as

$$J(S, M) = \frac{H(S, M)}{D \cdot H(EX)} \tag{25}$$

The $J(S, M)$ metric returns the sum of overlapping degrees among all considered tasks within the time window TW using for allocation the strategy S and in case that M agents are available. Moreover, the denominator of (25) multiplies the number $H(EX)$ by the average task duration transforming the metric units from number of items to time duration. As a result, $J(S, M)$ expresses the percentage of violation of the tasks constraints.

5.3.5.3 Efficiency criteria for evaluating the design case

The "*design case*" is suitable for the system design phase. This way, the goal is to find the minimum number of agents needed to achieve no tasks' overlapping. Thus, being aware of the traffic statistics of the tasks arrived in the system, the platform can be designed so as to guarantee satisfaction of the tasks' time constraints, with a simultaneous maximization of the workload balancing of the available agents. In the design case, the goal is to find the minimum number of agents required to eliminate tasks' overlapping when tasks of known statistics arrive. This is expressed as

$$M_{min} : F(S, M_{min}) = 0 \tag{26}$$

It should be mentioned that$M_{min} > H(EX)$. This is due to the fact that $H(EX)$ actually indicates that if the available agents equal the maximum number of overlaps, the simultaneous occurrence of the tasks can be avoided. Thus, $H(EX)$ is an ideal number which provides no explanation on how these tasks will be assigned to agents. Instead, M_{min} is the actual minimum number of agents derived by the application of the given task allocation strategy after it assigns all the pending tasks to agents. Thus, the quality of the algorithm can be measured by introducing a *resolvability factor*, defined as the ratio

$$\chi(s) = \frac{M_{min}}{B} \tag{27}$$

which actually indicates how many times the minimum number M_{min} obtained by a task allocation strategy exceeds the low bound B i.e., the number of agents that does not yield any tasks' overlapping in the ideal case that all tasks arrive sequentially one after the other. Hence, the algorithm's scheduling efficiency is defined as the inverse of the resolvability factor$\chi(S)$.

A drawback of the previous measure $\chi(S)$ is that it often under-estimates scheduling efficiency since low bound B is times smaller than the numberM_{min} . Ideally, the algorithm's performance should be compared with the optimal case (i.e., the valueM_{opt}) instead of the underestimated number B. Due, however, to the NP-completeness of the scheduling problem, the optimal number of resources M_{opt} cannot be found and thus such a comparison is impossible. An alternative solution would be to use the number $H(EX)$ which better approximates the number of agents required for no tasks' overlapping. Thus, the measure adopted for evaluating the performance of the proposed algorithm during the design phase is the following ratio, called *waste factor*

$$\zeta(s) = \frac{M_{min}}{H(EX)} \tag{28}$$

Now, it is proper to re-define scheduling efficiency as the inverse of the waste factor $\zeta(S)$. Although $\zeta(S)$ is a better bound for measuring algorithm efficiency than $\chi(S)$, it requires much more effort to calculate $\zeta(S)$ since it is known only if an exact realization of tasks arrival is given.

5.3.6 Experimental Results

Two fundamental input data are needed to generate sample data for testing the algorithm. These are the tasks start times and their durations. In the experiments conducted, both were randomly generated following a uniform distribution. Recalling from section 4.1.2 that a task's duration depends on the mail's type, thus, what is actually randomly generated is the mail type. In all experiments, 100 different realizations were conducted, in order to remove possible noise. The results presented here are the average of these realizations. In [131], a different experimental setup is used; nevertheless, the proposed algorithm appears to outperform in all cases.

The proposed task allocation strategy is also compared against the greedy approach and the min cut technique[171]. The greedy algorithm assigns tasks to the available agents sequentially one after that other (a quasi First-In First-out approach). This assignment takes into account the current load of the resources so that no tasks' overlapping is encountered. When a new task overlaps in time with some already assigned tasks, an extra agent is assumed to be required. In this greedy manner, zero overlapping is achieved. The greedy approach is implemented using two different versions. The first, which is the simplest, randomly selects an agent for task allocation provided that no tasks' overlapping occurs within this agent. This method is called *Greedy Algorithm-Approach A.* The second implementation, initially finds all agents that yield no overlapping of this task with the already assigned ones, for a given task. Then, among these agents, it picks the one which after the task assignment will have them minimum task load so that potentially more tasks can be assigned to this later on, and load is somehow balanced. This method is called *Greedy Algorithm-Approach B.* The other approach used for comparison is the min cut tree algorithm, often used for graph clustering. In this approach, a graph is used, the nodes of which correspond to the N tasks, whereas the edges show the non-overlapping degree between two tasks. The graph is then divided into two clusters by the application of a minimum cut technique. The minimum cut obtained through the use of a maximum flow algorithm [172] corresponds to a two clusters partitioning. Since in the defined problem, the tasks may be assigned to $M \geq 2$ resources, and thus a more clusters partitioning is needed, the two-class approach is iteratively applied, until the number M is reached. Although, both the proposed algorithm and Min Cut rely on graph partitioning, the concept of both approaches is different since the latter does not involve the denominator of equations (2) and (3). Therefore, without the denominator, the optimal solution tends to favor small clusters, a fact which deteriorates the algorithm's efficiency.

5.3.6.1 Testing the algorithm under different load conditions

The tests carried out in this section are suitable for estimating the minimum number of agents required to achieve no tasks' overlapping. In particular, the algorithm's efficiency is plotted, when different load condition are applied. As discussed earlier, factor $\chi(S)$ significantly deviates from the optimal value since it is compared with the low threshold B. Thus, a more appropriate measure is the waste measure$\zeta(S)$.

Figure 32 depicts the waste factor $\zeta(S)$ versus granularity for different values of B *(B = 1, 2, 5 and 10)* in case that the proposed algorithm is used. The results are derived for $B \geq 1$ to test the efficiency of the algorithm in a rather loaded environment.

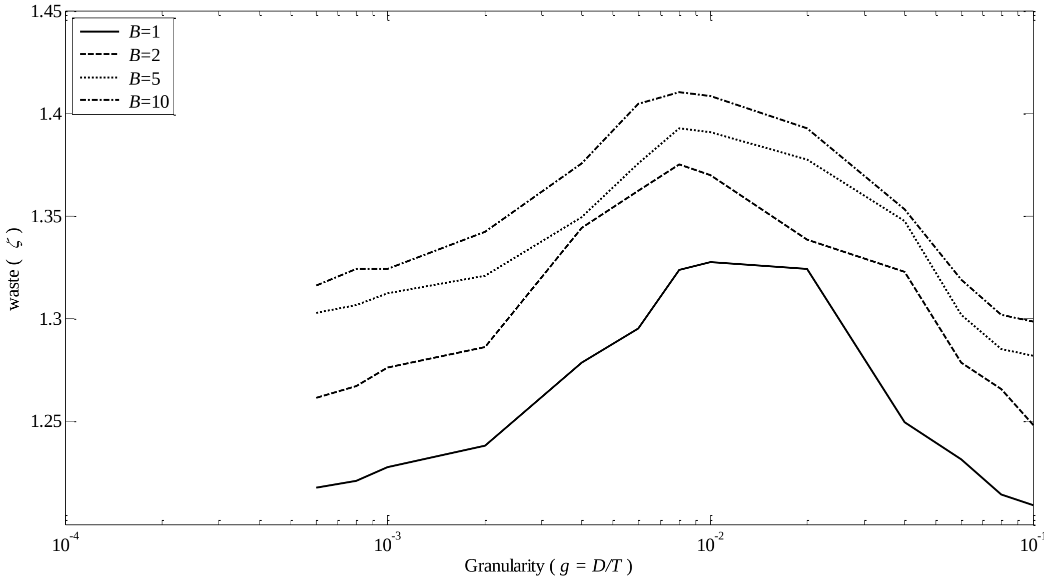


Figure 32 The waste factor versus granularity for different values of the low bound *B*.

For low granularity values, the waste factor $\zeta(S)$ initially increases as granularity increases but with a decreasing rate. This means that the factor remains bounded. It is also observed that for high granularity degrees the waste increases as g also increases since in this case the average duration of arrived tasks is comparable with the time window *TW*.

In all cases $\zeta(S)$ takes very satisfactory values, indicating that on the average the proposed task algorithm allocates the atomic tasks close to the optimal solution. Figure 33 presents the waste $\zeta(S)$ versus the number of tasks for different granularity levels. It is observed that as the number of tasks increases the waste values also increases. However, beyond a certain point, this increase is insignificant. This means the waste converges for a large number of tasks. However the upper limit, even for a large number of arrived tasks is close to the optimal value revealing the efficiency of the proposed task allocation algorithm.

Another parameter (besides load conditions) that affects the algorithm's efficiency is the number of iterations in the *k-means* algorithm for transforming the optimal solution in the continuous domain into a discrete one (section 5.3.4.3). In particular, in Figure 34 the results using 1, 30 and 50 iterations are illustrated. As expected, the performance is improved as the number of iterations increases; however, there is a limit beyond of which the improvement is slight. This means that a relatively small number of iterations (around 50) is practically adequate to get the solution. In our experiments, a maximum value of 50 iterations is used as the termination condition of the k-means algorithm, unless clustering converge is achieved in earlier steps.

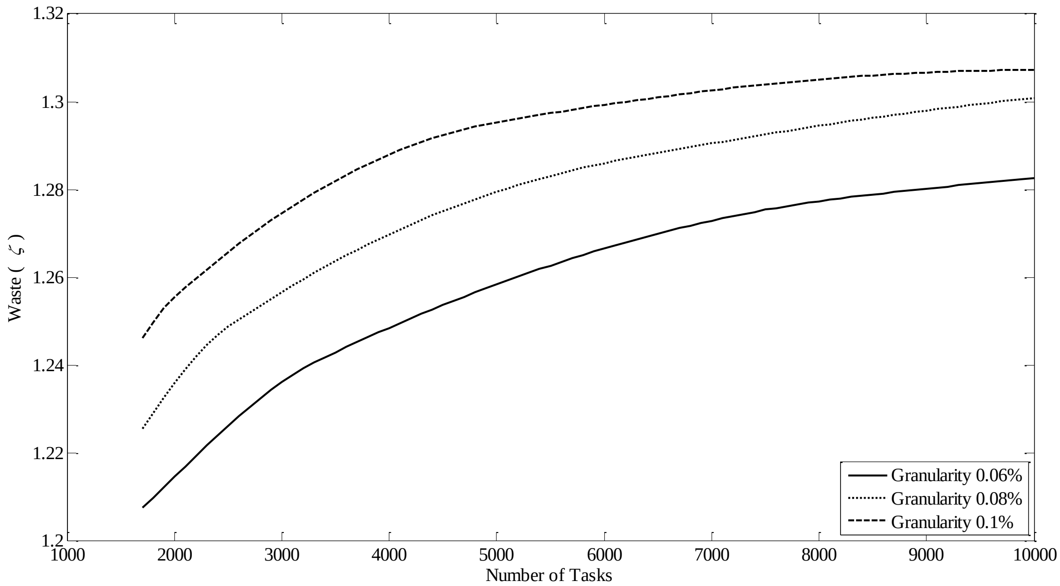


Figure 33 The waste factor versus the number of pending tasks for three different granularity values

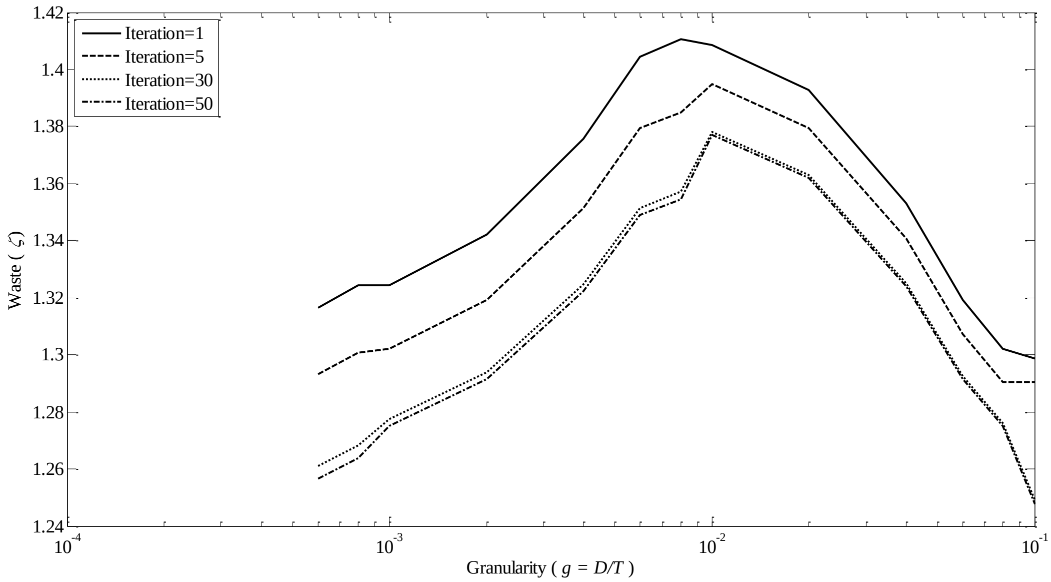


Figure 34 The algorithm's efficiency versus granularity when different number of iteration are used in the k-means descritization phase

5.3.6.2 Comparing the proposed algorithm with other approaches

In this paragraph, the results of the proposed scheduling strategy are compared to the two versions of the greedy approach and the minimum cut tree graph partitioning. The same experiments as in the previous paragraph are repeated, i.e., waste factor versus granularity and waste factor versus the number of pending tasks.

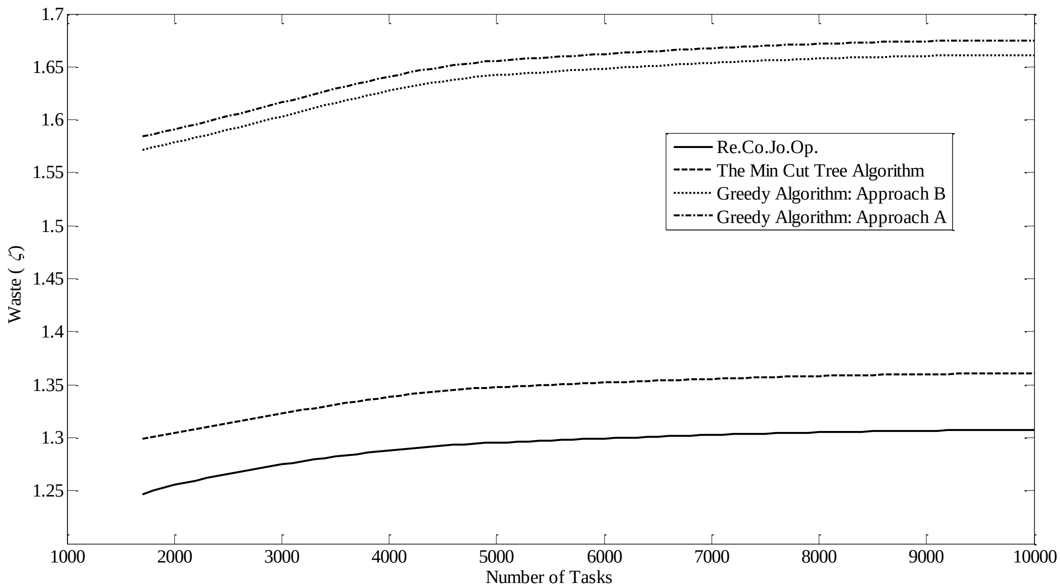


Figure 35 Comparison of different algorithms when the tasks' load augments. The granularity is fixed to 0.1%

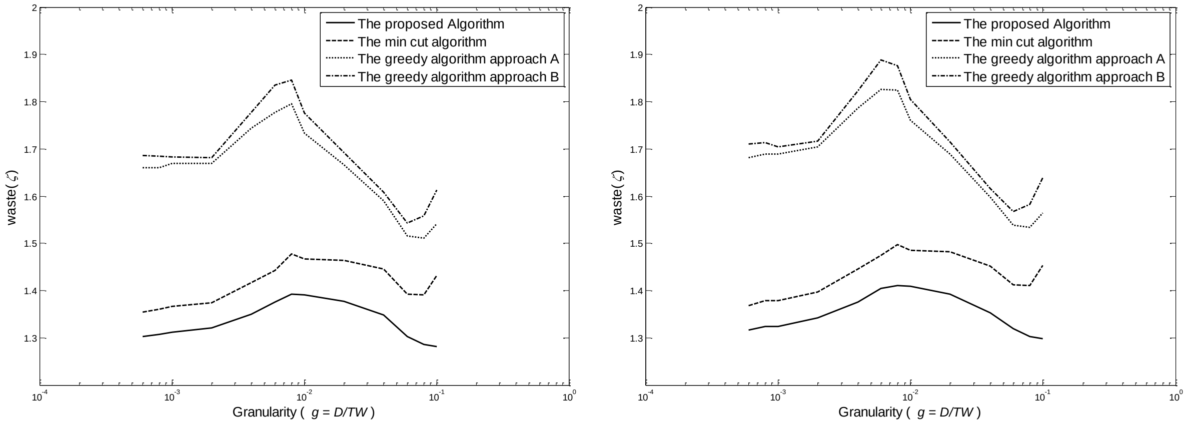


Figure 36 Comparison of the waste factor versus granularity for different algorithms for *B=5* (left) and *B=10* (right)

Figure 35 shows the effect of the waste factor ζ with respect to the number of tasks for a granularity value of g=0.1% for the proposed algorithm and the other three compared approaches. The same exponential performance as in Figure 33 is also derived. The proposed algorithms results in much smaller waste compared to the other methods. Figure 36 compares the performance of the proposed algorithm with the three aforementioned schemes for B=5 and $B = 10$ versus granularity. In both cases, the proposed task allocation algorithm outperforms the other approaches for all granularity values.

A different testing set of experiments is when the algorithm's performance is evaluated when the number of the available agents is constant. The evaluation metrics for such a case are the $F(S, M)$ and the $J(S, M)$ criteria, introduced in equations (24) and (25). The goal now is to estimate the percentage of tasks overlapping using the proposed task allocation scheme for a given number of agents. The same objective criteria are also used to compare the performance of the presented strategy with other algorithms.

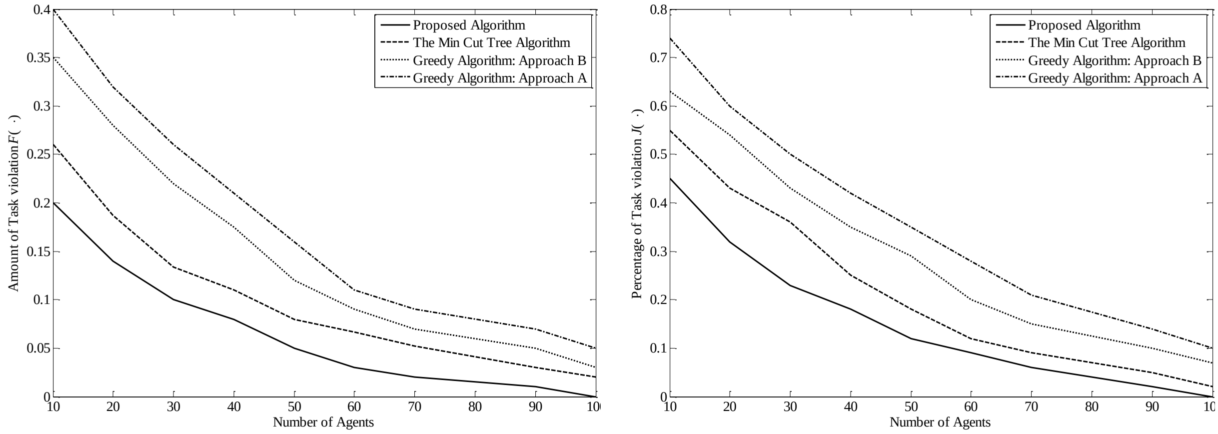
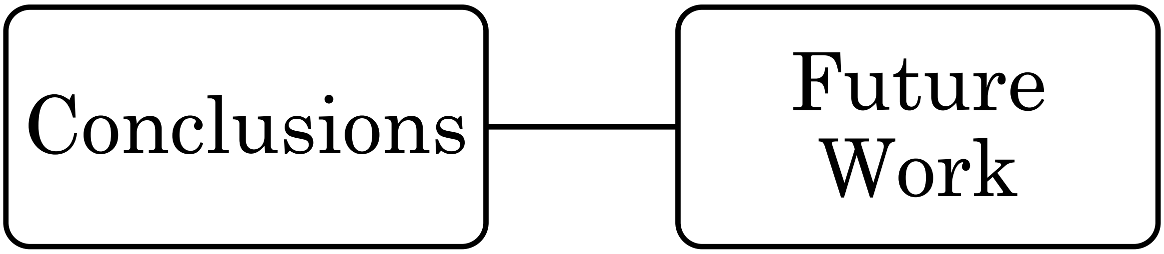


Figure 37 Tasks' overlapping versus the number of agents for different algorithms

Figure 37 (left) plots the $F(S,M)$ versus the number of agents M for *0.1%* granularity. The value of $F(S,M)$ drops as M increases, while when $M = M_{min}$ the $F(S,M)$ becomes zero. It is observed that the proposed algorithm yields smaller deviations from the tasks' time constraints. Similar results are derived for the $J(S,M)$ criterion (Figure 37 (right)).

CHAPTER 6



6 Conclusions

The introductory section of this work claimed a threefold contribution. The first fold, presented in chapter 0, is about a general classification scheme and an extended survey of existing works. As chapter 0 demonstrated, a critical amount of publications aim their attention to the intersection of these WFMS and agents. However, an overarching contextualization of the intersected are was missing before this work's publication. This thesis exploited popular standards of the workflow field to propose a unifying framework, and to clarify the vague picture of Agent-involved Workflow Management Systems.

As the integration of WFMS and agents is thoroughly examined, numerous integration patterns and contribution potentials are described in terms of a WFMS functional decomposition. The proposed classification scheme itself has a double contribution: Not only it provides a guided map of the WFMS functions that can be enhanced by agents, but it consists a reference text for researchers as well. The consolidation of WFMS and software agents is indeed practical and attainable even without a clear picture of the field, yet a unifying framework fairly encourages cross-fertilization.

The second fold, presented in chapter 0 and in section 5.1, concerns a prototype AWfMS. The primal goal of the prototype is to exhibit how some features of workflow management can be enhanced by agenthood, or the inverse, i.e., how multi-agent systems can benefit from the application of workflow logics. Advanced features, such as interaction protocols supporting the workflows, business logic support through a formal process language, agents' behaviours or ontologies, manual intervention, statefulness, and monitoring were designed and implemented, revealing the potential of mixing agents and WFMS.

The third fold, presented in section 5.3, suggests an innovative strategy which simultaneously tackles the problems of scheduling and task allocation. The proposed method jointly optimizes the two critical factors of the defined problem (Workload Balancing and Quality of Service). The proposed algorithm is evaluated under two different environments. The first is appropriate for the execution phase, considers a constant number of available agents and assigns the pending tasks to agents using the proposed allocation strategy. The second evaluation environment is proper for the system design phase. This way, the target is to find the minimum number of agents that will result in zero overlapping, i.e., in no violation of the tasks' time constraints. Thus, based on the traffic statistics of the tasks the system can be designed so that zero violations in tasks temporal constraints are guaranteed, while a non-wasteful number of agents are used. The algorithm's outperformance is evident for all granularity values, and under different assumptions about the system's load conditions.

6.1 Future Work

This text in delivered in tandem with a software piece: the prototype, which was described in the previous chapters and in the appendix. The prototype is a valuable tool to facilitate future research. It allows allow for transparent and replicable testing of new algorithms and computational tools with a reduced effort. Ideally, for each utility described in section 3.2, an optimization algorithm can be developed and tested. In particular, a topic which is already considered is the expansion of the scheduling algorithm proposed (see section 5.3), in order to tackle dynamically the changes in the workflow environment (new agents are added, existing agents are killed or fail to respond, etc.). An additional research theme that is considered for the prototype is about the integration of operative research allocation policies. More specifically, as resource allocation patterns in workflow have been explicitly defined [161], a natural subsequent step is to leverage those patterns in an multi-agent context.

An additional issue, regarding also the prototype is to consider an alternative architecture. As the literature review demonstrated, there is a significant number of cases where a more modular architecture is needed. A modular structural design will allow breaking an enterprise application into multiple modules and thus an easier management of cross-dependencies between them. As this kind of design finds its space and in business environments (e.g., virtual enterprises) and as the Service Oriented Architecture paradigm emerges, a more modular architecture of the prototype AWfMS will make it keep a pace with mainstream technology advancements, thus it will strengthen its practicality.

Considering the workflow concepts, a noteworthy matter with great potential emerges from the results of this thesis: Developing a formal definition of stigmergy for workflow processes. Although section 4.6 presented a way to incorporate stigmergy into the workflow context, a more formal method is required to allow generalization.

Concluding, the above points exhibit the research challenges that the introduction of a unifying framework brings forth. Starting from the work carried out during this thesis, future research is facilitated and stimulated as well. The answer to the key question "*Does it worth mixing agents and WFMS*" may be not unique, yet this thesis provides less complicated way to anticipate the response.

References

[1] WfMC, "WfMC Standards, Terminology & Glossary," Report No: WfMC-TC-1011, *Workflow Management Coalition* 1999.

[2] M. N. Huhns and M. P. Singh, "Workflow agents," *Internet Computing, IEEE,* vol. 2, pp. 94-96, 1998.

[3] P. Kotler and K. L. Keller, *Marketing Management*, Twelfth ed. New Jersey: Pearson Prentice Hall, 2006.

[4] WADE, "Workflow Agent Development Environment," 2008.

[5] M. P. Singh and M. N. Huhns, "Multiagent systems for workflow," *International Journal of Intelligent Systems in Accounting, Finance & Management,* vol. 8, pp. 105-117, 1999.

[6] K. R. Abbott and S. K. Sarin, "Experiences with workflow management: issues for the next generation," in *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, Chapel Hill, North Carolina, United States, 1994, pp. 113-120.

[7] A. P. Sheth, W. Van Der Aalst, and I. B. Arpinar, "Processes driving the networked economy," *IEEE Concurrency,* vol. 7, pp. 18-31, 1999.

[8] M. z. Muehlen, *Workflow-based Process Controlling*. Berlin: Logos verlag, 2004.

[9] E. A. Stohr and J. L. Zhao, "Workflow Automation: Overview and Research Issues," *Information Systems Frontiers,* vol. 3, pp. 281–296, 2001.

[10] H. Li and Z. Lu, "Decentralized workflow modeling and execution in service-oriented computing environment," in *Service-Oriented System Engineering, 2005. SOSE 2005. IEEE International Workshop*, Beijing, CHINA, 2005, pp. 29-34.

[11] S. Staab, W. van der Aalst, V. R. Benjamins, A. Sheth, J. A. Miller, C. Bussler, A. Maedche, D. Fensel, and D. Gannon, "Web services: been there, done that?," *Intelligent Systems,* vol. 18, pp. 72-85, 2003.

[12] M. T. Schmidt, "The evolution of workflow standards," *IEEE Concurrency* vol. 7, pp. 44-52, 1999.

[13] S. Meilin, Y. Guangxin, X. Yong, and W. Shangguang, "Workflow management systems: a survey," in *Communication Technology Proceedings, 1998. ICCT '98*, Beijing, China, 1998, pp. 20-26.

[14] WfMC, "The Workflow Management Coalition," [Online]. Available: http://www.wfmc.org/

[15] N. R. Jennings, T. J. Norman, and P. Faratin, "ADEPT: an agent-based approach to business process management," *ACM SIGMOD Record,* vol. 27, pp. 32-39, 1998.

[16] N. R. Jennings, T. J. Norman, P. Faratin, P. O'Brien, and B. Odgers, "Autonomous Agents For Business Process Management," *Applied Artificial Intelligence,* vol. 14, pp. 145-189, 2000.

[17] G. Fakas and B. Karakostas, "A workflow management system based on intelligent collaborative objects," *Information and Software Technology,* vol. 41, pp. 907-915, 1999.

[18] D. W. Judge, B. R. Odgers, J. W. Shepherdson, and Z. Cui, "Agent-enhanced Workflow," *BT Technology Journal,* vol. 16, pp. 79-85, 1998.

[19] R. Muller, U. Greiner, and E. Rahm, "AGENTWORK: a workflow system supporting rule-based workflow adaptation," *Data & Knowledge Engineering,* vol. 51, pp. 223-256, 2004.

[20] P. D. O'Brien and M. E. Wiegand, "Agent based process management: applying intelligent agents to workflow," *The Knowledge Engineering Review,* vol. 13, pp. 161-174, July 1998 1998.

[21] J. W. Shepherdson, S. G. Thompson, and B. R. Odgers, "Decentralised Workflows and Software Agents," *BT Technology Journal,* vol. 17, pp. 65-71, 1999.

[22] S. Wang, W. Shen, and Q. Hao, "An agent-based Web service workflow model for inter-enterprise collaboration," *Expert Systems with Applications,* vol. 31, pp. 787-799, 2006.

[23] Y. Yan, Z. Maamar, and S. Weiming, "Integration of workflow and agent technology for business process management," in *The Sixth International Conference on Computer Supported Cooperative Work in Design*, London, Ont., Canada, 2001, pp. 420-426.

[24] L. Zeng, A. Ngu, B. Benatallah, and M. O'Dell, "An agent-based approach for supporting cross-enterprise workflows," in *Proceedings of the 12th Australasian database conference* Queensland, Australia 2001, pp. 123-130.

[25] Y. Qu, X. Sheng, and W. Jiao, "A Multi-Agent Based Model of Workflow Management," in *10th International Conference on Computer Supported Cooperative Work in Design, 2006. CSCWD '06.* , Nanjing, China, 2006, pp. 1-5.

[26] L. Hongchen and S. Meilin, "Application of agents in workflow management system," in *Fifth Asia-Pacific Conference On Communications and Fourth Optoelectronics and Communications Conference APCC/OECC '99*, Beijing, China, 1999, pp. 1068-1072.

[27] J. Qiu, C. Wang, and Y. He, "Research on application of intelligent agents in the workflow management system," in *2005 IEEE Networking, Sensing and Control, ICNSC2005*, Tucson, Arizona, USA, 2005, pp. 827-830.

[28] J. W. Chang and C. T. Scott, "Agent-based workflow: TRP Support Environment (TSE)," *Computer Networks and ISDN Systems,* vol. 28, pp. 1501-1511, 1996.

[29] M. Merz and W. Lamersdorf, "Crossing Organizational Boundaries with Mobile Agents in Electronic Service Markets," *Integrated Computer-Aided Engineering,* vol. 6, pp. 91 - 104, 1999.

[30] G. Alonso, D. Agrawal, A. E. Abbadi, and C. Mohan, "Functionality and Limitations of Current Workflow Management Systems," *IEEE Expert,* vol. 12, 1997.

[31] D. Georgakopoulos, M. Hornick, and A. Sheth, "An overview of workflow management: From process modeling to workflow automation infrastructure," *Distributed and Parallel Databases,* vol. 3, pp. 119-153, 1995.

[32] Y.-H. Suh, H. Namgoong, J.-J. Yoo, and D.-I. Lee, "Design of a Mobile Agent-Based Workflow Management System," in *Mobile Agents for Telecommunication Applications: Third International Workshop, MATA 2001, Montreal, Canada, August 14-16, 2001. Proceedings*. vol. 2164, S. Pierre and R. Glitho, Eds.: Springer Berlin / Heidelberg, 2001, pp. 93-102.

[33] D. B. Lange and M. Oshima, "Seven good reasons for mobile agents " *Commun. ACM,* vol. 45, pp. 88-89, 1999.

[34] M. Merz, B. Liberman, and W. Lamersdorf, "Using Mobile Agents To Support interorganizational Workflow Management," *Applied Artificial Intelligence,* vol. 11, pp. 551 - 572, 1997.

[35] G. A. Bolcer and R. N. Taylor, "Advanced Workflow Management Technologies," *SOFTWARE PROCESS—Improvement and Practice,* vol. 4, pp. 125–171, 1998.

[36] A. Sheth and K. J. Kochut, "Workflow applications to research agenda : Scalable and dynamic work coordination and collaboration systems," in *Workflow Management Systems and Interoperability, NATO Advanced Study Institute on Workflow Management Systems and Interoperability*, Istanbul, Turkey, 1997.

[37] M. B. Blake and H. Gomaa, "Object-Oriented Modeling Approaches to Agent-Based Workflow Services," in *Software Engineering for Multi-Agent Systems II*. vol. 2940, C. Lucena, A. Garcia, A. Romanovsky, J. Castro, and P. Alencar, Eds.: Springer Berlin / Heidelberg, 2004, pp. 111-128.
[38] M. B. Blake and H. Gomaa, "Agent-oriented compositional approaches to services-based cross-organizational workflow," *Decision Support Systems,* vol. 40, pp. 31-50, 2005.
[39] R. Kishore, H. Zhang, and R. Ramesh, "Enterprise integration using the agent paradigm: foundations of multi-agent-based integrative business information systems," *Decision Support Systems,* vol. 42, pp. 48-78, 2006.
[40] M. Wang, H. Wang, and D. Xu, "The design of intelligent workflow monitoring with agent technology," *Knowledge-Based Systems,* vol. 18, pp. 257-266, 2005.
[41] J.-J. Yoo, D. Lee, Y.-H. Suh, and D.-I. Lee, "Scalable Workflow System Model Based on Mobile Agents," in *Intelligent Agents: Specification, Modeling, and Application : 4th Pacific Rim International Workshop on Multi-Agents, PRIMA 2001, Taipei, Taiwan, July 28-29, 2001. Proceedings*. vol. 2132, S.-T. Yuan and M. Yokoo, Eds.: Springer Berlin / Heidelberg, 2001, pp. 222-236.
[42] L. Yu and B. F. Schmid, "A conceptual framework for agent-oriented and role-based workflow modeling," in *Proceedings of the CaiSE Workshop on Agent Oriented Information Systems (AOIS99)*, 1999.
[43] P. Buhler, J. M. Vidal, and H. Verhagen, "Adaptive Workflow = Web Services + Agents.," in *ICWS '03*, Las Vegas, Nevada, USA, 2003, pp. 131-137.
[44] M. P. Singh, "Distributed enactment of multiagent workflows: temporal logic for web service composition," in *Second international joint conference on Autonomous agents and multiagent systems* Melbourne, Australia 2003, pp. 907-914.
[45] P. A. Buhler and J. M. Vidal, "Towards Adaptive Workflow Enactment Using Multiagent Systems," *Information Technology and Management,* vol. 6, pp. 61-87, 2005.
[46] G. Joeris, "Decentralized and Flexible Workflow Enactment Based on Task Coordination Agents," in *2nd Int'l. Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2000@CAiSE*00)*, Stockholm, Sweden, 2000, pp. 41-62.
[47] H. Stormer, "Task Scheduling in Agent-based Workflow," in *Int. ICSC Symp. on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce (MAMA'2000)*, Wollogong, Australia, 2000.
[48] H. V. D. Parunak, "Applications of distributed artificial intelligence in industry " in *Foundations of distributed artificial intelligence* G. O'Hare and N. R. Jennings, Eds.: John Wiley \& Sons, Inc., 1996, pp. 139-164.
[49] J. Meng, S. Helal, and S. Su, "An ad-hoc workflow system architecture based on mobile agents and rule-based processing," in *Proceedings of the 2000 international conference on artificial intelligence (ICAI2000)*, Las Vegas, 2000, pp. 245-251.
[50] S. McCready, "There is more than one kind of Workflow Software," *Computerworld,* vol. November 2, pp. 86–90, 1992.
[51] F. Leymann and D. Roller, *Production Workflow: Concepts and Techniques*. Upper Saddle River, NJ: Prentice Hall PTR, 2000.
[52] W. M. P. v. d. Aalst and K. M. v. Hee, *Workflow Management: Models, Methods, and Systems*. Cambridge, MA: MIT Press, 2002.
[53] G. J. Nutt, "The evolution towards flexible workflow systems," *Distributed Systems Engineering,* vol. 3, pp. 276–294, 1996.
[54] G. Vergivadis, "Inter-Organizational Workflow Management Systems," National Technical University of Greece, Athens, Greece, 2006.

[55] J. Shepherdson, S. Thompson, and B. Odgers, "Cross Organisational Workflow Co-ordinated by Software Agents," in *Cross-Organisational Workflow Management and Co-ordination*, San Francisco, USA, 1999.
[56] OASIS, "Web Services Business Process Execution Language (WSBPEL) TC," 2008, [Online]. Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel
[57] FIPA, "Interaction Protocols Specifications," 2008, [Online]. Available: http://www.fipa.org/repository/ips.php3
[58] M. N. Huhns, "Agents as Web services," *Internet Computing, IEEE,* vol. 6, pp. 93-95, 2002.
[59] B. T. R. Savarimuthu, M. Purvis, M. Purvis, and S. Cranefield, "Agent-based integration of Web Services with Workflow Management Systems," in *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems* The Netherlands 2005, pp. 1345-1346
[60] J. M. Vidal, P. Buhler, and C. Stahl, "Multiagent systems with workflows," *IEEE Internet Computing,* vol. 8, pp. 76-82, 2004.
[61] J. Korhonen, L. Pajunen, and J. Puustjarvi, "Automatic composition of Web service workflows using a semantic agent," in *Proceedings IEEE/WIC International Conference on Web Intelligence (WI 2003)*, Halifax, Canada, 2003, pp. 566-569.
[62] I. Foster, N. R. Jennings, and C. Kesselman, "Brain meets brawn:why grid and agents need each other," in *Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004. Proceedings of the Third International Joint Conference on*, 2004, pp. 8-15.
[63] A. Barker and R. Mann, "Flexible Service Composition," in *Cooperative Information Agents X*. vol. 4149, M. Klusch, M. Rovatsos, and T. R. Payne, Eds.: Springer Berlin / Heidelberg, 2006, pp. 446-460.
[64] Z. Zhao, A. Belloum, C. D. Laat, P. Adriaans, and B. Hertzberger, "Using Jade agent framework to prototype an e-Science workflow bus," in *Seventh IEEE International Symposium on Cluster Computing and the Grid, 2007. CCGRID 2007.* , Rio de Janeiro, Brazil 2007, pp. 655-660.
[65] L. Cao, M. Li, J. Cao, and Y. Wang, "Introduction to an Agent-Based Grid Workflow Management System," in *Parallel and Distributed Processing and Applications - ISPA 2005 Workshops*. vol. 3759, G. Chen, Y. Pan, M. Guo, and J. Lu, Eds., 2005, pp. 559-568.
[66] WfMC, "WfMC Standards, Workflow Reference Model " Report No: WfMC-TC-1003, *Workflow Management Coalition* 1995.
[67] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice.," *The Knowledge Engineering Review,* vol. 10, pp. 115–152, 1995.
[68] T. Winograd and F. Flores, *Understanding Computers and Cognition: A New Foundation for Design*. Reading, MA: Addison-Wesley, 1987.
[69] K. Palacz and D. Marinescu, "An agent-based workflow management system," in *Proc. AAAI Spring Symposium Workshop Bringing Knowledge to Business Processes*, Standford University, CA, 1999.
[70] A. Omicini, A. Ricci, and N. Zaghini, "Distributed Workflow upon Linkable Coordination Artifacts," in *Coordination Models and Languages*. vol. 4038, P. Ciancarini and H. Wiklicky, Eds.: Springer Berlin / Heidelberg, 2006, pp. 228-246.
[71] A. Ricci, A. Omicini, and E. Denti, "Virtual Enterprises and Worfklow Management as Agent Coordination Issues," *International Journal of Cooperative Information Systems,* vol. 11, pp. 355-379, 2002.

[72] P. Buhler and J. M. Vidal, "Enacting BPEL4WS Specified Workflows with Multiagent Systems," in *Proceedings of the Workshop on Web Services and Agent-Based Engineering*, 2004.
[73] U. M. Borghoff, P. Bottoni, P. Mussio, and R. Pareschi, "Reflective Agents for Adaptive Workflows," in *2nd International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM '97)*, London, U. K., 1997, pp. 405-420.
[74] B.-H. Ooi, "A Multi-agent Approach to Business Processes Management in an Electronic Market," in *Intelligent Agents and Multi-Agent Systems*. vol. 2891, J. Lee and M. Barley, Eds.: Springer Berlin / Heidelberg, 2003, pp. 1-12.
[75] T. Cai, P. Gloor, and S. Nog, "DartFlow: A Workflow Management System on the Web Using Transportable Agents," Report No: TR96-283, *Dept. of Computer Science, Dartmouth College*, Hanover, Technical Report 1996.
[76] Z. Budimac, D. Pesovic, M. Ivanovic, and N. Ibrajter, "Lessons Learned From the Implementation of a Workflow Management System Using Mobile Agents," *Novi Sad Journal of Mathematics,* vol. 36, pp. 65-79, 2006.
[77] A. Inamoto, "Agent oriented system approach for workflow automation," *International Journal of Production Economics,* vol. 60-61, pp. 327-335, 1999.
[78] D. Barbara, S. Mehrotra, and M. Rusinkiewicz, "INCAs: Managing Dynamic Workflows in Distributed Environments," *Journal of Database Management,* vol. 7, pp. 5-15, 1996.
[79] G. Kaiser and A. Dossick, "A Mobile Agent Approach to Lightweight Process Workflow," in *International Process Technology Workshop '99*, 1999.
[80] G. Valetto, G. Kaiser, and G. S. Kc, "A Mobile Agent Approach to Process-Based Dynamic Adaptation of Complex Software Systems," in *Software Process Technology: 8th European Workshop, EWSPT 2001, Witten, Germany, June 19-21, 2001, Proceedings*. vol. 2077, V. Ambriola, Ed.: Springer Berlin / Heidelberg, 2001, pp. 102-116.
[81] S. Helal, M. Wang, A. Jagatheesan, and R. Krithivasan, "Brokering Based Self Organizing E-Service Communities," in *Fifth International Symposium on Autonomous Decentralized Systems (ISADS)*, Dallas, Texas, 2001, pp. 349-356.
[82] M. B. Blake, "Coordinating multiple agents for workflow-oriented process orchestration," *Information Systems and E-Business Management,* vol. 1, pp. 387-404, 2003.
[83] Q. Chen, U. Dayal, M. Hsu, and M. Griss, "Dynamic-Agents, Workflow and XML for E-Commerce Automation," in *EC-Web 2000*. vol. 1875, K. Bauknecht, S. K. Madria, and G. Pernul, Eds. London, UK: Springer Berlin / Heidelberg, 2000, pp. 314-323.
[84] L. Ehrler, M. Fleurke, M. Purvis, and B. T. R. Savarimuthu, "Agent-based workflow management systems (WfMSs)," *Information Systems and E-Business Management,* vol. 4, pp. 5-23, 2006.
[85] C. A. Marın and R. F. Brena, "Multiagent Architecture for Decentralized Workflow Process Execution," Report No: CSI-RI-002, *Center for Intelligent Systems Tecnologico de Monterrey*, Monterrey, Mexico, Technical Report March 9th 2005.
[86] J.-W. Wang, C.-C. Li, and F.-J. Wang, "Dynamic activities on an agent-based workflow management system," in *The 3rd ACS/IEEE International Conference on Computer Systems and Applications, 2005.* , 2005, p. 122.
[87] S. Das, K. Kochut, J. Miller, A. Sheth, and D. Worah, "ORBWork:A Reliable Distributed CORBA-based Workflow Enactment System for METEOR 2," Report No: UGA-CS-TR-97-001, *University of Georgia* 1997.

[88] H. Stormer, "A Flexible Agent-Based Workflow System," in *Fifth International Conference on Autonomous Agents* Montreal, Canada, 2001.
[89] H. Gou, B. Huang, W. Liu, S. Ren, and Y. Li, "An agent-based approach for workflow management," in *IEEE International Conference on Systems, Man, and Cybernetics, 2000* Nashville, TN, USA, 2000, pp. 292-297.
[90] Q. Xu, R. Qiu, and F. Xu, "Agent-based workflow approach to the design and development of cross-enterprise information systems," in *IEEE International Conference on Systems, Man and Cybernetics, 2003.* , Washington, D.C., USA, 2003, pp. 2633- 2638.
[91] M. B. Blake, "An agent-based cross-organizational workflow architecture in support of Web services," in *Eleventh IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, WET ICE 2002*, Pittsburgh, Pennsylvania, USA, 2002, pp. 176- 181.
[92] X. Manmin and L. Huaicheng, "Cooperative software agents for workflow management system," in *Fifth Asia-Pacific Conference On Communications and Fourth Optoelectronics and Communications Conference APCC/OECC '99*, Beijing, China, 1999, pp. 1063-1067.
[93] J. Cao, J. Wang, S. Zhang, and M. Li, "A dynamically reconfigurable system based on workflow and service agents," *Engineering Applications of Artificial Intelligence,* vol. 17, pp. 771-782, 2004.
[94] M. Crowe and S. Kydd, "Agents and suggestions in a Web-based dynamic workflow model," *Automation in Construction,* vol. 10, pp. 639-643, 2001.
[95] I. Hawryszkiewycz and J. Debenham, "A Workflow System Based on Agents," in *Database and Expert Systems Applications*. vol. 1460, G. Quirchmayr, E. Schweighofer, and T. J. M. Bench-Capon, Eds.: Springer Berlin / Heidelberg, 1998, pp. 135–143.
[96] H. Tarumi, K. Kida, Y. Ishiguro, K. Yoshifu, and T. Asakura, "WorkWeb system—multi-workflow management with a multi-agent system " in *Supporting group work: the integration challenge*, Phoenix, Arizona, United States 1997, pp. 299-308.
[97] C.-J. Huang, C. V. Trappey, and C. C. Ku, "A JADE-based Autonomous Workflow Management System for Collaborative IC Design," in *11th International Conference on Computer Supported Cooperative Work in Design, 2007. CSCWD 2007.* , Melbourne, Australia 2007, pp. 777-782.
[98] T. Madhusudan, "An agent-based approach for coordinating product design workflows," *Computers in Industry,* vol. 56, pp. 235-259, 2005.
[99] H. Yanli, Y. Haicheng, H. Weiping, Z. Wei, and H. Xinping, "Flexible Workflow Driven Job Shop Manufacturing Execution and Automation Based on Multi Agent System," in *IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 2006. IAT '06.* , Hong Kong, China 2006, pp. 695-699.
[100] T. Aye and K. M. L. Tun, "A Collaborative Mobile Agent-based Workflow System," in *6th Asia-Pacific Symposium on Information and Telecommunication Technologies, 2005. APSITT 2005* Yangon, Myanmar, 2005, pp. 59-65.
[101] G. Kappel, S. Rausch-Schott, and W. Retschitzegger, "A framework for workflow management systems based on objects, rules and roles " *ACM Computing Surveys,* vol. 32 p. 27, 2000.
[102] J. Debenham, "Constructing an intelligent multi-agent workflow system," in *Advanced Topics in Artificial Intelligence*. vol. 1502: Springer Berlin / Heidelberg, 1998, pp. 119-130.
[103] H. Stormer and K. Knorr, "PDA- and Agent-based Execution of Workflow Tasks," in *Informatik 2001 Conference*, Vienna, Austria, 2001, pp. 968-973.

[104] G. Q. Huang, J. Huang, and K. L. Mak, "Agent-based workflow management in collaborative product development on the Internet," *Computer-Aided Design,* vol. 32, pp. 133-144, 2000.

[105] S. Aknine and S. Pinson, "Agent Oriented Conceptual Modeling of Parallel Workflow Systems," in *Multiple Approaches to Intelligent Systems*. vol. 1611: Springer Berlin / Heidelberg, 1999, pp. 500-509.

[106] J.-Y. Kuo, "A document-driven agent-based approach for business processes management," *Information and Software Technology,* vol. 46, pp. 373-382, 2004.

[107] E. Gudes and A. Tubman, "AutoWF--A secure Web workflow system using autonomous objects," *Data & Knowledge Engineering,* vol. 43, pp. 1-27, 2002.

[108] D. Xu and H. Wang, "Multi-agent collaboration for B2B workflow monitoring," *Knowledge-Based Systems,* vol. 15, pp. 485-491, 2002.

[109] J. Liu, S. Zhang, and J. Hu, "A case study of an inter-enterprise workflow-supported supply chain management system," *Information & Management,* vol. 42, pp. 441-454, 2005.

[110] Z. Zhao, A. Belloum, C. de Laat, P. Adriaans, and B. Hertzberger, "Distributed execution of aggregated multi domain workflows using an agent framework," in *IEEE Congress on Services, 2007*, Salt Lake City, UT, USA, 2007, pp. 183-190.

[111] H. Zhuge, J. Chen, Y. Feng, and X. Shi, "A federation-agent-workflow simulation framework for virtual organisation development," *Information & Management,* vol. 39, pp. 325-336, 2002.

[112] H. Stormer, K. Knorr, and J. H. P. Eloff, "A model for security in agent-based workflows," *INFORMATIK / INFORMATIQUE,* pp. 24-29, 2000.

[113] S. Wang, W. Shen, and Q. Hao, "Agent based workflow ontology for dynamic business process composition," in *Proceedings of the Ninth InternationaI Conference on Computer Supported Cooperative Work in Design*, Coventry, UK, 2005, pp. 452-457.

[114] H. Jingjing, C. Yuanda, and Z. Zhen, "Workflow management system based on agent for virtual enterprise," in *The 8th International Conference on Computer Supported Cooperative Work in Design, 2004*, Xiamen China, 2004, pp. 373-378.

[115] J. Korhonen, L. Pajunen, and J. Puustjärvi, "Using Transactional Workflow Ontology in Agent Cooperation," in *First EurAsian Conference on Advances in Information and Communication Technology (EURASIA-ICT 2002)* Tehran, Iran, 2002.

[116] A. Schill and C. Mittasch, "Workflow management systems on top of OSF DCE and OMG CORBA," *Distributed Systems Engineering,* vol. 3, pp. 250–262, 1996.

[117] H. Wang and D. Xu, "Collaborative multi-agents for workflow management," in *34th Annual Hawaii International Conference on System Sciences, 2001.* , Maui, Hawaii, 2001, p. 9 pp.

[118] R. G. Smith, "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver," *IEEE Trans. Comput.,* vol. 29, pp. 1104-1113, 1980.

[119] L. Biegus and C. Branki, "InDiA: a framework for workflow interoperability support by means of multi-agent systems," *Engineering Applications of Artificial Intelligence,* vol. 17, pp. 825-839, 2004.

[120] A. K. Jain, M. I. V. Aparico, and M. P. Singh, "Agents for process coherence in virtual enterprises," *Communications of the ACM,* vol. 42, pp. 62-69, 1999.

[121] B. T. R. Savarimuthu, M. Purvis, and M. Purvis, "Different Perspectives on Modeling Workflows in an Agent Based Workflow Management System," in *Knowledge-Based Intelligent Information and Engineering Systems*. vol. 3684, R. Khosla, R. J. Howlett, and L. C. Jain, Eds.: Springer Berlin / Heidelberg, 2005, pp. 208-214.

[122] D. Kaponis, L. Kamara, J. Pitt, and K. Clark, "A mechanism for trusted agent-based workflow transport," in *Engineering Societies in the Agents World '03* London, UK, 2003.
[123] J. Hickie, J. Kennedy, G. Koudouridis, V. Ouzounis, and M. Studley., "A Scaleable Heterogeneous Architecture for Agent-Oriented Workflow Management.," in *International Joint Conference on Artificial Intelligence 1999* Stockholm, 1999.
[124] M. Sierhuis, W. J. Clancey, and R. J. J. V. Hoof, "Brahms: a multi-agent modelling environment for simulating work processes and practices," *International Journal of Simulation and Process Modelling,* vol. 3, pp. 134 - 152, 2007.
[125] J. Debenham, "Who does what in a multiagent system for emergent process management," in *Ninth Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS' 02)*, Lund, Sweden, 2002, pp. 35-40.
[126] Z. Budimac, M. Ivanovic, and A. Popovic, "Workflow Management System Using Mobile Agents," in *Advances in Databases and Information Systems: Third East European Conference, ADBIS'99, Maribor, Slovenia, September 1999. Proceedings*. vol. 1691, J. Eder, I. Rozman, and TatjanaWelzer, Eds.: Springer Berlin / Heidelberg, 1999, pp. 168-178.
[127] P. T. Harker and L. H. Ungar, "A market-based approach to workflow automation," in *Proceedings of NSF. Workshop on Workflows and Process Automation in Information Systems: State of the Art and Future Directions.* , Athens, GA, USA, 1996.
[128] H. Stormer, "A Flexible Agent-Based Workflow System," in *Fifth International Conference on Autonomous Agents* Montreal, Canada, 2001.
[129] J. P. Moore, R. Inder, P. W. H. Chung, A. Macintosh, and J. Stader, "Who Does What? Matching Agents to Tasks in Adaptive Workflow," in *International Conference on Enterprise Information Systems*, 2000, pp. 181-185.
[130] P. Delias, A. Doulamis, and N. Matsatsinis, "A Joint Optimization Algorithm for Dispatching Tasks in Agent-based Workflow Management Systems," in *Proceedings of the 10th International Conference on Enterprise Information Systems, ICEIS 2008*, Barcelona, Spain, 2008, pp. 199-206.
[131] P. Delias, A. Doulamis, and N. Matsatsinis, "Optimizing Resource Conflicts in Workflow Management Systems," *IEEE Transactions on Knowledge and Data Engineering,* (accepted) 2008.
[132] A. Padalkar, P. Nabar, S. Arora, and P. Naik, "SWIFT:scalable workflow management system using mobile agents," Bombay: Kanwal Rekhi School of Information Technology, 2000.
[133] C.-J. Huang, A. J. C. Trappey, and Y.-H. Yao, "Developing an agent-based workflow management system for collaborative product design," *Industrial Management & Data Systems,* vol. 106, pp. 680 - 699, 2006.
[134] M. B. Blake, "Agent-Based Communication For Distributed Workflow Management Using JINI Technologies," *International Journal on Artificial Intelligence Tools,* vol. 12, pp. 81-99, 2003.
[135] J. Debenham and S. Simoff, "Intelligent Agents that Span the Process Management Spectrum," in *3rd International IEEE Conference on Intelligent Systems, 2006* London 2006, pp. 386-389.
[136] A. T.-I. Yaung, "Workflow agent for a multimedia database system." vol. US 6,405,215 B1 USA: International Business Machines Corp., 2002.
[137] M. L. Roberts and P. D. Berger, *Direct Marketing Management*: Prentice Hall, 1999.

[138] R. A. Greve, R. Sharda, M. Kamath, and A. Kadam, "Modelling and analysis of e-mail management for improved customer relationship management," *International Journal of Simulation and Process Modelling,* vol. 1, pp. 125 - 137, 2005.

[139] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE: a FIPA2000 compliant agent development environment," in *Proceedings of the fifth international conference on Autonomous agents* Montreal, Quebec, Canada: ACM, 2001, pp. 216-217.

[140] FIPA, "FIPA ACL Message Structure Specification," Report No: SC00061G, *Foundation for Intelligent Physical Agents*, Geneva, Switzerland 2002.

[141] M. D. Sadek, "Attitudes Mentales et Interaction Rationnelle: Vers une Theorie Formelle de la Communication," These de Doctorat Informatique, Universite de Rennes I, France, 1991.

[142] FIPA, "FIPA Interaction Protocol Library Specification," Report No: DC00025F, *Foundation of Intelligent Physical Agents*, Geneva, Switzerland 2003.

[143] FIPA, "FIPA Contract Net Interaction Protocol Specification," Report No: SC00029H, *Foundation for Intelligent Physical Agents*, Geneva, Switzerland 2002.

[144] WfMC, "XPDL - XML Process Definition Language," Report No: WFMC-TC-1025, *Workflow Management Coalition*, Hingham, MA, USA, WfMC Specification Documents 2008.

[145] WFMC, "XPDL Support & Resources," 2009, [Online]. Available: http://www.wfmc.org/xpdl.html

[146] B. Chandrasekaran, J. R. Josephson, and V. R. Benjamins, "What Are Ontologies, and Why Do we Need Them?," *IEEE Intelligent Systems,* vol. 14, pp. 20-26, 1999.

[147] P. Eeles, K. A. Houston, and W. Kozaczynski, *Building J2EE™ Applications with the Rational Unified Process*. Indianapolis: Addison-Wesley Professional, 2003.

[148] P.-P. Grassé, "La reconstruction du nid et les coordinations interindividuelles chezBellicositermes natalensis etCubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs," *Insectes Sociaux,* vol. 6, pp. 41-80, 1959.

[149] M. Dorigo, E. Bonabeaub, and G. Theraulaz, "Ant algorithms and stigmergy," *Future Generation Computer Systems,* vol. 16, pp. 851-871, 2000.

[150] K. Schmidt and I. Wagner, "Ordering systems: Coordinative practices and artifacts in architectural design and planning," *Computer Supported Cooperative Work,* vol. 13, pp. 349-408, 2004.

[151] H. Van Dyke Parunak, "A Survey of Environments and Mechanisms for Human-Human Stigmergy," in *Environments for Multi-Agent Systems II*, DannyWeyns, H. V. D. Parunak, and F. Michel, Eds.: Springer Berlin / Heidelberg, 2006, pp. 163-186.

[152] A. Ricci, A. Omicini, M. Viroli, L. Gardelli, and E. Oliva, "Cognitive Stigmergy: Towards a Framework Based on Agents and Artifacts," in *Environments for Multi-Agent Systems III*, DannyWeyns, H. V. D. Parunak, and F. Michel, Eds.: Springer Berlin / Heidelberg, 2007, pp. 124-140.

[153] M. Aksit, K. Wakita, J. Bosch, L. Bergmans, and A. Yonezawa, "Abstracting Object Interactions Using Composition Filters," in *ECOOP 1993 Workshop on Object-Based Distributed Programming*, 1993, pp. 152–184.

[154] W. M. P. v. d. Aalst, "The Application of Petri Nets to Workflow Management," *Journal of Circuits, Systems, and Computers,* vol. 8, pp. 21-66, 1998.

[155] N. Adam, V. Atluri, and W. Huang, "Modeling and analysis of workflows using Petri nets," *Journal of Intelligent Information Systems,* vol. 10, pp. 131-158, 1998.

[156] H. Davulcu, M. Kifer, C. R. Ramakrishnan, and I. V. Ramakrishnan, "Logic based modeling and analysis of workflows," in *ACM Symposium on Principles of Database Systems*, Seattle, Washington, 1998, pp. 25-33.
[157] M. P. Singh, "Synthesizing distributed constrained events from transactional workflow specifications," in *Proceedings of 12th IEEE International Conference on Data Engineering,* New Orleans, LA, 1996, pp. 616-623.
[158] W. Du, J. Davis, Y. Huang, and M. Shan, "Enterprise workflow resource management," in *International Workshop on Research Issues in Data Engineering*, Sydney, Australia, 1999, pp. 108-115.
[159] Y. Huang and M. Shan, "Policies in a resource manager of workflow systems: Modeling, enforcement and management.," in *International Conference on Data Engineering*, 1999.
[160] M. z. Muhlen, "Resource modeling in workflow applications," in *Workflow Management Conference*, Muenster, Germany, 1999, pp. 137-153.
[161] N. Russell, A. H. M. t. Hofstede, D. Edmond, and W. M. P. v. d. Aalst, "WORKFLOW RESOURCE PATTERNS."
[162] A. Bajaj and S. Ram, "SEAM: A state-entity-activity-model for a well-defined workflow development methodology," *Knowledge and Data Engineering, IEEE Transactions on,* vol. 14, pp. 415-431, 2002.
[163] W. M. P. v. d. Aalst, "Three good reasons for using a Petri-net-based workflow management system.," in *Information and Process Integration in Enterprises: Rethinking Documents*. vol. 428, S. K. T. Wakayama, C.M. Khoong, S. Navathe, J. Yates, Ed. Boston, MA: Kluwer Academic Publishers, 1998, pp. 161–182.
[164] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems (2nd ed.).* New Jersey: Prentice Hall, 2002.
[165] G. Greco, A. Guzzo, L. Ponieri, and D. Sacca, "Discovering expressive process models by clustering log traces," *Knowledge and Data Engineering, IEEE Transactions on,* vol. 18, pp. 1010-1027, 2006.
[166] B. Joonsoo, B. Hyerim, K. Suk-Ho, and K. Yeongho, "Automatic control of workflow processes using ECA rules," *Knowledge and Data Engineering, IEEE Transactions on,* vol. 16, pp. 1010-1023, 2004.
[167] K. v. Hee, A. Serebrenik, N. Sidorova, and M. Voorhoeve, "Soundness of Resource-Constrained Workflow Nets," in *Applications and Theory of Petri Nets 2005*. vol. 3536: Springer Berlin / Heidelberg, 2005, pp. 250-267.
[168] H. A. Reijers, "Resource Allocation in Workflows," in *Design and Control of Workflow Processes: Business Process Management for the Service Industry*. vol. 2617: Springer Berlin / Heidelberg, 2003, pp. 177-206.
[169] I. Nakic and K. Veselic, "Wielandt and Ky-Fan Theorem for Matrix Pairs," *Linear Algebra and its Applications,* vol. 369, pp. 77-73, August 2003 2003.
[170] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On Spectral Clustering: Analysis and an algorithm," in *Advances in Neural Information Processing Systems 14*: MIT Press, 2001, pp. 849--856.
[171] E. L. Johnson, A. Mehrotra, and G. L. Nemhauser, "Min-cut clustering," *Mathematical Programming,* vol. 62, pp. 133-151, 1993.
[172] S. Even, *Graph Algorithms*. NY, USA: W. H. Freeman & Co., 1979.

List of candidate's Publications

Related to Thesis Topic

P. Delias and N. F. Matsatsinis, "Multiple Criteria Decision Making in Multi-Agent Systems," in *The 18th International Conference on Multiple Criteria Decision Making - MCDM 2006* Chania, Greece, 2006.

P. Delias and N. F. Matsatsinis, "The multiple criteria paradigm as a background for agent methodologies," in *8th Annual International Workshop "Engineering Societies in the Agents World"*, Athens, Greece, 2007, pp. 227-237.

P. Delias, "Workflow Management Systems and Agents. Do They Fit Together?," in *6th Doctoral Consortium on Enterprise Information Systems, DCEIS 2008*, Barcelona, Spain, 2008, pp. 3-11.

P. Delias, A. Doulamis, and N. Matsatsinis, "A Joint Optimization Algorithm for Dispatching Tasks in Agent-based Workflow Management Systems," in *Proceedings of the 10th International Conference on Enterprise Information Systems, ICEIS 2008*, Barcelona, Spain, 2008, pp. 199-206.

P. Delias, A. Doulamis, and N. Matsatsinis, "Optimizing Resource Conflicts in Workflow Management Systems," *IEEE Transactions on Knowledge and Data Engineering,* (accepted) 2008.

P. Delias, K. Ntalianis, A. Doulamis, and N. Matsatsinis, "Automating Marketing Campaign Management Through an Agent-based Workflow Management System," in *13th WSEAS International Conference on Communications*, Rodos (Rhodes) Island, Greece, 2009, pp. 37-45.

P. Delias, A. Doulamis, and N. Matsatsinis, "What Agents Can Do in Workflow Management Systems," *IEEE Transactions on Knowledge and Data Engineering,* (under review) 2009.

Related to the sponsor program topic

N. F. Matsatsinis, K. Lakiotaki, and P. Delias, "A System based on Multiple Criteria Analysis for Scientific Paper Recommendation," in PCI' 2007 11th Panhellenic Conference in Informatics, Patras, Greece, 2007, pp. 135-149.

K. Lakiotaki, P. Delias, V. Sakkalis, and N. Matsatsinis, "User profiling based on multi-criteria analysis: the role of utility functions," Operational Research, vol. 9, pp. 3-16, 2009.

Appendix A

English	Ελληνικά
Activity	(Επιμέρους) εργασία
Actuators	Μηχανισμοί κίνησης
Adjacency matrix	Πίνακας γειτνίασης
Agent-based WFMS	Βασισμένα σε πράκτορες ΣΔΡΕ
Agent-enhanced WFMS	Ενισχυμένα από πράκτορες ΣΔΡΕ
Agent-involved WFMS	ΣΔΡΕ με εμπλοκή της τεχνολογίας πρακτόρων
Aggregation	Συγκρότηση
Audit Management	Επιστασία
Build-time	Χρόνος κατασκευής
Business Process	Επιχειρηματική διαδικασία
Common Interpretation	Από κοινού ερμηνεία
Coordination	Συντονισμός
Data Interchange	Διαμοιρασμός δεδομένων
Direct mail campaign	Διαφημιστική εκστρατεία διά αλληλογραφίας
Encapsulation	Ενθυλάκωση
graph	Γράφημα
Inheritance	Κληρονομικότητα
Instance	Στιγμιότυπο
Interaction Protocol	Πρωτόκολλο αλληλεπίδρασης
Interface	Διεπαφή
Intervention	Παρέμβαση
Manual Activity	Χειροκίνητη εργασία
Message exchange pattern	Μοτίβο ανταλλαγής μηνυμάτων
Mobility	Κινητικότητα
non-interruptible	μη διακοπτόμενου
non-preemptable	μη προεκχωρήσιμου
Proactive	Προνοητικός
Procedural	Διαδικαστικό
Process Definition	Ορισμός Διαδικασίας
Prototype	Αρχέτυπο
Reactive	Αναδραστικός

Resource Allocation	Εκχώρηση πόρων
Runtime	Χρόνος εκτέλεσης
Runtime Control Environment	Περιβάλλον Ελέγχου Εκτέλεσης
Scheduling	Χρονοπρογραμματισμός
Statefulness	Διατήρηση Κατάστασης
Transition	Μετάβαση
User Interface	Διεπιφάνεια Χρήστη
Workflow	Ροή εργασιών
Workflow Enactment Service	Υπηρεσία εκτέλεσης ροών εργασιών
Workflow Engine	Μηχανή ΣΔΡΕ
Workflow Management System (WFMS)	Σύστημα Διαχείρισης Ροών Εργασιών (ΣΔΡΕ)
Workflow Monitoring	Επίβλεψη ροών εργασιών
Worklist	Λίστα εργασιών

Appendix B

Marketing Workflow Source Code Documentation

Package Summary		Page
agents		137
agents.contactCenter		155
applications.contactCenter		164
applications.directMail		168
generic		190
marketing.wf.gui		209
monitoring		228
ontology		235
ontology.beans		245
util		253
util.objects		260
util.ws		263
util.ws.crm		280
workflows		287
workflows.auxiliary		332

Package agents

Class Summary		Page
ApplicationEngineAgent	The Application Engine is used in case of ontology-based workflow execution It receives requests from other agents and serve the actions related to the domain ontology either by its own or by delegating the actions to other agents.	137
ApplicationEngineAgent.ApplicationEngineRequestServer	A cyclic behaviour that listens if there are any requests related to a specific domain ontology.	141
MarCom	The class for the Marketing Communicator agent.	142
MarkAssistant	The class for the Marketing Assistant agent.	144
MarketingDirector	The class for the Marketing Director agent.	146
MediaVendor	The class to represent a Vendor by an agent.	148
MediaVendor.ProposalServer	Inner Class ProposalServer This class serves the incoming requests for media production	151
ProductManager	The class for the Product Manager Agent.	152

Class ApplicationEngineAgent

agents

```
java.lang.Object
  └ jade.core.Agent
      └ com.tilab.wade.commons.WadeAgentImpl
          └ com.tilab.wade.performer.WorkflowEngineAgent
              └ agents.ApplicationEngineAgent
```

All Implemented Interfaces:

Runnable, jade.util.leap.Serializable, Serializable, jade.core.TimerListener, com.tilab.wade.commons.WadeAgent

---

```
public class ApplicationEngineAgent
extends com.tilab.wade.performer.WorkflowEngineAgent
```

The Application Engine is used in case of ontology-based workflow execution It receives requests from other agents and serve the actions related to the domain ontology either by its own or by delegating the actions to other agents.

Author:

Pavlos Delias

Nested Class Summary		Page
private class	ApplicationEngineAgent.ApplicationEngineRequestServer A cyclic behaviour that listens if there are any requests related to a specific domain ontology.	141

Nested classes/interfaces inherited from class com.tilab.wade.performer.WorkflowEngineAgent
WorkflowEngineAgent.WorkflowExecutor

Nested classes/interfaces inherited from class jade.core.Agent
`Agent.Interrupted`

Field Summary		Page
`private String`	**BATCH_MAIL_REQUIREMENT**	*140*
`Connection`	**conn**	*139*
`private String`	**CONTACT_SCHEDULE_REQUIREMENT**	*140*
`Statement`	**ins**	*139*
`private jade.util.leap.List`	**mailsReceived**	*140*
`private int`	**processId**	*139*
`ResultSet`	**rs**	*139*
`private static long`	**serialVersionUID**	*139*
`Statement`	**stmt**	*139*

Fields inherited from class com.tilab.wade.performer.WorkflowEngineAgent
`ACTIVE_CNT_ATTRIBUTE, BUSY_EXECUTORS_ATTRIBUTE, codec, DEFAULT_WORKFLOW_TIMEOUT_ATTRIBUTE, DONE_STATUS, ENQUEUED_CNT_ATTRIBUTE, EXECUTING_STATUS, executors, IDLE_STATUS, onto, POOL_SIZE_ATTRIBUTE, SUSPENDED_STATUS, tbf, TERMINATING_STATUS, THREAD_CNT_ATTRIBUTE, WAITING_STATUS, WORKFLOW_CNT_ATTRIBUTE`

Fields inherited from class com.tilab.wade.commons.WadeAgentImpl
`arguments, myLogger`

Fields inherited from class jade.core.Agent
`AP_ACTIVE, AP_DELETED, AP_IDLE, AP_INITIATED, AP_MAX, AP_MIN, AP_SUSPENDED, AP_WAITING, D_ACTIVE, D_MAX, D_MIN, D_RETIRED, D_SUSPENDED, D_UNKNOWN, MSG_QUEUE_CLASS`

Fields inherited from interface com.tilab.wade.commons.WadeAgent
`ADMINISTRATOR_ROLE, AGENT_CLASSNAME, AGENT_LOCATION, AGENT_OWNER, AGENT_POOL, AGENT_ROLE, AGENT_TYPE, BCA_AGENT_TYPE, CONFIGURATION_AGENT_TYPE, CONTROL_AGENT_TYPE, DUMP_ARGUMENTS, HOSTADDRESS, HOSTNAME, JADE_ADDITIONAL_ARGS, JADE_PROFILE, JAVA_PROFILE, MDB_AGENT_TYPE, MESSAGE_QUEUE_SIZE_ATTRIBUTE, NONE_OWNER, NULL, RAA_AGENT_TYPE, RESTARTING, STARTUP_TIME_ATTRIBUTE, TRANSIENT_AGENT_ARGUMENT, WFENGINE_AGENT_TYPE, WORKFLOW_EXECUTOR_ROLE`

Constructor Summary	Page
ApplicationEngineAgent`()`	*140*

Method Summary		Page
`protected void`	**agentSpecificSetup**`()`	*140*
`int`	**getProcessId**`()`	*141*
`Mail`	**processMessage**`(Message message)` Transforms an ACLMessage to a Mail object.	*140*
`void`	**receiveMail**`(String popServer, String popUser, String popPassword)` This method is used to fetch messages and process them from a specific account on a specific Server.	*140*
`private String`	**saveList2XL**`(jade.util.leap.List items)` A method that save mail items to an Excel File	*141*
`private String`	**saveWorklist2XL**`(Worklist todo)` A method that saves a `Worklist` to an Excel File	*141*

private void	**sendNotification**(jade.content.onto.basic.Action actExpr, jade.lang.acl.ACLMessage request, int performative, Object result) This method sends back to the requester the result of an action in a uniform way regardless of whether or not the action succeeded.	*140*
void	**serveAddWorklist**(AddWorklist action, jade.content.onto.basic.Action actExpr, jade.lang.acl.ACLMessage msg) Serves the AddWorklist action of the ContactCenter Ontology.	*140*
void	**serveReceiveMails**(ReceiveMails action, jade.content.onto.basic.Action actExpr, jade.lang.acl.ACLMessage msg) Serves the ReceiveMails action of the Contact Center Ontology.	*140*
void	**serveRequestsOf**(RequestsOf action, jade.content.onto.basic.Action actExpr, jade.lang.acl.ACLMessage msg) A method used for audit purposes.	*141*
void	**serveSendMailBatch**(SendMailBatch action, jade.content.onto.basic.Action actExpr, jade.lang.acl.ACLMessage msg) Serves the SendMailBatch Action of the ContactCenterOntology.	*140*
void	**serveSetProcess**(SetProcess action, jade.content.onto.basic.Action actExpr, jade.lang.acl.ACLMessage msg) This method is used for the engine to share the current process Id with the GUI	*140*
void	**setProcessId**(int processId)	*141*
private void	**updateDB**(String file, String req) Updates the DB rows by associating files with requirements	*141*

Methods inherited from class com.tilab.wade.performer.WorkflowEngineAgent
adjustControlInfo, afterMove, beforeMove, createExecutionId, createGenericError, dequeue, enqueue, getActiveCnt, getBusyExecutors, getClassLoaderIdentifier, getCommitTimeout, getDefaultWorkflowTimeout, getEnqueuedCnt, getExecutionContext, getExecutorsTableStatus, getLanguage, getOntology, getPoolSize, getRollbackTimeout, getSuspendedCnt, getThreadCnt, getWorkflowClassLoader, getWorkflowCnt, handleAbortedTransaction, handleBeginActivity, handleBeginApplication, handleBeginWorkflow, handleCleanupWorkflow, handleCommittedTransaction, handleCompletedSubflow, handleDelegatedSubflow, handleEndActivity, handleEndApplication, handleEndWorkflow, handleError, handleEvent, handleFailedTransaction, handleIncomingWorkflow, handleOpenedTransaction, handleUnknownAction, isWorking, loadRollbackWorkflow, removeConversation, removeFromQueue, reply, serveExecuteWorkflow, serveGetPoolSize, serveGetSessionStatus, serveGetWRD, serveKillWorkflow, serveSetControlInfo, serveSetPoolSize, serveSetWRD, setPoolSize, takeDown

Methods inherited from class com.tilab.wade.commons.WadeAgentImpl
getAttributes, getDFDescription, getManagementResponder, getMessageQueueSize, getOwner, getRole, getStartupTime, getType, setAttributes, setup

Methods inherited from class jade.core.Agent
addBehaviour, afterClone, beforeClone, blockingReceive, blockingReceive, blockingReceive, blockingReceive, changeStateTo, clean, createMessageQueue, doActivate, doClone, doDelete, doMove, doSuspend, doTimeOut, doWait, doWait, doWake, getAgentState, getAID, getAMS, getArguments, getBootProperties, getContainerController, getContentManager, getCurQueueSize, getDefaultDF, getHap, getHelper, getLocalName, getName, getO2AObject, getProperty, getQueueSize, getState, here, isRestarting, join, notifyChangeBehaviourState, notifyRestarted, postMessage, putBack, putO2AObject, receive, receive, removeBehaviour, removeTimer, restartLater, restore, restoreBufferedState, run, send, setArguments, setEnabledO2ACommunication, setGenerateBehaviourEvents, setO2AManager, setQueueSize, waitUntilStarted, write

Field Detail

private static final long **serialVersionUID**

Connection **conn**

Statement **stmt**

Statement **ins**

ResultSet **rs**

private int **processId**

```
private jade.util.leap.List mailsReceived
```

```
private final String BATCH_MAIL_REQUIREMENT
```

```
private final String CONTACT_SCHEDULE_REQUIREMENT
```

Constructor Detail

```
public ApplicationEngineAgent()
```

Method Detail

```
protected void agentSpecificSetup()
                           throws com.tilab.wade.commons.AgentInitializationException
```

Overrides:

`agentSpecificSetup` in class `com.tilab.wade.performer.WorkflowEngineAgent`

Throws:

`com.tilab.wade.commons.AgentInitializationException`

```
public void receiveMail(String popServer,
                        String popUser,
                        String popPassword)
```

This method is used to fetch messages and process them from a specific account on a specific Server. The server must be a POP3 one.

```
public Mail processMessage(Message message)
```

Transforms an ACLMessage to a Mail object.

Returns:

Mail

```
private void sendNotification(jade.content.onto.basic.Action actExpr,
                              jade.lang.acl.ACLMessage request,
                              int performative,
                              Object result)
```

This method sends back to the requester the result of an action in a uniform way regardless of whether or not the action succeeded. This informative message is required to match the FIPA REQUEST Interaction Protocol

Parameters:

`actExpr` - The Action expression that embedded the served action
`request` - The message that embedded the request to serve the action
`performative` - The ACL performative to use in the reply
`result` - The result (if any) produced by the action in case of success or an error code in case of failure.

```
public void serveAddWorklist(AddWorklist action,
                             jade.content.onto.basic.Action actExpr,
                             jade.lang.acl.ACLMessage msg)
```

Serves the AddWorklist action of the ContactCenter Ontology. Ultimately, it sends a message to an agent containing the filepath of the file that represents the agent's worklist

---

```
public void serveSetProcess(SetProcess action,
                            jade.content.onto.basic.Action actExpr,
                            jade.lang.acl.ACLMessage msg)
```

This method is used for the engine to share the current process Id with the GUI

```
public void serveSendMailBatch(SendMailBatch action,
                               jade.content.onto.basic.Action actExpr,
                               jade.lang.acl.ACLMessage msg)
```

Serves the SendMailBatch Action of the ContactCenterOntology. Ultimately, it sends a message which contains the path of the file that stores all the mails that have been received.

```
public void serveReceiveMails(ReceiveMails action,
                              jade.content.onto.basic.Action actExpr,
                              jade.lang.acl.ACLMessage msg)
```

Serves the ReceiveMails action of the Contact Center Ontology. Actually, it calls the `receiveMail(String, String, String)` method passing the arguments specified in the request mesage.

```
public void serveRequestsOf(RequestsOf action,
                            jade.content.onto.basic.Action actExpr,
                            jade.lang.acl.ACLMessage msg)
```

A method used for audit purposes. It queries the DB and returns all messages exchanged between two agents, specified within the `RequestsOf.RequestsOf()` action

---

```
private String saveWorklist2XL(Worklist todo)
```

A method that saves a `Worklist` to an Excel File

Returns:
file path

```
private String saveList2XL(jade.util.leap.List items)
```

A method that save mail items to an Excel File

Returns:
String - The file name of the saved file

```
private void updateDB(String file,
                      String req)
```

Updates the DB rows by associating files with requirements

```
public void setProcessId(int processId)
```

```
public int getProcessId()
```

Class ApplicationEngineAgent.ApplicationEngineRequestServer

agents

```
java.lang.Object
  └ jade.core.behaviours.Behaviour
      └ jade.core.behaviours.SimpleBehaviour
          └ jade.core.behaviours.CyclicBehaviour
              └ agents.ApplicationEngineAgent.ApplicationEngineRequestServer
```

All Implemented Interfaces:
jade.util.leap.Serializable, Serializable

Enclosing class:
ApplicationEngineAgent

```
private class ApplicationEngineAgent.ApplicationEngineRequestServer
extends jade.core.behaviours.CyclicBehaviour
```

A cyclic behaviour that listens if there are any requests related to a specific domain ontology. If any, then the agent decode the message and according to the action that the request specifies, it serves a different method

Author:
Pavlos Delias

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
Behaviour.RunnableChangedEvent

Field Summary		Page
private jade.lang.acl.MessageTemplate	**template**	*142*

Fields inherited from class jade.core.behaviours.Behaviour
myAgent, myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING

Constructor Summary		Page
private	**ApplicationEngineAgent.ApplicationEngineRequestServer**()	*142*

Method Summary		Page
void	**action**()	*142*

Methods inherited from class jade.core.behaviours.CyclicBehaviour
done

Methods inherited from class jade.core.behaviours.SimpleBehaviour
reset

Methods inherited from class jade.core.behaviours.Behaviour
actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, handle, handleBlockEvent, handleRestartEvent, isRunnable, onEnd, onStart, restart, root, setAgent, setBehaviourName, setDataStore, setExecutionState

Field Detail

`private jade.lang.acl.MessageTemplate` **`template`**

Constructor Detail

`private` **`ApplicationEngineAgent.ApplicationEngineRequestServer`**`()`

Method Detail

`public void` **`action`**`()`

Overrides:

`action` in class `jade.core.behaviours.Behaviour`

Class MarCom

agents

```
java.lang.Object
  └ jade.core.Agent
      └ com.tilab.wade.commons.WadeAgentImpl
          └ com.tilab.wade.performer.WorkflowEngineAgent
              └ agents.MarCom
```

All Implemented Interfaces:

Runnable, jade.util.leap.Serializable, Serializable, jade.core.TimerListener, com.tilab.wade.commons.WadeAgent

---

`public class` **`MarCom`**
`extends com.tilab.wade.performer.WorkflowEngineAgent`

The class for the Marketing Communicator agent. A typical job description for MarCom is that he/she is responsible to assist sales and marketing management with communications media and advertising materials to effectively represent the company's products and services to customers and prospects

Author:
Pavlos Delias

Nested classes/interfaces inherited from class com.tilab.wade.performer.WorkflowEngineAgent
WorkflowEngineAgent.WorkflowExecutor

Nested classes/interfaces inherited from class jade.core.Agent
Agent.Interrupted

Field Summary		Page
private Vector<jade.core.AID>	**knownVendors**	144
private static long	**serialVersionUID**	144

Fields inherited from class com.tilab.wade.performer.WorkflowEngineAgent
ACTIVE_CNT_ATTRIBUTE, BUSY_EXECUTORS_ATTRIBUTE, codec, DEFAULT_WORKFLOW_TIMEOUT_ATTRIBUTE, DONE_STATUS, ENQUEUED_CNT_ATTRIBUTE, EXECUTING_STATUS, executors, IDLE_STATUS, onto, POOL_SIZE_ATTRIBUTE, SUSPENDED_STATUS, tbf, TERMINATING_STATUS, THREAD_CNT_ATTRIBUTE, WAITING_STATUS, WORKFLOW_CNT_ATTRIBUTE

Fields inherited from class com.tilab.wade.commons.WadeAgentImpl
arguments, myLogger

Fields inherited from class jade.core.Agent
AP_ACTIVE, AP_DELETED, AP_IDLE, AP_INITIATED, AP_MAX, AP_MIN, AP_SUSPENDED, AP_WAITING, D_ACTIVE, D_MAX, D_MIN, D_RETIRED, D_SUSPENDED, D_UNKNOWN, MSG_QUEUE_CLASS

Fields inherited from interface com.tilab.wade.commons.WadeAgent
ADMINISTRATOR_ROLE, AGENT_CLASSNAME, AGENT_LOCATION, AGENT_OWNER, AGENT_POOL, AGENT_ROLE, AGENT_TYPE, BCA_AGENT_TYPE, CONFIGURATION_AGENT_TYPE, CONTROL_AGENT_TYPE, DUMP_ARGUMENTS, HOSTADDRESS, HOSTNAME, JADE_ADDITIONAL_ARGS, JADE_PROFILE, JAVA_PROFILE, MDB_AGENT_TYPE, MESSAGE_QUEUE_SIZE_ATTRIBUTE, NONE_OWNER, NULL, RAA_AGENT_TYPE, RESTARTING, STARTUP_TIME_ATTRIBUTE, TRANSIENT_AGENT_ARGUMENT, WFENGINE_AGENT_TYPE, WORKFLOW_EXECUTOR_ROLE

Constructor Summary	Page
MarCom()	144

Method Summary		Page
protected void	**agentSpecificSetup**()	144
static com.tilab.wade.commons.AgentType	**getMyType**()	144
Vector<jade.core.AID>	**getVendors**()	144
private void	**subscribeForVendors**() subscribe to the DF to keep the list of vendors up to date Vendors are identified by their "position" property (set to "vendor")	144

Methods inherited from class com.tilab.wade.performer.WorkflowEngineAgent
adjustControlInfo, afterMove, beforeMove, createExecutionId, createGenericError, dequeue, enqueue, getActiveCnt, getBusyExecutors, getClassLoaderIdentifier, getCommitTimeout, getDefaultWorkflowTimeout, getEnqueuedCnt, getExecutionContext, getExecutorsTableStatus, getLanguage, getOntology, getPoolSize, getRollbackTimeout, getSuspendedCnt, getThreadCnt, getWorkflowClassLoader, getWorkflowCnt, handleAbortedTransaction, handleBeginActivity, handleBeginApplication, handleBeginWorkflow, handleCleanupWorkflow, handleCommittedTransaction, handleCompletedSubflow, handleDelegatedSubflow, handleEndActivity, handleEndApplication, handleEndWorkflow, handleError, handleEvent, handleFailedTransaction, handleIncomingWorkflow, handleOpenedTransaction, handleUnknownAction, isWorking, loadRollbackWorkflow, removeConversation, removeFromQueue, reply, serveExecuteWorkflow, serveGetPoolSize, serveGetSessionStatus, serveGetWRD, serveKillWorkflow, serveSetControlInfo, serveSetPoolSize, serveSetWRD, setPoolSize, takeDown

Methods inherited from class com.tilab.wade.commons.WadeAgentImpl
getAttributes, getDFDescription, getManagementResponder, getMessageQueueSize, getOwner, getRole, getStartupTime, getType, setAttributes, setup

Methods inherited from class jade.core.Agent
addBehaviour, afterClone, beforeClone, blockingReceive, blockingReceive, blockingReceive, blockingReceive, changeStateTo, clean, createMessageQueue, doActivate, doClone, doDelete, doMove, doSuspend, doTimeOut, doWait, doWait, doWake, getAgentState, getAID, getAMS, getArguments, getBootProperties, getContainerController, getContentManager, getCurQueueSize, getDefaultDF, getHap, getHelper, getLocalName, getName, getO2AObject, getProperty, getQueueSize, getState, here, isRestarting, join, notifyChangeBehaviourState, notifyRestarted, postMessage, putBack, putO2AObject, receive, receive, removeBehaviour, removeTimer, restartLater, restore, restoreBufferedState, run, send, setArguments, setEnabledO2ACommunication, setGenerateBehaviourEvents, setO2AManager, setQueueSize, waitUntilStarted, write

Field Detail

`private static final long` **`serialVersionUID`**

`private Vector<jade.core.AID>` **`knownVendors`**

Constructor Detail

`public` **`MarCom`**`()`

Method Detail

```
protected void agentSpecificSetup()
                           throws com.tilab.wade.commons.AgentInitializationException
```

Overrides:

`agentSpecificSetup` in class `com.tilab.wade.performer.WorkflowEngineAgent`

Throws:

`com.tilab.wade.commons.AgentInitializationException`

`private void` **`subscribeForVendors`**`()`

subscribe to the DF to keep the list of vendors up to date Vendors are identified by their "position" property (set to "vendor")

`public static com.tilab.wade.commons.AgentType` **`getMyType`**`()`

`public Vector<jade.core.AID>` **`getVendors`**`()`

Class MarkAssistant

[agents]

```
java.lang.Object
  └ jade.core.Agent
      └ com.tilab.wade.commons.WadeAgentImpl
          └ com.tilab.wade.performer.WorkflowEngineAgent
              └ agents.MarkAssistant
```

All Implemented Interfaces:

Runnable, jade.util.leap.Serializable, Serializable, jade.core.TimerListener, com.tilab.wade.commons.WadeAgent

```
public class MarkAssistant
extends com.tilab.wade.performer.WorkflowEngineAgent
```

The class for the Marketing Assistant agent. Typically, The Marketing Assistant provides administrative support to the staff of the Marketing Department. Duties include general research, clerical, and project based work.

Author:

Pavlos Delias

Nested classes/interfaces inherited from class com.tilab.wade.performer.WorkflowEngineAgent
`WorkflowEngineAgent.WorkflowExecutor`

Nested classes/interfaces inherited from class jade.core.Agent
`Agent.Interrupted`

Fields inherited from class com.tilab.wade.performer.WorkflowEngineAgent
`ACTIVE_CNT_ATTRIBUTE, BUSY_EXECUTORS_ATTRIBUTE, codec, DEFAULT_WORKFLOW_TIMEOUT_ATTRIBUTE, DONE_STATUS, ENQUEUED_CNT_ATTRIBUTE, EXECUTING_STATUS, executors, IDLE_STATUS, onto, POOL_SIZE_ATTRIBUTE, SUSPENDED_STATUS, tbf, TERMINATING_STATUS, THREAD_CNT_ATTRIBUTE, WAITING_STATUS, WORKFLOW_CNT_ATTRIBUTE`

Fields inherited from class com.tilab.wade.commons.WadeAgentImpl
`arguments, myLogger`

Fields inherited from class jade.core.Agent
`AP_ACTIVE, AP_DELETED, AP_IDLE, AP_INITIATED, AP_MAX, AP_MIN, AP_SUSPENDED, AP_WAITING, D_ACTIVE, D_MAX, D_MIN, D_RETIRED, D_SUSPENDED, D_UNKNOWN, MSG_QUEUE_CLASS`

Fields inherited from interface com.tilab.wade.commons.WadeAgent
`ADMINISTRATOR_ROLE, AGENT_CLASSNAME, AGENT_LOCATION, AGENT_OWNER, AGENT_POOL, AGENT_ROLE, AGENT_TYPE, BCA_AGENT_TYPE, CONFIGURATION_AGENT_TYPE, CONTROL_AGENT_TYPE, DUMP_ARGUMENTS, HOSTADDRESS, HOSTNAME, JADE_ADDITIONAL_ARGS, JADE_PROFILE, JAVA_PROFILE, MDB_AGENT_TYPE, MESSAGE_QUEUE_SIZE_ATTRIBUTE, NONE_OWNER, NULL, RAA_AGENT_TYPE, RESTARTING, STARTUP_TIME_ATTRIBUTE, TRANSIENT_AGENT_ARGUMENT, WFENGINE_AGENT_TYPE, WORKFLOW_EXECUTOR_ROLE`

Constructor Summary	*Page*
`MarkAssistant``()`	*146*

Method Summary		*Page*
`protected void`	**`agentSpecificSetup`**`()`	*146*

Methods inherited from class com.tilab.wade.performer.WorkflowEngineAgent
`adjustControlInfo, afterMove, beforeMove, createExecutionId, createGenericError, dequeue, enqueue, getActiveCnt, getBusyExecutors, getClassLoaderIdentifier, getCommitTimeout, getDefaultWorkflowTimeout, getEnqueuedCnt, getExecutionContext, getExecutorsTableStatus, getLanguage, getOntology, getPoolSize, getRollbackTimeout, getSuspendedCnt, getThreadCnt, getWorkflowClassLoader, getWorkflowCnt, handleAbortedTransaction, handleBeginActivity, handleBeginApplication, handleBeginWorkflow, handleCleanupWorkflow,`

handleCommittedTransaction, handleCompletedSubflow, handleDelegatedSubflow, handleEndActivity, handleEndApplication, handleEndWorkflow, handleError, handleEvent, handleFailedTransaction, handleIncomingWorkflow, handleOpenedTransaction, handleUnknownAction, isWorking, loadRollbackWorkflow, removeConversation, removeFromQueue, reply, serveExecuteWorkflow, serveGetPoolSize, serveGetSessionStatus, serveGetWRD, serveKillWorkflow, serveSetControlInfo, serveSetPoolSize, serveSetWRD, setPoolSize, takeDown

Methods inherited from class com.tilab.wade.commons.WadeAgentImpl
getAttributes, getDFDescription, getManagementResponder, getMessageQueueSize, getOwner, getRole, getStartupTime, getType, setAttributes, setup

Methods inherited from class jade.core.Agent
addBehaviour, afterClone, beforeClone, blockingReceive, blockingReceive, blockingReceive, blockingReceive, changeStateTo, clean, createMessageQueue, doActivate, doClone, doDelete, doMove, doSuspend, doTimeOut, doWait, doWait, doWake, getAgentState, getAID, getAMS, getArguments, getBootProperties, getContainerController, getContentManager, getCurQueueSize, getDefaultDF, getHap, getHelper, getLocalName, getName, getO2AObject, getProperty, getQueueSize, getState, here, isRestarting, join, notifyChangeBehaviourState, notifyRestarted, postMessage, putBack, putO2AObject, receive, receive, removeBehaviour, removeTimer, restartLater, restore, restoreBufferedState, run, send, setArguments, setEnabledO2ACommunication, setGenerateBehaviourEvents, setO2AManager, setQueueSize, waitUntilStarted, write

Constructor Detail

`public` **`MarkAssistant`**`()`

Method Detail

```
protected void agentSpecificSetup()
                           throws com.tilab.wade.commons.AgentInitializationException
```

Overrides:

`agentSpecificSetup` in class `com.tilab.wade.performer.WorkflowEngineAgent`

Throws:

`com.tilab.wade.commons.AgentInitializationException`

Class MarketingDirector

agents

```
java.lang.Object
  └ jade.core.Agent
      └ com.tilab.wade.commons.WadeAgentImpl
          └ com.tilab.wade.performer.WorkflowEngineAgent
              └ agents.MarketingDirector
```

All Implemented Interfaces:

Runnable, jade.util.leap.Serializable, Serializable, jade.core.TimerListener, com.tilab.wade.commons.WadeAgent

---

```
public class MarketingDirector
extends com.tilab.wade.performer.WorkflowEngineAgent
```

The class for the Marketing Director agent. Typically, the marketing director is responsible to direct firm's overall marketing and strategic planning programs.

Author:

Pavlos Delias

Nested classes/interfaces inherited from class com.tilab.wade.performer.WorkflowEngineAgent
`WorkflowEngineAgent.WorkflowExecutor`

Nested classes/interfaces inherited from class jade.core.Agent
`Agent.Interrupted`

Field Summary		*Page*
`private boolean`	**checkListUploaded**	148
`private com.tilab.wade.dispatcher.DispatchingCapabilities`	**dc**	148
`private boolean`	**meetingOrganized**	148

Fields inherited from class com.tilab.wade.performer.WorkflowEngineAgent
`ACTIVE_CNT_ATTRIBUTE, BUSY_EXECUTORS_ATTRIBUTE, codec, DEFAULT_WORKFLOW_TIMEOUT_ATTRIBUTE, DONE_STATUS, ENQUEUED_CNT_ATTRIBUTE, EXECUTING_STATUS, executors, IDLE_STATUS, onto, POOL_SIZE_ATTRIBUTE, SUSPENDED_STATUS, tbf, TERMINATING_STATUS, THREAD_CNT_ATTRIBUTE, WAITING_STATUS, WORKFLOW_CNT_ATTRIBUTE`

Fields inherited from class com.tilab.wade.commons.WadeAgentImpl
`arguments, myLogger`

Fields inherited from class jade.core.Agent
`AP_ACTIVE, AP_DELETED, AP_IDLE, AP_INITIATED, AP_MAX, AP_MIN, AP_SUSPENDED, AP_WAITING, D_ACTIVE, D_MAX, D_MIN, D_RETIRED, D_SUSPENDED, D_UNKNOWN, MSG_QUEUE_CLASS`

Fields inherited from interface com.tilab.wade.commons.WadeAgent
`ADMINISTRATOR_ROLE, AGENT_CLASSNAME, AGENT_LOCATION, AGENT_OWNER, AGENT_POOL, AGENT_ROLE, AGENT_TYPE, BCA_AGENT_TYPE, CONFIGURATION_AGENT_TYPE, CONTROL_AGENT_TYPE, DUMP_ARGUMENTS, HOSTADDRESS, HOSTNAME, JADE_ADDITIONAL_ARGS, JADE_PROFILE, JAVA_PROFILE, MDB_AGENT_TYPE, MESSAGE_QUEUE_SIZE_ATTRIBUTE, NONE_OWNER, NULL, RAA_AGENT_TYPE, RESTARTING, STARTUP_TIME_ATTRIBUTE, TRANSIENT_AGENT_ARGUMENT, WFENGINE_AGENT_TYPE, WORKFLOW_EXECUTOR_ROLE`

Constructor Summary	*Page*
MarketingDirector`()`	148

Method Summary		*Page*
`protected void`	**agentSpecificSetup**`()`	148
`boolean`	**isCheckListUploaded**`()`	148
`boolean`	**isMeetingOrganized**`()`	148
`void`	**setCheckListUploaded**`(boolean checkListUploaded)`	148
`void`	**setMeetingOrganized**`(boolean meetingOrganized)`	148

Methods inherited from class com.tilab.wade.performer.WorkflowEngineAgent
`adjustControlInfo, afterMove, beforeMove, createExecutionId, createGenericError, dequeue, enqueue, getActiveCnt, getBusyExecutors, getClassLoaderIdentifier, getCommitTimeout, getDefaultWorkflowTimeout, getEnqueuedCnt, getExecutionContext, getExecutorsTableStatus, getLanguage, getOntology, getPoolSize, getRollbackTimeout, getSuspendedCnt, getThreadCnt, getWorkflowClassLoader, getWorkflowCnt, handleAbortedTransaction, handleBeginActivity, handleBeginApplication, handleBeginWorkflow, handleCleanupWorkflow, handleCommittedTransaction, handleCompletedSubflow, handleDelegatedSubflow, handleEndActivity, handleEndApplication, handleEndWorkflow, handleError, handleEvent, handleFailedTransaction, handleIncomingWorkflow, handleOpenedTransaction,`

handleUnknownAction, isWorking, loadRollbackWorkflow, removeConversation, removeFromQueue, reply, serveExecuteWorkflow, serveGetPoolSize, serveGetSessionStatus, serveGetWRD, serveKillWorkflow, serveSetControlInfo, serveSetPoolSize, serveSetWRD, setPoolSize, takeDown

Methods inherited from class com.tilab.wade.commons.WadeAgentImpl
getAttributes, getDFDescription, getManagementResponder, getMessageQueueSize, getOwner, getRole, getStartupTime, getType, setAttributes, setup

Methods inherited from class jade.core.Agent
addBehaviour, afterClone, beforeClone, blockingReceive, blockingReceive, blockingReceive, blockingReceive, changeStateTo, clean, createMessageQueue, doActivate, doClone, doDelete, doMove, doSuspend, doTimeOut, doWait, doWait, doWake, getAgentState, getAID, getAMS, getArguments, getBootProperties, getContainerController, getContentManager, getCurQueueSize, getDefaultDF, getHap, getHelper, getLocalName, getName, getO2AObject, getProperty, getQueueSize, getState, here, isRestarting, join, notifyChangeBehaviourState, notifyRestarted, postMessage, putBack, putO2AObject, receive, receive, removeBehaviour, removeTimer, restartLater, restore, restoreBufferedState, run, send, setArguments, setEnabledO2ACommunication, setGenerateBehaviourEvents, setO2AManager, setQueueSize, waitUntilStarted, write

Field Detail

`private boolean` **`meetingOrganized`**

`private boolean` **`checkListUploaded`**

`private com.tilab.wade.dispatcher.DispatchingCapabilities` **`dc`**

Constructor Detail

`public` **`MarketingDirector`**`()`

Method Detail

```
protected void agentSpecificSetup()
                           throws com.tilab.wade.commons.AgentInitializationException
```

Overrides:

`agentSpecificSetup` in class `com.tilab.wade.performer.WorkflowEngineAgent`

Throws:

`com.tilab.wade.commons.AgentInitializationException`

`public void` **`setMeetingOrganized`**`(boolean meetingOrganized)`

`public boolean` **`isMeetingOrganized`**`()`

`public void` **`setCheckListUploaded`**`(boolean checkListUploaded)`

`public boolean` **`isCheckListUploaded`**`()`

Class MediaVendor

[agents]

```
java.lang.Object
  └ jade.core.Agent
      └ com.tilab.wade.commons.WadeAgentImpl
          └ com.tilab.wade.performer.WorkflowEngineAgent
              └ agents.MediaVendor
```

All Implemented Interfaces:

Runnable, jade.util.leap.Serializable, Serializable, jade.core.TimerListener, com.tilab.wade.commons.WadeAgent

```
public class MediaVendor
extends com.tilab.wade.performer.WorkflowEngineAgent
```

The class to represent a Vendor by an agent. A Media Vendor is considered a vendor organization to which the base organization can outsource some of its functions.

Author:
Pavlos Delias

Nested Class Summary		Page
private class	**MediaVendor.ProposalServer** Inner Class ProposalServer This class serves the incoming requests for media production	151

Nested classes/interfaces inherited from class com.tilab.wade.performer.WorkflowEngineAgent
WorkflowEngineAgent.WorkflowExecutor

Nested classes/interfaces inherited from class jade.core.Agent
Agent.Interrupted

Field Summary		Page
private jade.lang.acl.MessageTemplate	**acceptProposalTemplate**	150
private boolean	**calculated**	150
private jade.lang.acl.ACLMessage	**CFP**	150
private com.tilab.wade.dispatcher.DispatchingCapabilities	**dc**	150
private double	**myOffer**	150
private int	**myStyle**	150

Fields inherited from class com.tilab.wade.performer.WorkflowEngineAgent
ACTIVE_CNT_ATTRIBUTE, BUSY_EXECUTORS_ATTRIBUTE, codec, DEFAULT_WORKFLOW_TIMEOUT_ATTRIBUTE, DONE_STATUS, ENQUEUED_CNT_ATTRIBUTE, EXECUTING_STATUS, executors, IDLE_STATUS, onto, POOL_SIZE_ATTRIBUTE, SUSPENDED_STATUS, tbf, TERMINATING_STATUS, THREAD_CNT_ATTRIBUTE, WAITING_STATUS, WORKFLOW_CNT_ATTRIBUTE

Fields inherited from class com.tilab.wade.commons.WadeAgentImpl
arguments, myLogger

Fields inherited from class jade.core.Agent
AP_ACTIVE, AP_DELETED, AP_IDLE, AP_INITIATED, AP_MAX, AP_MIN, AP_SUSPENDED, AP_WAITING, D_ACTIVE, D_MAX, D_MIN, D_RETIRED, D_SUSPENDED, D_UNKNOWN, MSG_QUEUE_CLASS

Fields inherited from interface com.tilab.wade.commons.WadeAgent
ADMINISTRATOR_ROLE, AGENT_CLASSNAME, AGENT_LOCATION, AGENT_OWNER, AGENT_POOL, AGENT_ROLE, AGENT_TYPE, BCA_AGENT_TYPE, CONFIGURATION_AGENT_TYPE, CONTROL_AGENT_TYPE, DUMP_ARGUMENTS, HOSTADDRESS, HOSTNAME, JADE_ADDITIONAL_ARGS, JADE_PROFILE, JAVA_PROFILE, MDB_AGENT_TYPE, MESSAGE_QUEUE_SIZE_ATTRIBUTE, NONE_OWNER, NULL, RAA_AGENT_TYPE, RESTARTING, STARTUP_TIME_ATTRIBUTE, TRANSIENT_AGENT_ARGUMENT, WFENGINE_AGENT_TYPE, WORKFLOW_EXECUTOR_ROLE

Constructor Summary	Page
MediaVendor()	150

Method Summary		Page
protected void	**agentSpecificSetup**()	150
void	**calculateOfferWF**(Offer o) This method calls the execution of the VendorOffer workflow, passing an Offer argument	150

double	**getMyOffer**()	*150*
boolean	**isCalculated**()	*151*
void	**setCalculated**(boolean calculated)	*151*
void	**setMyOffer**(double myOffer)	*150*

Methods inherited from class com.tilab.wade.performer.WorkflowEngineAgent
adjustControlInfo, afterMove, beforeMove, createExecutionId, createGenericError, dequeue, enqueue, getActiveCnt, getBusyExecutors, getClassLoaderIdentifier, getCommitTimeout, getDefaultWorkflowTimeout, getEnqueuedCnt, getExecutionContext, getExecutorsTableStatus, getLanguage, getOntology, getPoolSize, getRollbackTimeout, getSuspendedCnt, getThreadCnt, getWorkflowClassLoader, getWorkflowCnt, handleAbortedTransaction, handleBeginActivity, handleBeginApplication, handleBeginWorkflow, handleCleanupWorkflow, handleCommittedTransaction, handleCompletedSubflow, handleDelegatedSubflow, handleEndActivity, handleEndApplication, handleEndWorkflow, handleError, handleEvent, handleFailedTransaction, handleIncomingWorkflow, handleOpenedTransaction, handleUnknownAction, isWorking, loadRollbackWorkflow, removeConversation, removeFromQueue, reply, serveExecuteWorkflow, serveGetPoolSize, serveGetSessionStatus, serveGetWRD, serveKillWorkflow, serveSetControlInfo, serveSetPoolSize, serveSetWRD, setPoolSize, takeDown

Methods inherited from class com.tilab.wade.commons.WadeAgentImpl
getAttributes, getDFDescription, getManagementResponder, getMessageQueueSize, getOwner, getRole, getStartupTime, getType, setAttributes, setup

Methods inherited from class jade.core.Agent
addBehaviour, afterClone, beforeClone, blockingReceive, blockingReceive, blockingReceive, blockingReceive, changeStateTo, clean, createMessageQueue, doActivate, doClone, doDelete, doMove, doSuspend, doTimeOut, doWait, doWait, doWake, getAgentState, getAID, getAMS, getArguments, getBootProperties, getContainerController, getContentManager, getCurQueueSize, getDefaultDF, getHap, getHelper, getLocalName, getName, getO2AObject, getProperty, getQueueSize, getState, here, isRestarting, join, notifyChangeBehaviourState, notifyRestarted, postMessage, putBack, putO2AObject, receive, receive, removeBehaviour, removeTimer, restartLater, restore, restoreBufferedState, run, send, setArguments, setEnabledO2ACommunication, setGenerateBehaviourEvents, setO2AManager, setQueueSize, waitUntilStarted, write

Field Detail

private int **myStyle**

private double **myOffer**

private boolean **calculated**

private com.tilab.wade.dispatcher.DispatchingCapabilities **dc**

private jade.lang.acl.ACLMessage **CFP**

private jade.lang.acl.MessageTemplate **acceptProposalTemplate**

Constructor Detail

public **MediaVendor**()

Method Detail

protected void **agentSpecificSetup**()
throws com.tilab.wade.commons.AgentInitializationException

Overrides:

agentSpecificSetup in class com.tilab.wade.performer.WorkflowEngineAgent

Throws:

com.tilab.wade.commons.AgentInitializationException

public void **calculateOfferWF**(Offer o)

This method calls the execution of the VendorOffer workflow, passing an Offer argument

public void **setMyOffer**(double myOffer)

public double **getMyOffer**()

```
public void setCalculated(boolean calculated)
```

```
public boolean isCalculated()
```

Class MediaVendor.ProposalServer

agents

```
java.lang.Object
  └ jade.core.behaviours.Behaviour
      └ jade.core.behaviours.SimpleBehaviour
          └ jade.core.behaviours.CyclicBehaviour
              └ agents.MediaVendor.ProposalServer
```

All Implemented Interfaces:

jade.util.leap.Serializable, Serializable

Enclosing class:

MediaVendor

```
private class MediaVendor.ProposalServer
extends jade.core.behaviours.CyclicBehaviour
```

Inner Class ProposalServer This class serves the incoming requests for media production

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
Behaviour.RunnableChangedEvent

Field Summary		Page
MediaVendor	**MV**	152

Fields inherited from class jade.core.behaviours.Behaviour
myAgent, myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING

Constructor Summary	Page
MediaVendor.ProposalServer()	152

Method Summary		Page
void	**action**()	152
private MediaDecisionsGUI.MediaFormat	**getFormat**(String f)	152
private void	**serveAcceptProposal**(jade.lang.acl.ACLMessage msg)	152
private void	**serveCFP**(jade.lang.acl.ACLMessage msg)	152

Methods inherited from class jade.core.behaviours.CyclicBehaviour
done

Methods inherited from class jade.core.behaviours.SimpleBehaviour
reset

Methods inherited from class jade.core.behaviours.Behaviour
actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, handle, handleBlockEvent, handleRestartEvent, isRunnable, onEnd, onStart,

`restart, root, setAgent, setBehaviourName, setDataStore, setExecutionState`

Field Detail

`MediaVendor` **`MV`**

Constructor Detail

`public` **`MediaVendor.ProposalServer`**`()`

Method Detail

`public void` **`action`**`()`

Overrides:

`action` in class `jade.core.behaviours.Behaviour`

`private MediaDecisionsGUI.MediaFormat` **`getFormat`**`(String f)`

`private void` **`serveCFP`**`(jade.lang.acl.ACLMessage msg)`

`private void` **`serveAcceptProposal`**`(jade.lang.acl.ACLMessage msg)`

Class ProductManager

agents

```
java.lang.Object
  └ jade.core.Agent
      └ com.tilab.wade.commons.WadeAgentImpl
          └ com.tilab.wade.performer.WorkflowEngineAgent
              └ agents.ProductManager
```

All Implemented Interfaces:

Runnable, jade.util.leap.Serializable, Serializable, jade.core.TimerListener, com.tilab.wade.commons.WadeAgent

---

```
public class ProductManager
extends com.tilab.wade.performer.WorkflowEngineAgent
```

The class for the Product Manager Agent. Typically, The Product Manager is responsible for the product planning and execution throughout the product lifecycle, including: gathering and prioritizing product and customer requirements, defining the product vision, and working closely with engineering, sales, marketing and support to ensure revenue and customer satisfaction goals are met. The Product Manager's job also includes ensuring that the product supports the company's overall strategy and goals.

Author:

Pavlos Delias

Nested classes/interfaces inherited from class com.tilab.wade.performer.WorkflowEngineAgent
`WorkflowEngineAgent.WorkflowExecutor`

Nested classes/interfaces inherited from class jade.core.Agent
`Agent.Interrupted`

Field Summary		Page
`private Vector<jade.core.AID>`	**`assistants`**	*154*
`private jade.lang.acl.MessageTemplate`	**`meeting_template`**	*154*

Fields inherited from class com.tilab.wade.performer.WorkflowEngineAgent
`ACTIVE_CNT_ATTRIBUTE, BUSY_EXECUTORS_ATTRIBUTE, codec, DEFAULT_WORKFLOW_TIMEOUT_ATTRIBUTE,`

DONE_STATUS, ENQUEUED_CNT_ATTRIBUTE, EXECUTING_STATUS, executors, IDLE_STATUS, onto, POOL_SIZE_ATTRIBUTE, SUSPENDED_STATUS, tbf, TERMINATING_STATUS, THREAD_CNT_ATTRIBUTE, WAITING_STATUS, WORKFLOW_CNT_ATTRIBUTE

Fields inherited from class com.tilab.wade.commons.WadeAgentImpl
arguments, myLogger

Fields inherited from class jade.core.Agent
AP_ACTIVE, AP_DELETED, AP_IDLE, AP_INITIATED, AP_MAX, AP_MIN, AP_SUSPENDED, AP_WAITING, D_ACTIVE, D_MAX, D_MIN, D_RETIRED, D_SUSPENDED, D_UNKNOWN, MSG_QUEUE_CLASS

Fields inherited from interface com.tilab.wade.commons.WadeAgent
ADMINISTRATOR_ROLE, AGENT_CLASSNAME, AGENT_LOCATION, AGENT_OWNER, AGENT_POOL, AGENT_ROLE, AGENT_TYPE, BCA_AGENT_TYPE, CONFIGURATION_AGENT_TYPE, CONTROL_AGENT_TYPE, DUMP_ARGUMENTS, HOSTADDRESS, HOSTNAME, JADE_ADDITIONAL_ARGS, JADE_PROFILE, JAVA_PROFILE, MDB_AGENT_TYPE, MESSAGE_QUEUE_SIZE_ATTRIBUTE, NONE_OWNER, NULL, RAA_AGENT_TYPE, RESTARTING, STARTUP_TIME_ATTRIBUTE, TRANSIENT_AGENT_ARGUMENT, WFENGINE_AGENT_TYPE, WORKFLOW_EXECUTOR_ROLE

Constructor Summary	Page
ProductManager()	154

Method Summary		Page
protected void	**agentSpecificSetup**()	154
Vector<jade.core.AID>	**getAssistants**()	154
static com.tilab.wade.commons.AgentType	**getMyType**()	154
private void	**proposeResponderAction**() This method adds a cyclic behaviour to check if there are any meeting proposals arrived, and if any properly respond to them.	154
private void	**subscribeForAssistants**() subscribe to the DF to keep the list of Marketing Assistants up to date Assistants are identified by their "position" property (set to "assistant")	154

Methods inherited from class com.tilab.wade.performer.WorkflowEngineAgent
adjustControlInfo, afterMove, beforeMove, createExecutionId, createGenericError, dequeue, enqueue, getActiveCnt, getBusyExecutors, getClassLoaderIdentifier, getCommitTimeout, getDefaultWorkflowTimeout, getEnqueuedCnt, getExecutionContext, getExecutorsTableStatus, getLanguage, getOntology, getPoolSize, getRollbackTimeout, getSuspendedCnt, getThreadCnt, getWorkflowClassLoader, getWorkflowCnt, handleAbortedTransaction, handleBeginActivity, handleBeginApplication, handleBeginWorkflow, handleCleanupWorkflow, handleCommittedTransaction, handleCompletedSubflow, handleDelegatedSubflow, handleEndActivity, handleEndApplication, handleEndWorkflow, handleError, handleEvent, handleFailedTransaction, handleIncomingWorkflow, handleOpenedTransaction, handleUnknownAction, isWorking, loadRollbackWorkflow, removeConversation, removeFromQueue, reply, serveExecuteWorkflow, serveGetPoolSize, serveGetSessionStatus, serveGetWRD, serveKillWorkflow, serveSetControlInfo, serveSetPoolSize, serveSetWRD, setPoolSize, takeDown

Methods inherited from class com.tilab.wade.commons.WadeAgentImpl
getAttributes, getDFDescription, getManagementResponder, getMessageQueueSize, getOwner, getRole, getStartupTime, getType, setAttributes, setup

Methods inherited from class jade.core.Agent
addBehaviour, afterClone, beforeClone, blockingReceive, blockingReceive, blockingReceive, blockingReceive, changeStateTo, clean, createMessageQueue, doActivate, doClone, doDelete, doMove, doSuspend, doTimeOut, doWait, doWait, doWake, getAgentState, getAID, getAMS, getArguments, getBootProperties, getContainerController, getContentManager, getCurQueueSize,

```
getDefaultDF, getHap, getHelper, getLocalName, getName, getO2AObject, getProperty,
getQueueSize, getState, here, isRestarting, join, notifyChangeBehaviourState,
notifyRestarted, postMessage, putBack, putO2AObject, receive, receive, removeBehaviour,
removeTimer, restartLater, restore, restoreBufferedState, run, send, setArguments,
setEnabledO2ACommunication, setGenerateBehaviourEvents, setO2AManager, setQueueSize,
waitUntilStarted, write
```

Field Detail

`private Vector<jade.core.AID>` **`assistants`**

`private jade.lang.acl.MessageTemplate` **`meeting_template`**

Constructor Detail

`public` **`ProductManager`**`()`

Method Detail

`protected void` **`agentSpecificSetup`**`()`
`throws com.tilab.wade.commons.AgentInitializationException`

Overrides:

`agentSpecificSetup` in class `com.tilab.wade.performer.WorkflowEngineAgent`

Throws:

`com.tilab.wade.commons.AgentInitializationException`

`private void` **`subscribeForAssistants`**`()`

subscribe to the DF to keep the list of Marketing Assistants up to date Assistants are identified by their "position" property (set to "assistant")

`public Vector<jade.core.AID>` **`getAssistants`**`()`

`private void` **`proposeResponderAction`**`()`

This method adds a cyclic behaviour to check if there are any meeting proposals arrived, and if any properly respond to them.

`public static com.tilab.wade.commons.AgentType` **`getMyType`**`()`

Package agents.contactCenter

Class Summary		Page
AssignmentAgent	This agent is responsible for assigning jobs to employees.	155
AssignmentAgent.ContactRequestServer	A cyclic behavior that helps the AssignmentAgent to listen for requests related to the Contact Center ontology.	158
AssignmentAgent.Task	A supporting class to map an e-mail to a Task object	159
ContactAgent	The class to represent the employee of the Contact Center.	160
ContactAgent.ContactRequestServer	A Cyclic Behavior to support the agent to listen to requests related to the contact center ontology.	162

Class AssignmentAgent

agents.contactCenter

```
java.lang.Object
  └ jade.core.Agent
      └ com.tilab.wade.commons.WadeAgentImpl
          └ com.tilab.wade.performer.WorkflowEngineAgent
              └ agents.contactCenter.AssignmentAgent
```

All Implemented Interfaces:

Runnable, jade.util.leap.Serializable, Serializable, jade.core.TimerListener, com.tilab.wade.commons.WadeAgent

```
public class AssignmentAgent
extends com.tilab.wade.performer.WorkflowEngineAgent
```

This agent is responsible for assigning jobs to employees.

Author:

Pavlos Delias

Nested Class Summary		Page
private class	**AssignmentAgent.ContactRequestServer** A cyclic behavior that helps the AssignmentAgent to listen for requests related to the Contact Center ontology.	158
class	**AssignmentAgent.Task** A supporting class to map an e-mail to a Task object	159

Nested classes/interfaces inherited from class com.tilab.wade.performer.WorkflowEngineAgent
WorkflowEngineAgent.WorkflowExecutor

Nested classes/interfaces inherited from class jade.core.Agent
Agent.Interrupted

Field Summary		Page
private Vector<Integer>	**deadlines**	157
private Vector<Integer>	**durations**	157

private Vector<Mail>	**mails**	157
private Vector<String>	**names**	157
private Vector<Integer>	**releaseTimes**	157
private Vector<jade.core.AID>	**taskAgents**	157

Fields inherited from class com.tilab.wade.performer.WorkflowEngineAgent
ACTIVE_CNT_ATTRIBUTE, BUSY_EXECUTORS_ATTRIBUTE, codec, DEFAULT_WORKFLOW_TIMEOUT_ATTRIBUTE, DONE_STATUS, ENQUEUED_CNT_ATTRIBUTE, EXECUTING_STATUS, executors, IDLE_STATUS, onto, POOL_SIZE_ATTRIBUTE, SUSPENDED_STATUS, tbf, TERMINATING_STATUS, THREAD_CNT_ATTRIBUTE, WAITING_STATUS, WORKFLOW_CNT_ATTRIBUTE

Fields inherited from class com.tilab.wade.commons.WadeAgentImpl
arguments, myLogger

Fields inherited from class jade.core.Agent
AP_ACTIVE, AP_DELETED, AP_IDLE, AP_INITIATED, AP_MAX, AP_MIN, AP_SUSPENDED, AP_WAITING, D_ACTIVE, D_MAX, D_MIN, D_RETIRED, D_SUSPENDED, D_UNKNOWN, MSG_QUEUE_CLASS

Fields inherited from interface com.tilab.wade.commons.WadeAgent
ADMINISTRATOR_ROLE, AGENT_CLASSNAME, AGENT_LOCATION, AGENT_OWNER, AGENT_POOL, AGENT_ROLE, AGENT_TYPE, BCA_AGENT_TYPE, CONFIGURATION_AGENT_TYPE, CONTROL_AGENT_TYPE, DUMP_ARGUMENTS, HOSTADDRESS, HOSTNAME, JADE_ADDITIONAL_ARGS, JADE_PROFILE, JAVA_PROFILE, MDB_AGENT_TYPE, MESSAGE_QUEUE_SIZE_ATTRIBUTE, NONE_OWNER, NULL, RAA_AGENT_TYPE, RESTARTING, STARTUP_TIME_ATTRIBUTE, TRANSIENT_AGENT_ARGUMENT, WFENGINE_AGENT_TYPE, WORKFLOW_EXECUTOR_ROLE

Constructor Summary	Page
AssignmentAgent()	157

Method Summary		Page
protected void	**agentSpecificSetup**()	157
private void	**createBatchfromXL**(String filename) Takes as input an Excel file and creates a batch of Mails	157
private void	**createInput4WF**() Supporting method to prepare the parameters of the workflow	157
private jade.lang.acl.ACLMessage	**prepareExecuteWorkflowRequest**(com.tilab.wade.performer.descriptors.WorkflowDescriptor wd) Prepares a message that requests execution of a workflow according to the WorkflowDescriptor, provided as input parameter	158
private com.tilab.wade.performer.descriptors.WorkflowDescriptor	**prepareWorkflowDescriptor**() Prepares a SpectralScheduling workflow by filling its parameters	157
void	**serveRead**(Read action, jade.content.onto.basic.Action actExpr, jade.lang.acl.ACLMessage msg) Serves the Read action of the Contact Center ontology.	157
private void	**subscribeForTaskAgents**() subscribe to the DF to keep the list of Contact Agents up to date Agents are identified by their "position" property (set to "employee")	158

Methods inherited from class com.tilab.wade.performer.WorkflowEngineAgent
adjustControlInfo, afterMove, beforeMove, createExecutionId, createGenericError, dequeue,

```
enqueue, getActiveCnt, getBusyExecutors, getClassLoaderIdentifier, getCommitTimeout,
getDefaultWorkflowTimeout, getEnqueuedCnt, getExecutionContext, getExecutorsTableStatus,
getLanguage, getOntology, getPoolSize, getRollbackTimeout, getSuspendedCnt, getThreadCnt,
getWorkflowClassLoader, getWorkflowCnt, handleAbortedTransaction, handleBeginActivity,
handleBeginApplication, handleBeginWorkflow, handleCleanupWorkflow,
handleCommittedTransaction, handleCompletedSubflow, handleDelegatedSubflow,
handleEndActivity, handleEndApplication, handleEndWorkflow, handleError, handleEvent,
handleFailedTransaction, handleIncomingWorkflow, handleOpenedTransaction,
handleUnknownAction, isWorking, loadRollbackWorkflow, removeConversation, removeFromQueue,
reply, serveExecuteWorkflow, serveGetPoolSize, serveGetSessionStatus, serveGetWRD,
serveKillWorkflow, serveSetControlInfo, serveSetPoolSize, serveSetWRD, setPoolSize, takeDown
```

Methods inherited from class com.tilab.wade.commons.WadeAgentImpl
getAttributes, getDFDescription, getManagementResponder, getMessageQueueSize, getOwner, getRole, getStartupTime, getType, setAttributes, setup

Methods inherited from class jade.core.Agent
addBehaviour, afterClone, beforeClone, blockingReceive, blockingReceive, blockingReceive, blockingReceive, changeStateTo, clean, createMessageQueue, doActivate, doClone, doDelete, doMove, doSuspend, doTimeOut, doWait, doWait, doWake, getAgentState, getAID, getAMS, getArguments, getBootProperties, getContainerController, getContentManager, getCurQueueSize, getDefaultDF, getHap, getHelper, getLocalName, getName, getO2AObject, getProperty, getQueueSize, getState, here, isRestarting, join, notifyChangeBehaviourState, notifyRestarted, postMessage, putBack, putO2AObject, receive, receive, removeBehaviour, removeTimer, restartLater, restore, restoreBufferedState, run, send, setArguments, setEnabledO2ACommunication, setGenerateBehaviourEvents, setO2AManager, setQueueSize, waitUntilStarted, write

Field Detail

`private Vector<Mail> mails`

`private Vector<String> names`

`private Vector<Integer> releaseTimes`

`private Vector<Integer> deadlines`

`private Vector<Integer> durations`

`private Vector<jade.core.AID> taskAgents`

Constructor Detail

`public AssignmentAgent()`

Method Detail

```
protected void agentSpecificSetup()
                           throws com.tilab.wade.commons.AgentInitializationException
```

Overrides:

`agentSpecificSetup` in class `com.tilab.wade.performer.WorkflowEngineAgent`

Throws:

`com.tilab.wade.commons.AgentInitializationException`

```
public void serveRead(Read action,
                      jade.content.onto.basic.Action actExpr,
                      jade.lang.acl.ACLMessage msg)
```

Serves the `Read` action of the Contact Center ontology. Ultimately, it reads the file specified in the input message and invokes a workflow execution though the `prepareWorkflowDescriptor()` and the `prepareExecuteWorkflowRequest(WorkflowDescriptor)` methods.

`private void createInput4WF()`

Supporting method to prepare the parameters of the workflow

`private void createBatchfromXL(String filename)`

Takes as input an Excel file and creates a batch of Mails

`private com.tilab.wade.performer.descriptors.WorkflowDescriptor prepareWorkflowDescriptor()`

Prepares a `SpectralScheduling` workflow by filling its parameters

Returns:
a `com.tilab.wade.performer.descriptors.WorkflowDescriptor` object

`private jade.lang.acl.ACLMessage` **`prepareExecuteWorkflowRequest`**`(com.tilab.wade.performer.descriptors.WorkflowDescriptor wd)`

Prepares a message that requests execution of a workflow according to the WorkflowDescriptor, provided as input parameter

Returns:
a request message

`private void` **`subscribeForTaskAgents`**`()`

subscribe to the DF to keep the list of Contact Agents up to date Agents are identified by their "position" property (set to "employee")

Class AssignmentAgent.ContactRequestServer

agents.contactCenter

```
java.lang.Object
  └ jade.core.behaviours.Behaviour
      └ jade.core.behaviours.SimpleBehaviour
          └ jade.core.behaviours.CyclicBehaviour
              └ agents.contactCenter.AssignmentAgent.ContactRequestServer
```

All Implemented Interfaces:
jade.util.leap.Serializable, Serializable

Enclosing class:
AssignmentAgent

`private class` **`AssignmentAgent.ContactRequestServer`**
`extends jade.core.behaviours.CyclicBehaviour`

A cyclic behavior that helps the AssignmentAgent to listen for requests related to the Contact Center ontology. If there are any requests received, the agent decodes the message and serves the request bu calling the appropriate methods.

Author:
Pavlos Delias

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
`Behaviour.RunnableChangedEvent`

Field Summary		Page
`private jade.lang.acl.MessageTemplate`	**template**	159

Fields inherited from class jade.core.behaviours.Behaviour
`myAgent, myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING`

Constructor Summary		Page
private	AssignmentAgent.ContactRequestServer()	159

Method Summary		Page
void	action()	159

Methods inherited from class jade.core.behaviours.CyclicBehaviour
done

Methods inherited from class jade.core.behaviours.SimpleBehaviour
reset

Methods inherited from class jade.core.behaviours.Behaviour
actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, handle, handleBlockEvent, handleRestartEvent, isRunnable, onEnd, onStart, restart, root, setAgent, setBehaviourName, setDataStore, setExecutionState

Field Detail

`private jade.lang.acl.MessageTemplate` **template**

Constructor Detail

`private` **AssignmentAgent.ContactRequestServer**`()`

Method Detail

`public void` **action**`()`

Overrides:

`action` in class `jade.core.behaviours.Behaviour`

Class AssignmentAgent.Task

agents.contactCenter

```
java.lang.Object
  └ agents.contactCenter.AssignmentAgent.Task
```

Enclosing class:

AssignmentAgent

`class` **AssignmentAgent.Task**
`extends Object`

A supporting class to map an e-mail to a Task object

Author:

Pavlos Delias

Field Summary		Page
private int	deadline	160
private String	name	160
private int	processingTime	160

private int	**releaseTime**	160

Constructor Summary	Page
AssignmentAgent.Task(Mail m, int cnt)	160

Method Summary		Page
int	**getDeadline**()	160
String	**getName**()	160
int	**getProcessingTime**()	160
int	**getReleaseTime**()	160
void	**setDeadline**(int deadline)	160
void	**setName**(String name)	160
void	**setProcessingTime**(int processingTime)	160
void	**setReleaseTime**(int releaseTime)	160

Field Detail

```
private String name
private int processingTime
private int releaseTime
private int deadline
```

Constructor Detail

```
public AssignmentAgent.Task(Mail m,
                            int cnt)
```

Method Detail

```
public void setName(String name)
public String getName()
public void setProcessingTime(int processingTime)
public int getProcessingTime()
public void setReleaseTime(int releaseTime)
public int getReleaseTime()
public void setDeadline(int deadline)
public int getDeadline()
```

Class ContactAgent

agents.contactCenter

```
java.lang.Object
  └ jade.core.Agent
      └ com.tilab.wade.commons.WadeAgentImpl
          └ com.tilab.wade.performer.WorkflowEngineAgent
              └ agents.contactCenter.ContactAgent
```

All Implemented Interfaces:

Runnable, jade.util.leap.Serializable, Serializable, jade.core.TimerListener, com.tilab.wade.commons.WadeAgent

```
public class ContactAgent
extends com.tilab.wade.performer.WorkflowEngineAgent
```

The class to represent the employee of the Contact Center.

Author:

Pavlos Delias

Nested Class Summary		Page
private class	**ContactAgent.ContactRequestServer** A Cyclic Behavior to support the agent to listen to requests related to the contact center ontology.	162

Nested classes/interfaces inherited from class com.tilab.wade.performer.WorkflowEngineAgent
WorkflowEngineAgent.WorkflowExecutor

Nested classes/interfaces inherited from class jade.core.Agent
Agent.Interrupted

Fields inherited from class com.tilab.wade.performer.WorkflowEngineAgent
ACTIVE_CNT_ATTRIBUTE, BUSY_EXECUTORS_ATTRIBUTE, codec, DEFAULT_WORKFLOW_TIMEOUT_ATTRIBUTE, DONE_STATUS, ENQUEUED_CNT_ATTRIBUTE, EXECUTING_STATUS, executors, IDLE_STATUS, onto, POOL_SIZE_ATTRIBUTE, SUSPENDED_STATUS, tbf, TERMINATING_STATUS, THREAD_CNT_ATTRIBUTE, WAITING_STATUS, WORKFLOW_CNT_ATTRIBUTE

Fields inherited from class com.tilab.wade.commons.WadeAgentImpl
arguments, myLogger

Fields inherited from class jade.core.Agent
AP_ACTIVE, AP_DELETED, AP_IDLE, AP_INITIATED, AP_MAX, AP_MIN, AP_SUSPENDED, AP_WAITING, D_ACTIVE, D_MAX, D_MIN, D_RETIRED, D_SUSPENDED, D_UNKNOWN, MSG_QUEUE_CLASS

Fields inherited from interface com.tilab.wade.commons.WadeAgent
ADMINISTRATOR_ROLE, AGENT_CLASSNAME, AGENT_LOCATION, AGENT_OWNER, AGENT_POOL, AGENT_ROLE, AGENT_TYPE, BCA_AGENT_TYPE, CONFIGURATION_AGENT_TYPE, CONTROL_AGENT_TYPE, DUMP_ARGUMENTS, HOSTADDRESS, HOSTNAME, JADE_ADDITIONAL_ARGS, JADE_PROFILE, JAVA_PROFILE, MDB_AGENT_TYPE, MESSAGE_QUEUE_SIZE_ATTRIBUTE, NONE_OWNER, NULL, RAA_AGENT_TYPE, RESTARTING, STARTUP_TIME_ATTRIBUTE, TRANSIENT_AGENT_ARGUMENT, WFENGINE_AGENT_TYPE, WORKFLOW_EXECUTOR_ROLE

Constructor Summary	Page
ContactAgent()	162

Method Summary		Page
protected void	**agentSpecificSetup**()	162
void	**serveTodo**(Todo action, jade.content.onto.basic.Action actExpr, jade.lang.acl.ACLMessage msg) Serves the Todo action of the Contact Center ontology.	162

Methods inherited from class com.tilab.wade.performer.WorkflowEngineAgent
adjustControlInfo, afterMove, beforeMove, createExecutionId, createGenericError, dequeue, enqueue, getActiveCnt, getBusyExecutors, getClassLoaderIdentifier, getCommitTimeout, getDefaultWorkflowTimeout, getEnqueuedCnt, getExecutionContext, getExecutorsTableStatus, getLanguage, getOntology, getPoolSize, getRollbackTimeout, getSuspendedCnt, getThreadCnt, getWorkflowClassLoader, getWorkflowCnt, handleAbortedTransaction, handleBeginActivity, handleBeginApplication, handleBeginWorkflow, handleCleanupWorkflow, handleCommittedTransaction, handleCompletedSubflow, handleDelegatedSubflow, handleEndActivity, handleEndApplication, handleEndWorkflow, handleError, handleEvent, handleFailedTransaction, handleIncomingWorkflow, handleOpenedTransaction, handleUnknownAction, isWorking, loadRollbackWorkflow, removeConversation, removeFromQueue, reply, serveExecuteWorkflow, serveGetPoolSize, serveGetSessionStatus, serveGetWRD,

serveKillWorkflow, serveSetControlInfo, serveSetPoolSize, serveSetWRD, setPoolSize, takeDown

Methods inherited from class com.tilab.wade.commons.WadeAgentImpl
getAttributes, getDFDescription, getManagementResponder, getMessageQueueSize, getOwner, getRole, getStartupTime, getType, setAttributes, setup

Methods inherited from class jade.core.Agent
addBehaviour, afterClone, beforeClone, blockingReceive, blockingReceive, blockingReceive, blockingReceive, changeStateTo, clean, createMessageQueue, doActivate, doClone, doDelete, doMove, doSuspend, doTimeOut, doWait, doWait, doWake, getAgentState, getAID, getAMS, getArguments, getBootProperties, getContainerController, getContentManager, getCurQueueSize, getDefaultDF, getHap, getHelper, getLocalName, getName, getO2AObject, getProperty, getQueueSize, getState, here, isRestarting, join, notifyChangeBehaviourState, notifyRestarted, postMessage, putBack, putO2AObject, receive, receive, removeBehaviour, removeTimer, restartLater, restore, restoreBufferedState, run, send, setArguments, setEnabledO2ACommunication, setGenerateBehaviourEvents, setO2AManager, setQueueSize, waitUntilStarted, write

Constructor Detail

`public` **`ContactAgent`**`()`

Method Detail

```
protected void agentSpecificSetup()
                           throws com.tilab.wade.commons.AgentInitializationException
```

Overrides:

`agentSpecificSetup` in class `com.tilab.wade.performer.WorkflowEngineAgent`

Throws:

`com.tilab.wade.commons.AgentInitializationException`

```
public void serveTodo(Todo action,
                      jade.content.onto.basic.Action actExpr,
                      jade.lang.acl.ACLMessage msg)
```

Serves the `Todo` action of the Contact Center ontology.

Class ContactAgent.ContactRequestServer

agents.contactCenter

```
java.lang.Object
  └ jade.core.behaviours.Behaviour
      └ jade.core.behaviours.SimpleBehaviour
          └ jade.core.behaviours.CyclicBehaviour
              └ agents.contactCenter.ContactAgent.ContactRequestServer
```

All Implemented Interfaces:

jade.util.leap.Serializable, Serializable

Enclosing class:

ContactAgent

```
private class ContactAgent.ContactRequestServer
extends jade.core.behaviours.CyclicBehaviour
```

A Cyclic Behavior to support the agent to listen to requests related to the contact center ontology. If any, the agent properly decodes the message and serves the corresponding action

Author:

Pavlos Delias

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
Behaviour.RunnableChangedEvent

Field Summary		Page
private jade.lang.acl.MessageTemplate	template	163

Fields inherited from class jade.core.behaviours.Behaviour
myAgent, myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING

Constructor Summary		Page
private	ContactAgent.ContactRequestServer()	163

Method Summary		Page
void	action()	163

Methods inherited from class jade.core.behaviours.CyclicBehaviour
done

Methods inherited from class jade.core.behaviours.SimpleBehaviour
reset

Methods inherited from class jade.core.behaviours.Behaviour
actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, handle, handleBlockEvent, handleRestartEvent, isRunnable, onEnd, onStart, restart, root, setAgent, setBehaviourName, setDataStore, setExecutionState

Field Detail

private jade.lang.acl.MessageTemplate **template**

Constructor Detail

private **ContactAgent.ContactRequestServer**()

Method Detail

public void **action**()

Overrides:

action in class jade.core.behaviours.Behaviour

Package applications.contactCenter

Class Summary		Page
FindTasksPerAgent	This application uses the optimistic single-threaded execution strategy to reassure that an AddWorklist request is send to the Application engine when at least an assignment exists, and after all assignments are decided.	164
SpectralSchedulingByMATLAB	Calls MATLAB to execute a specific scheduling algorithm.	165

Class FindTasksPerAgent

applications.contactCenter

```
java.lang.Object
  └ com.tilab.wade.performer.Application
      └ com.tilab.wade.performer.BaseApplication
          └ applications.contactCenter.FindTasksPerAgent
```

```
public class FindTasksPerAgent
extends com.tilab.wade.performer.BaseApplication
```

This application uses the optimistic single-threaded execution strategy to reassure that an AddWorklist request is send to the Application engine when at least an assignment exists, and after all assignments are decided.

Author:

Pavlos Delias

Field Summary		Page
HashMap<String,Integer>	assignments	165

Fields inherited from class com.tilab.wade.performer.Application
formalParams, myAgent, myExecutionId, myLogger, myName, mySessionId

Constructor Summary	Page
FindTasksPerAgent()	165

Method Summary		Page
private void	createRequest(jade.core.AID engine) Send an AddWorklist request to the engine agent, specified in the arguments The method is synchronized to reassure that the assignments Map is ready.	165
private Worklist	createWorklistFromMap(HashMap<String,Integer> map) A Worklist is created through the assignments Map.	165
void	execute()	165
private void	fillAssignments() The assignments are filtered and grouped by agent.	165
private jade.core.AID	getApplicationEngine() Supporting method that talks to the DF to retrieve the Application Engine Agent	165

Methods inherited from class com.tilab.wade.performer.BaseApplication
checkParameters, extract, fill, fillFormalParameters, getDataStore, setDataStore

Methods inherited from class com.tilab.wade.performer.Application
`commit, fill, fill, fill, fill, fill, fireEvent, get, getControlInfo, getFormalParameters, getModifier, getModifiers, getTracer, getTransactionManager, getValid, getWorkflowDescriptor, getWorkflowFailureReason, getWorkflowLastErrorEvent, isTransactional, rollback, set, setWorkflowFailureReason, trace, trace`

Field Detail

`HashMap<String,Integer>` **`assignments`**

Constructor Detail

`public` **`FindTasksPerAgent`**`()`

Method Detail

`public void` **`execute`**`()`
`throws Throwable`

Overrides:

`execute` in class `com.tilab.wade.performer.Application`

Throws:

`Throwable`

`private synchronized void` **`fillAssignments`**`()`

The assignments are filtered and grouped by agent. The map `assignments` is filled for the specified agent

---

`private synchronized void` **`createRequest`**`(jade.core.AID engine)`

Send an `AddWorklist` request to the engine agent, specified in the arguments The method is synchronized to reassure that the `assignments` Map is ready.

`private jade.core.AID` **`getApplicationEngine`**`()`

Supporting method that talks to the DF to retrieve the Application Engine Agent

Returns:

`jade.core.AID` engine

`private Worklist` **`createWorklistFromMap`**`(HashMap<String,Integer> map)`

A `Worklist` is created through the `assignments` Map.

Returns:

`Worklist` worklist

Class SpectralSchedulingByMATLAB

applications.contactCenter

```
java.lang.Object
  └ com.tilab.wade.performer.Application
      └ com.tilab.wade.performer.BaseApplication
          └ applications.contactCenter.SpectralSchedulingByMATLAB
```

`public class` **`SpectralSchedulingByMATLAB`**
`extends com.tilab.wade.performer.BaseApplication`

Calls MATLAB to execute a specific scheduling algorithm. The data input are provided as FormalParameter by the workflow class (`SpectralScheduling`) that calls this application

Author:

Pavlos Delias

Field Summary		Page
private static Object	**lock**	166

Fields inherited from class com.tilab.wade.performer.Application
formalParams, myAgent, myExecutionId, myLogger, myName, mySessionId

Constructor Summary	Page
SpectralSchedulingByMATLAB()	166

Method Summary		Page
private String	**createNamesStringFromVector**(Vector<String> n) Supportive method to create a String from a Vector.	166
private String	**createTimesStringFromVector**(Vector<Integer> times) Supportive method to create a String from a Vector.	166
void	**execute**()	166
private void	**save**(BufferedImage image, String ext)	167
private BufferedImage	**toBufferedImage**(Image src)	167

Methods inherited from class com.tilab.wade.performer.BaseApplication
checkParameters, extract, fill, fillFormalParameters, getDataStore, setDataStore

Methods inherited from class com.tilab.wade.performer.Application
commit, fill, fill, fill, fill, fill, fireEvent, get, getControlInfo, getFormalParameters, getModifier, getModifiers, getTracer, getTransactionManager, getValid, getWorkflowDescriptor, getWorkflowFailureReason, getWorkflowLastErrorEvent, isTransactional, rollback, set, setWorkflowFailureReason, trace, trace

Field Detail

private static Object **lock**

Constructor Detail

public **SpectralSchedulingByMATLAB**()

Method Detail

public void **execute**()
throws Throwable

Overrides:

execute in class com.tilab.wade.performer.Application

Throws:

Throwable

---

private String **createNamesStringFromVector**(Vector<String> n)

Supportive method to create a String from a Vector. The String format is the required input data format for the MATLAB engine

Returns:

String names

---

private String **createTimesStringFromVector**(Vector<Integer> times)

Supportive method to create a String from a Vector. The String format is the required input data format for the MATLAB engine

Returns:

String times

```
private BufferedImage toBufferedImage(Image src)
```

```
private void save(BufferedImage image,
                  String ext)
```

Package applications.directMail

Class Summary		Page
AssistantUpdateContacts	The marketing assistant executes the tasks specified in an Excel file, updates the file and saves the updated version.	168
clusteringByMatlab	This application calls MATLAB to execute a clustering algorithm.	169
CreateCustomersToContactXL	Creates an Excel File that contains all the customers that all marketing assistants should contact through direct-mail.	170
CreateExcelForAssistant	An application that returns an Excel File with the tasks (customer contacts) that one marketing assistant should perform.	171
CreateExcelFromMap	This application copies some specified ranges from an Excel File to another.	172
CreateExcelSegmentation	This application takes the clustering MATLAB results and returns an Excel File with clusters information.	173
CreateOfferFromTxt	An application used to read a txt file and transform it into an `Offer` object.	174
ExecuteAssignClusters	Supporting application which opens and handles results from the `AssignClustersGUI` GUI.	175
ExecuteReviewDraft	Supporting application which opens and handles results from the `ReviewDraftGUI` GUI.	176
ExecuteROI	Supporting application which opens and handles results from the `MarketingROI` GUI.	177
GatherTODOCustomers	Reads an Excel file with the customer clusters, and creates a new Excel file as a worklist by joining customers from different clusters.	178
GetDataForClustering	It read an Excel file, and copies from it the data needed for the clustering algorithm to another file It is called by the `Segmentation` workflow.	179
GetDataForScheduling	Reads an Excel File and identifies the data needed for the scheduling algorithm.	180
GetDataForTAM	Supporting application that opens and handles the `GetExcelDataByRangeName` GUI.	181
GetDataFromSchedule	Reads an Excel File that contains schedule data and copies them into two Vectors.	183
mailTo	Sends an e-mail using a pre-defined account.	184
MediaDecisions	Supporting application that opens and handles the results of a `MediaDecisionsGUI` GUI.	185
RenameOrMoveFile	A supporting application that performs some ordinary File actions.	186
SchedulingByMatlab	Calls the MATLAB to apply a scheduling algorithm.	188

Enum Summary		Page
RenameOrMoveFile.FileAction		187

Class AssistantUpdateContacts

applications.directMail

```
java.lang.Object
  └ com.tilab.wade.performer.Application
      └ com.tilab.wade.performer.BaseApplication
          └ applications.directMail.AssistantUpdateContacts
```

```
public class AssistantUpdateContacts
extends com.tilab.wade.performer.BaseApplication
```

The marketing assistant executes the tasks specified in an Excel file, updates the file and saves the updated version. This application is called by the `AssistantLaunching` workflow The parameters are passed and get caught by the workflow class

Author:

Pavlos Delias

Fields inherited from class com.tilab.wade.performer.Application
`formalParams, myAgent, myExecutionId, myLogger, myName, mySessionId`

Constructor Summary	Page
`AssistantUpdateContacts``()`	169

Method Summary		Page
`void`	**`execute`**`()`	169

Methods inherited from class com.tilab.wade.performer.BaseApplication
`checkParameters, extract, fill, fillFormalParameters, getDataStore, setDataStore`

Methods inherited from class com.tilab.wade.performer.Application
`commit, fill, fill, fill, fill, fill, fireEvent, get, getControlInfo, getFormalParameters, getModifier, getModifiers, getTracer, getTransactionManager, getValid, getWorkflowDescriptor, getWorkflowFailureReason, getWorkflowLastErrorEvent, isTransactional, rollback, set, setWorkflowFailureReason, trace, trace`

Constructor Detail

```
public AssistantUpdateContacts()
```

Method Detail

```
public void execute()
             throws Throwable
```

Overrides:

`execute` in class `com.tilab.wade.performer.Application`

Throws:

`Throwable`

Class clusteringByMatlab

applications.directMail

```
java.lang.Object
  └ com.tilab.wade.performer.Application
      └ com.tilab.wade.performer.BaseApplication
          └ applications.directMail.clusteringByMatlab
```

```
public class clusteringByMatlab
extends com.tilab.wade.performer.BaseApplication
```

This application calls MATLAB to execute a clustering algorithm. The input data parameters and the results are handled by the `Segmentation` workflow, which calls this application

Author:
Pavlos Delias

Fields inherited from class com.tilab.wade.performer.Application
formalParams, myAgent, myExecutionId, myLogger, myName, mySessionId

Constructor Summary	Page
clusteringByMatlab()	170

Method Summary		Page
void	execute()	170

Methods inherited from class com.tilab.wade.performer.BaseApplication
checkParameters, extract, fill, fillFormalParameters, getDataStore, setDataStore

Methods inherited from class com.tilab.wade.performer.Application
commit, fill, fill, fill, fill, fill, fireEvent, get, getControlInfo, getFormalParameters, getModifier, getModifiers, getTracer, getTransactionManager, getValid, getWorkflowDescriptor, getWorkflowFailureReason, getWorkflowLastErrorEvent, isTransactional, rollback, set, setWorkflowFailureReason, trace, trace

Constructor Detail

`public` **`clusteringByMatlab`**`()`

Method Detail

```
public void execute()
             throws Throwable
```

Overrides:
`execute` in class `com.tilab.wade.performer.Application`

Throws:
`Throwable`

Class CreateCustomersToContactXL

applications.directMail

```
java.lang.Object
  └ com.tilab.wade.performer.Application
      └ com.tilab.wade.performer.BaseApplication
          └ applications.directMail.CreateCustomersToContactXL
```

```
public class CreateCustomersToContactXL
extends com.tilab.wade.performer.BaseApplication
```

Creates an Excel File that contains all the customers that all marketing assistants should contact through direct-mail. This application is called by the `LaunchCampaign` workflow, which manages the In/Out parameters

Author:
Pavlos Delias

Fields inherited from class com.tilab.wade.performer.Application
formalParams, myAgent, myExecutionId, myLogger, myName, mySessionId

Constructor Summary	Page
CreateCustomersToContactXL()	171

Method Summary		Page
void	execute()	171

Methods inherited from class com.tilab.wade.performer.BaseApplication
checkParameters, extract, fill, fillFormalParameters, getDataStore, setDataStore

Methods inherited from class com.tilab.wade.performer.Application
commit, fill, fill, fill, fill, fill, fireEvent, get, getControlInfo, getFormalParameters, getModifier, getModifiers, getTracer, getTransactionManager, getValid, getWorkflowDescriptor, getWorkflowFailureReason, getWorkflowLastErrorEvent, isTransactional, rollback, set, setWorkflowFailureReason, trace, trace

Constructor Detail

`public` **`CreateCustomersToContactXL`**`()`

Method Detail

```
public void execute()
             throws Throwable
```

Overrides:

`execute` in class `com.tilab.wade.performer.Application`

Throws:

`Throwable`

Class CreateExcelForAssistant

applications.directMail

```
java.lang.Object
  └ com.tilab.wade.performer.Application
      └ com.tilab.wade.performer.BaseApplication
          └ applications.directMail.CreateExcelForAssistant
```

```
public class CreateExcelForAssistant
extends com.tilab.wade.performer.BaseApplication
```

An application that returns an Excel File with the tasks (customer contacts) that one marketing assistant should perform. It is called by the `CreateJobSchedules` workflow class, iteratively for every assistant

Author:

Pavlos Delias

Fields inherited from class com.tilab.wade.performer.Application
formalParams, myAgent, myExecutionId, myLogger, myName, mySessionId

Constructor Summary	Page
CreateExcelForAssistant()	172

Method Summary	Page
void **execute**()	172

Methods inherited from class com.tilab.wade.performer.BaseApplication
checkParameters, extract, fill, fillFormalParameters, getDataStore, setDataStore

Methods inherited from class com.tilab.wade.performer.Application
commit, fill, fill, fill, fill, fill, fireEvent, get, getControlInfo, getFormalParameters, getModifier, getModifiers, getTracer, getTransactionManager, getValid, getWorkflowDescriptor, getWorkflowFailureReason, getWorkflowLastErrorEvent, isTransactional, rollback, set, setWorkflowFailureReason, trace, trace

Constructor Detail

`public CreateExcelForAssistant()`

Method Detail

```
public void execute()
             throws Throwable
```

Overrides:

`execute` in class `com.tilab.wade.performer.Application`

Throws:

`Throwable`

Class CreateExcelFromMap

applications.directMail

```
java.lang.Object
  └ com.tilab.wade.performer.Application
      └ com.tilab.wade.performer.BaseApplication
          └ applications.directMail.CreateExcelFromMap
```

```
public class CreateExcelFromMap
extends com.tilab.wade.performer.BaseApplication
```

This application copies some specified ranges from an Excel File to another. As an intermediate mean, ranges are stored to a HashMap. It is called by the `CreateTAMFile`.

Author:

Pavlos Delias

Fields inherited from class com.tilab.wade.performer.Application
formalParams, myAgent, myExecutionId, myLogger, myName, mySessionId

Constructor Summary	Page
CreateExcelFromMap()	173

Method Summary	Page
void **execute**()	173

Methods inherited from class com.tilab.wade.performer.BaseApplication
checkParameters, extract, fill, fillFormalParameters, getDataStore, setDataStore

Methods inherited from class com.tilab.wade.performer.Application
commit, fill, fill, fill, fill, fill, fireEvent, get, getControlInfo, getFormalParameters, getModifier, getModifiers, getTracer, getTransactionManager, getValid, getWorkflowDescriptor, getWorkflowFailureReason, getWorkflowLastErrorEvent, isTransactional, rollback, set, setWorkflowFailureReason, trace, trace

Constructor Detail

`public` **`CreateExcelFromMap`**`()`

Method Detail

```
public void execute()
            throws Throwable
```

Overrides:

`execute` in class `com.tilab.wade.performer.Application`

Throws:

`Throwable`

Class CreateExcelSegmentation

applications.directMail

```
java.lang.Object
  └ com.tilab.wade.performer.Application
      └ com.tilab.wade.performer.BaseApplication
          └ applications.directMail.CreateExcelSegmentation
```

```
public class CreateExcelSegmentation
extends com.tilab.wade.performer.BaseApplication
```

This application takes the clustering MATLAB results and returns an Excel File with clusters information. The first sheet contains the customers that each cluster includes, the second contains centroids data and the third sheet contains some meta-data about the clusters. The application is called by the `Segmentation` workflow.

Author:

Pavlos Delias

Fields inherited from class com.tilab.wade.performer.Application
formalParams, myAgent, myExecutionId, myLogger, myName, mySessionId

Constructor Summary	Page
CreateExcelSegmentation()	174

Method Summary		Page
void	**execute**()	174
static double	**getMaxValue**(double[] numbers)	174

Methods inherited from class com.tilab.wade.performer.BaseApplication
checkParameters, extract, fill, fillFormalParameters, getDataStore, setDataStore

Methods inherited from class com.tilab.wade.performer.Application
commit, fill, fill, fill, fill, fill, fireEvent, get, getControlInfo, getFormalParameters,

```
getModifier, getModifiers, getTracer, getTransactionManager, getValid, getWorkflowDescriptor,
getWorkflowFailureReason, getWorkflowLastErrorEvent, isTransactional, rollback, set,
setWorkflowFailureReason, trace, trace
```

Constructor Detail

`public **CreateExcelSegmentation**()`

Method Detail

```
public void execute()
             throws Throwable
```

Overrides:

`execute` in class `com.tilab.wade.performer.Application`

Throws:

`Throwable`

`public static double getMaxValue(double[] numbers)`

Class CreateOfferFromTxt

applications.directMail

```
java.lang.Object
  └ com.tilab.wade.performer.Application
      └ com.tilab.wade.performer.BaseApplication
          └ applications.directMail.CreateOfferFromTxt
```

```
public class CreateOfferFromTxt
extends com.tilab.wade.performer.BaseApplication
```

An application used to read a txt file and transform it into an `Offer` object. Actually, it returns a HashMap with all the Offer objects as the value set. It is called by the `PreparePiece` workflow.

Formal Parameters

- file
- (OUTPUT) offers

Author:

Pavlos Delias

Fields inherited from class com.tilab.wade.performer.Application
formalParams, myAgent, myExecutionId, myLogger, myName, mySessionId

Constructor Summary	Page
CreateOfferFromTxt()	175

Method Summary		Page
void	execute()	175

Methods inherited from class com.tilab.wade.performer.BaseApplication
checkParameters, extract, fill, fillFormalParameters, getDataStore, setDataStore

Methods inherited from class com.tilab.wade.performer.Application
commit, fill, fill, fill, fill, fill, fireEvent, get, getControlInfo, getFormalParameters,

`getModifier, getModifiers, getTracer, getTransactionManager, getValid, getWorkflowDescriptor, getWorkflowFailureReason, getWorkflowLastErrorEvent, isTransactional, rollback, set, setWorkflowFailureReason, trace, trace`

Constructor Detail

`public` **`CreateOfferFromTxt`**`()`

Method Detail

```
public void execute()
             throws Throwable
```

Overrides:

`execute` in class `com.tilab.wade.performer.Application`

Throws:

`Throwable`

Class ExecuteAssignClusters

applications.directMail

```
java.lang.Object
  └ com.tilab.wade.performer.Application
      └ com.tilab.wade.performer.BaseApplication
          └ applications.directMail.ExecuteAssignClusters
```

```
public class ExecuteAssignClusters
extends com.tilab.wade.performer.BaseApplication
```

Supporting application which opens and handles results from the `AssignClustersGUI` GUI.

Formal Parameters

- fileName
- agents
- (OUTPUT) assignments

Author:

Administrator

Field Summary		Page
`AssignClustersGUI`	**`myGui`**	176

Fields inherited from class com.tilab.wade.performer.Application
`formalParams, myAgent, myExecutionId, myLogger, myName, mySessionId`

Constructor Summary	Page
`ExecuteAssignClusters``()`	176

Method Summary		Page
`void`	**`execute`**`()`	176
`HashMap<String,Vector<jade.core.AID>>`	**`getAgents`**`()`	176
`String`	**`getFileName`**`()`	176
`void`	**`setAgents`**`(HashMap<String,Vector<jade.core.AID>> agents)`	176
`void`	**`setFileName`**`(String fileName)`	176

Methods inherited from class com.tilab.wade.performer.BaseApplication
`checkParameters, extract, fill, fillFormalParameters, getDataStore, setDataStore`

Methods inherited from class com.tilab.wade.performer.Application
`commit, fill, fill, fill, fill, fill, fireEvent, get, getControlInfo, getFormalParameters, getModifier, getModifiers, getTracer, getTransactionManager, getValid, getWorkflowDescriptor, getWorkflowFailureReason, getWorkflowLastErrorEvent, isTransactional, rollback, set, setWorkflowFailureReason, trace, trace`

Field Detail

`final AssignClustersGUI` **`myGui`**

Constructor Detail

`public` **`ExecuteAssignClusters`**`()`

Method Detail

```
public void execute()
             throws Throwable
```

Overrides:

`execute` in class `com.tilab.wade.performer.Application`

Throws:

`Throwable`

`public void` **`setFileName`**`(String fileName)`

`public String` **`getFileName`**`()`

`public void` **`setAgents`**`(HashMap<String,Vector<jade.core.AID>> agents)`

`public HashMap<String,Vector<jade.core.AID>>` **`getAgents`**`()`

Class ExecuteReviewDraft

applications.directMail

```
java.lang.Object
  └ com.tilab.wade.performer.Application
      └ com.tilab.wade.performer.BaseApplication
          └ applications.directMail.ExecuteReviewDraft
```

```
public class ExecuteReviewDraft
extends com.tilab.wade.performer.BaseApplication
```

Supporting application which opens and handles results from the `ReviewDraftGUI` GUI.

Formal Parameters

- (OUTPUT) result
- (OUTPUT) fileName
- MarCom

Author:

Pavlos Delias

Field Summary		**Page**
`Thread`	**`myThread`**	*177*
`ReviewDraftGUI`	**`review`**	*177*

Fields inherited from class com.tilab.wade.performer.Application
`formalParams, myAgent, myExecutionId, myLogger, myName, mySessionId`

Constructor Summary	Page
ExecuteReviewDraft()	177

Method Summary		Page
void	**execute**()	177

Methods inherited from class com.tilab.wade.performer.BaseApplication
`checkParameters, extract, fill, fillFormalParameters, getDataStore, setDataStore`

Methods inherited from class com.tilab.wade.performer.Application
`commit, fill, fill, fill, fill, fill, fireEvent, get, getControlInfo, getFormalParameters, getModifier, getModifiers, getTracer, getTransactionManager, getValid, getWorkflowDescriptor, getWorkflowFailureReason, getWorkflowLastErrorEvent, isTransactional, rollback, set, setWorkflowFailureReason, trace, trace`

Field Detail

public Thread **myThread**

final ReviewDraftGUI **review**

Constructor Detail

public **ExecuteReviewDraft**()

Method Detail

```
public void execute()
             throws Throwable
```

Overrides:

`execute` in class `com.tilab.wade.performer.Application`

Throws:

`Throwable`

Class ExecuteROI

applications.directMail

```
java.lang.Object
  └ com.tilab.wade.performer.Application
      └ com.tilab.wade.performer.BaseApplication
          └ applications.directMail.ExecuteROI
```

```
public class ExecuteROI
extends com.tilab.wade.performer.BaseApplication
```

Supporting application which opens and handles results from the `MarketingROI` GUI.

Formal Parameters

- (OUTPUT) ROI file

Author:

Administrator

Field Summary		Page
Thread	**myThread**	178
MarketingROI	**ROI**	178

Fields inherited from class com.tilab.wade.performer.Application
formalParams, myAgent, myExecutionId, myLogger, myName, mySessionId

Constructor Summary	Page
ExecuteROI()	178

Method Summary		Page
void	**execute**()	178
File	**getROIFile**()	178
void	**setROIFile**(File rOIFile)	178

Methods inherited from class com.tilab.wade.performer.BaseApplication
checkParameters, extract, fill, fillFormalParameters, getDataStore, setDataStore

Methods inherited from class com.tilab.wade.performer.Application
commit, fill, fill, fill, fill, fill, fireEvent, get, getControlInfo, getFormalParameters, getModifier, getModifiers, getTracer, getTransactionManager, getValid, getWorkflowDescriptor, getWorkflowFailureReason, getWorkflowLastErrorEvent, isTransactional, rollback, set, setWorkflowFailureReason, trace, trace

Field Detail

public Thread **myThread**

final MarketingROI **ROI**

Constructor Detail

public **ExecuteROI**()

Method Detail

public void **execute**()
throws Throwable

Overrides:

execute in class com.tilab.wade.performer.Application

Throws:

Throwable

public void **setROIFile**(File rOIFile)

public File **getROIFile**()

Class GatherTODOCustomers

applications.directMail

```
java.lang.Object
  └ com.tilab.wade.performer.Application
      └ com.tilab.wade.performer.BaseApplication
          └ applications.directMail.GatherTODOCustomers
```

public class **GatherTODOCustomers**
extends com.tilab.wade.performer.BaseApplication

Reads an Excel file with the customer clusters, and creates a new Excel file as a worklist by joining customers from different clusters. It is called by the `LaunchCampaign` workflow.

Formal Parameters

- fileName
- groupOfAgents
- assignments
- (OUTPUT) todoLists

Author:

Pavlos Delias

Fields inherited from class com.tilab.wade.performer.Application
`formalParams, myAgent, myExecutionId, myLogger, myName, mySessionId`

Constructor Summary	Page
`GatherTODOCustomers``()`	179

Method Summary		Page
`void`	**`execute`**`()`	179

Methods inherited from class com.tilab.wade.performer.BaseApplication
`checkParameters, extract, fill, fillFormalParameters, getDataStore, setDataStore`

Methods inherited from class com.tilab.wade.performer.Application
`commit, fill, fill, fill, fill, fill, fireEvent, get, getControlInfo, getFormalParameters, getModifier, getModifiers, getTracer, getTransactionManager, getValid, getWorkflowDescriptor, getWorkflowFailureReason, getWorkflowLastErrorEvent, isTransactional, rollback, set, setWorkflowFailureReason, trace, trace`

Constructor Detail

```
public GatherTODOCustomers()
```

Method Detail

```
public void execute()
             throws Throwable
```

Overrides:

`execute` in class `com.tilab.wade.performer.Application`

Throws:

`Throwable`

Class GetDataForClustering

applications.directMail

```
java.lang.Object
  └ com.tilab.wade.performer.Application
      └ com.tilab.wade.performer.BaseApplication
          └ applications.directMail.GetDataForClustering
```

```
public class GetDataForClustering
extends com.tilab.wade.performer.BaseApplication
```

It read an Excel file, and copies from it the data needed for the clustering algorithm to another file It is called by the `Segmentation` workflow.

Formal Parameters

- rangeName
- sheetName
- fileName
- (OUTPUT) cells

Author:
Pavlos Delias

Fields inherited from class com.tilab.wade.performer.Application
`formalParams, myAgent, myExecutionId, myLogger, myName, mySessionId`

Constructor Summary	Page
`GetDataForClustering``()`	*180*

Method Summary		Page
`void`	**`execute`**`()`	*180*

Methods inherited from class com.tilab.wade.performer.BaseApplication
`checkParameters, extract, fill, fillFormalParameters, getDataStore, setDataStore`

Methods inherited from class com.tilab.wade.performer.Application
`commit, fill, fill, fill, fill, fill, fireEvent, get, getControlInfo, getFormalParameters, getModifier, getModifiers, getTracer, getTransactionManager, getValid, getWorkflowDescriptor, getWorkflowFailureReason, getWorkflowLastErrorEvent, isTransactional, rollback, set, setWorkflowFailureReason, trace, trace`

Constructor Detail

```
public GetDataForClustering()
```

Method Detail

```
public void execute()
             throws Throwable
```

Overrides:
`execute` in class `com.tilab.wade.performer.Application`

Throws:
`Throwable`

Class GetDataForScheduling

applications.directMail

```
java.lang.Object
  └ com.tilab.wade.performer.Application
      └ com.tilab.wade.performer.BaseApplication
          └ applications.directMail.GetDataForScheduling
```

```
public class GetDataForScheduling
extends com.tilab.wade.performer.BaseApplication
```

Reads an Excel File and identifies the data needed for the scheduling algorithm. It is called by the `CreateJobSchedules` workflow.

Formal Parameters

- customersToContactFile
- (OUTPUT) customerNames
- (OUTPUT) processingTimes

Author:
Pavlos Delias

Fields inherited from class com.tilab.wade.performer.Application
`formalParams, myAgent, myExecutionId, myLogger, myName, mySessionId`

Constructor Summary	Page
`GetDataForScheduling``()`	*181*

Method Summary		Page
`void`	**`execute`**`()`	*181*

Methods inherited from class com.tilab.wade.performer.BaseApplication
`checkParameters, extract, fill, fillFormalParameters, getDataStore, setDataStore`

Methods inherited from class com.tilab.wade.performer.Application
`commit, fill, fill, fill, fill, fill, fireEvent, get, getControlInfo, getFormalParameters, getModifier, getModifiers, getTracer, getTransactionManager, getValid, getWorkflowDescriptor, getWorkflowFailureReason, getWorkflowLastErrorEvent, isTransactional, rollback, set, setWorkflowFailureReason, trace, trace`

Constructor Detail

```
public GetDataForScheduling()
```

Method Detail

```
public void execute()
             throws Throwable
```

Overrides:
`execute` in class `com.tilab.wade.performer.Application`

Throws:
`Throwable`

Class GetDataForTAM

applications.directMail

```
java.lang.Object
  └ com.tilab.wade.performer.Application
      └ com.tilab.wade.performer.BaseApplication
          └ applications.directMail.GetDataForTAM
```

```
public class GetDataForTAM
extends com.tilab.wade.performer.BaseApplication
```

Supporting application that opens and handles the GetExcelDataByRangeName GUI. It is called by the QuantifyTAM workflow.

Formal Parameters

- (OUTPUT) theFile
- (OUTPUT) rangeNames

Author:
Pavlos Delias

Field Summary		Page
private File	**fileName**	182
GetExcelDataByRangeName	**myGui**	182
Thread	**myThread**	182
private List<String>	**ranges**	182

Fields inherited from class com.tilab.wade.performer.Application
formalParams, myAgent, myExecutionId, myLogger, myName, mySessionId

Constructor Summary	Page
GetDataForTAM()	182

Method Summary		Page
void	**execute**()	182
File	**getFileName**()	183
List<String>	**getRanges**()	183
void	**setFileName**(File fileName)	183
void	**setRanges**(List<String> ranges)	183

Methods inherited from class com.tilab.wade.performer.BaseApplication
checkParameters, extract, fill, fillFormalParameters, getDataStore, setDataStore

Methods inherited from class com.tilab.wade.performer.Application
commit, fill, fill, fill, fill, fill, fireEvent, get, getControlInfo, getFormalParameters, getModifier, getModifiers, getTracer, getTransactionManager, getValid, getWorkflowDescriptor, getWorkflowFailureReason, getWorkflowLastErrorEvent, isTransactional, rollback, set, setWorkflowFailureReason, trace, trace

Field Detail

final GetExcelDataByRangeName **myGui**

public Thread **myThread**

private File **fileName**

private List<String> **ranges**

Constructor Detail

public **GetDataForTAM**()

Method Detail

```
public void execute()
             throws Throwable
```

Overrides:

`execute` in class `com.tilab.wade.performer.Application`

Throws:

`Throwable`

`public void setFileName(File fileName)`

`public File getFileName()`

`public List<String> getRanges()`

`public void setRanges(List<String> ranges)`

Class GetDataFromSchedule

applications.directMail

```
java.lang.Object
  └ com.tilab.wade.performer.Application
      └ com.tilab.wade.performer.BaseApplication
          └ applications.directMail.GetDataFromSchedule
```

```
public class GetDataFromSchedule
extends com.tilab.wade.performer.BaseApplication
```

Reads an Excel File that contains schedule data and copies them into two Vectors. It is called by the `AssistantLaunching` workflow.

Formal Parameters

- scheduleFileName
- (OUTPUT) customerNames
- (OUTPUT) processingTimes

Author:

Pavlos Delias

Fields inherited from class com.tilab.wade.performer.Application
`formalParams, myAgent, myExecutionId, myLogger, myName, mySessionId`

Constructor Summary	Page
`GetDataFromSchedule()`	184

Method Summary		Page
`void`	`execute()`	184

Methods inherited from class com.tilab.wade.performer.BaseApplication
`checkParameters, extract, fill, fillFormalParameters, getDataStore, setDataStore`

Methods inherited from class com.tilab.wade.performer.Application
`commit, fill, fill, fill, fill, fill, fireEvent, get, getControlInfo, getFormalParameters, getModifier, getModifiers, getTracer, getTransactionManager, getValid, getWorkflowDescriptor, getWorkflowFailureReason, getWorkflowLastErrorEvent, isTransactional, rollback, set, setWorkflowFailureReason, trace, trace`

Constructor Detail

public **GetDataFromSchedule**()

Method Detail

```
public void execute()
             throws Throwable
```

Overrides:

execute in class com.tilab.wade.performer.Application

Throws:

Throwable

Class mailTo

[applications.directMail]

```
java.lang.Object
  └ com.tilab.wade.performer.Application
      └ com.tilab.wade.performer.BaseApplication
          └ applications.directMail.mailTo
```

public class **mailTo**
extends com.tilab.wade.performer.BaseApplication

Sends an e-mail using a pre-defined account. Subject, content, recipients, and attachments are defined by the Formal parameters. It is called by the EstablishTargetMarkets workflow.

Formal Parameters

- subject
- content
- recipient
- attachmentFile

Author:

Pavlos Delias

Fields inherited from class com.tilab.wade.performer.Application
formalParams, myAgent, myExecutionId, myLogger, myName, mySessionId

Constructor Summary	Page
mailTo()	185

Method Summary		Page
void	**execute**()	185

Methods inherited from class com.tilab.wade.performer.BaseApplication
checkParameters, extract, fill, fillFormalParameters, getDataStore, setDataStore

Methods inherited from class com.tilab.wade.performer.Application
commit, fill, fill, fill, fill, fill, fireEvent, get, getControlInfo, getFormalParameters, getModifier, getModifiers, getTracer, getTransactionManager, getValid, getWorkflowDescriptor, getWorkflowFailureReason, getWorkflowLastErrorEvent, isTransactional, rollback, set, setWorkflowFailureReason, trace, trace

Constructor Detail

`public mailTo()`

Method Detail

```
public void execute()
             throws Throwable
```

Overrides:

`execute` in class `com.tilab.wade.performer.Application`

Throws:

`Throwable`

Class MediaDecisions

[applications.directMail]

```
java.lang.Object
  └ com.tilab.wade.performer.Application
      └ com.tilab.wade.performer.BaseApplication
          └ applications.directMail.MediaDecisions
```

```
public class MediaDecisions
extends com.tilab.wade.performer.BaseApplication
```

Supporting application that opens and handles the results of a `MediaDecisionsGUI` GUI.

Formal Parameter

- (OUTPUT) file

Author:

Pavlos Delias

Field Summary		Page
MediaDecisionsGUI	**myGUI**	186

Fields inherited from class com.tilab.wade.performer.Application
formalParams, myAgent, myExecutionId, myLogger, myName, mySessionId

Constructor Summary	Page
MediaDecisions()	186

Method Summary		Page
void	**execute**()	186

Methods inherited from class com.tilab.wade.performer.BaseApplication
checkParameters, extract, fill, fillFormalParameters, getDataStore, setDataStore

Methods inherited from class com.tilab.wade.performer.Application
commit, fill, fill, fill, fill, fill, fireEvent, get, getControlInfo, getFormalParameters, getModifier, getModifiers, getTracer, getTransactionManager, getValid, getWorkflowDescriptor, getWorkflowFailureReason, getWorkflowLastErrorEvent, isTransactional, rollback, set, setWorkflowFailureReason, trace, trace

Field Detail

`final MediaDecisionsGUI myGUI`

Constructor Detail

`public MediaDecisions()`

Method Detail

```
public void execute()
             throws Throwable
```

Overrides:

`execute` in class `com.tilab.wade.performer.Application`

Throws:

`Throwable`

Class RenameOrMoveFile

applications.directMail

```
java.lang.Object
  └ com.tilab.wade.performer.Application
      └ com.tilab.wade.performer.BaseApplication
          └ applications.directMail.RenameOrMoveFile
```

```
public class RenameOrMoveFile
extends com.tilab.wade.performer.BaseApplication
```

A supporting application that performs some ordinary File actions.

Formal Parameters

- oldFile
- (OUTPUT) newFile

Author:

Pavlos Delias

Nested Class Summary		Page
static enum	**RenameOrMoveFile.FileAction**	*187*

Fields inherited from class com.tilab.wade.performer.Application
formalParams, myAgent, myExecutionId, myLogger, myName, mySessionId

Constructor Summary	Page
RenameOrMoveFile()	*187*

Method Summary		Page
void	**execute**()	*187*

Methods inherited from class com.tilab.wade.performer.BaseApplication
checkParameters, extract, fill, fillFormalParameters, getDataStore, setDataStore

Methods inherited from class com.tilab.wade.performer.Application
commit, fill, fill, fill, fill, fill, fireEvent, get, getControlInfo, getFormalParameters,

```
getModifier, getModifiers, getTracer, getTransactionManager, getValid, getWorkflowDescriptor,
getWorkflowFailureReason, getWorkflowLastErrorEvent, isTransactional, rollback, set,
setWorkflowFailureReason, trace, trace
```

Constructor Detail

public **RenameOrMoveFile**()

Method Detail

public void **execute**()
throws Throwable

Overrides:

execute in class com.tilab.wade.performer.Application

Throws:

Throwable

Enum RenameOrMoveFile.FileAction

applications.directMail

java.lang.Object
└ java.lang.Enum<RenameOrMoveFile.FileAction>
└ applications.directMail.RenameOrMoveFile.FileAction

All Implemented Interfaces:

Comparable<RenameOrMoveFile.FileAction>, Serializable

Enclosing class:

RenameOrMoveFile

public static enum **RenameOrMoveFile.FileAction**
extends Enum<RenameOrMoveFile.FileAction>

Enum Constant Summary	Page
RENAME_IN_PLACE	*187*
RENAME_MOVE	*187*

Constructor Summary		Page
private	**RenameOrMoveFile.FileAction**()	*187*

Method Summary		Page
static RenameOrMoveFile.FileAction	**valueOf**(String name)	*187*
static RenameOrMoveFile.FileAction[]	**values**()	*187*

Enum Constant Detail

public static final RenameOrMoveFile.FileAction **RENAME_IN_PLACE**

public static final RenameOrMoveFile.FileAction **RENAME_MOVE**

Constructor Detail

private **RenameOrMoveFile.FileAction**()

Method Detail

public static RenameOrMoveFile.FileAction[] **values**()

public static RenameOrMoveFile.FileAction **valueOf**(String name)

Class SchedulingByMatlab

applications.directMail

```
java.lang.Object
  └ com.tilab.wade.performer.Application
      └ com.tilab.wade.performer.BaseApplication
          └ applications.directMail.SchedulingByMatlab
```

```
public class SchedulingByMatlab
extends com.tilab.wade.performer.BaseApplication
```

Calls the MATLAB to apply a scheduling algorithm. The MATLAB engine is called by synchronized statements to assure non-existence of conflicts.

Formal Parameters

- names
- times
- num
- (OUTPUT) startTimes
- (OUTPUT) processors

Author:
Pavlos Delias

Field Summary		Page
private String	**imageFileName**	189
private static Object	**lock**	189

Fields inherited from class com.tilab.wade.performer.Application
formalParams, myAgent, myExecutionId, myLogger, myName, mySessionId

Constructor Summary	Page
SchedulingByMatlab()	189

Method Summary		Page
private String	**createNamesStringFromVector**(Vector<String> n) Transforms a Vector into an appropriate String to be entered into MATLAB	189
private String	**createTimesStringFromVector**(Vector<Integer> t) Transforms a Vector into an appropriate String to be entered into MATLAB	189
void	**execute**()	189
private void	**save**(BufferedImage image, String ext)	189
private BufferedImage	**toBufferedImage**(Image src)	189

Methods inherited from class com.tilab.wade.performer.BaseApplication
checkParameters, extract, fill, fillFormalParameters, getDataStore, setDataStore

Methods inherited from class com.tilab.wade.performer.Application

`commit, fill, fill, fill, fill, fill, fireEvent, get, getControlInfo, getFormalParameters, getModifier, getModifiers, getTracer, getTransactionManager, getValid, getWorkflowDescriptor, getWorkflowFailureReason, getWorkflowLastErrorEvent, isTransactional, rollback, set, setWorkflowFailureReason, trace, trace`

Field Detail

`private String` **`imageFileName`**

`private static Object` **`lock`**

Constructor Detail

`public` **`SchedulingByMatlab`**`()`

Method Detail

```
public void execute()
             throws Throwable
```

Overrides:

`execute` in class `com.tilab.wade.performer.Application`

Throws:

`Throwable`

`private String` **`createNamesStringFromVector`**`(Vector<String> n)`

Transforms a Vector into an appropriate String to be entered into MATLAB

Returns:

String MATLAB statement

`private String` **`createTimesStringFromVector`**`(Vector<Integer> t)`

Transforms a Vector into an appropriate String to be entered into MATLAB

Returns:

String MATLAB Statement

`private BufferedImage` **`toBufferedImage`**`(Image src)`

```
private void save(BufferedImage image,
                  String ext)
```

Package generic

Class Summary		Page
AssignClustersGUI	A GUI to help assign clusters to agents.	190
GetExcelDataByRangeName	Reads some specified cell areas (ranges) from an Excel File.	193
MarketingDirectorGui	A supportive GUI to help Marketing Director functions (e.g., upload the checklist file)	195
MarketingROI	A supportive GUI to help create a Return on Investment report.	196
MediaDecisionsGUI	A supportive GUI to specify media requirements for every cluster (customer segment).	199
MediaDecisionsGUI.Cluster	An inner class used by the `MediaDecisionsGUI` to represent the cluster notion	202
Product		204
ReviewDraftGUI	A supportive GUI to help marketing communicator review artwork drafts.	205

Enum Summary		Page
MediaDecisionsGUI.MediaFormat		204

Class AssignClustersGUI

generic

```
java.lang.Object
  └ java.awt.Component
      └ java.awt.Container
          └ java.awt.Window
              └ java.awt.Frame
                  └ javax.swing.JFrame
                      └ generic.AssignClustersGUI
```

All Implemented Interfaces:

Accessible, ImageObserver, MenuContainer, RootPaneContainer, Serializable, TransferHandler.HasGetTransferHandler, WindowConstants

```
public class AssignClustersGUI
extends JFrame
```

A GUI to help assign clusters to agents.

Author:

Delias Pavlos

Field Summary		Page
private HashMap<String,Vector<jade.core.AID>>	agents	191
private HashMap<String,Vector<String>>	assignments	191
private JButton	btn_Assign	192
private JButton	btn_Done	192
private JButton	btn_Exit	192

private JButton	**btn_Remove**	192
private HashMap<String,Vector<Double>>	**clusters**	191
private JComboBox	**cmb_Agents**	192
private JComboBox	**cmb_Clusters**	192
private boolean	**done**	192
private JScrollPane	**jScrollPane1**	192
private JScrollPane	**jScrollPane2**	192
private JTree	**jtr_Assignments**	192
private JLabel	**lbl_SelectAgents**	192
private JLabel	**lbl_SelectCluster**	192
private ExecuteAssignClusters	**myApp**	191
private JTable	**tbl_ClusterData**	192

Constructor Summary	**Page**
AssignClustersGUI() Creates new form AssignClustersGUI	192

Method Summary		**Page**
private void	**assignCluster**() Assign a cluster to the selected agent, and removes the assigned cluster from the clusters set.	192
private void	**btn_AssignActionPerformed**(ActionEvent evt)	192
private void	**btn_DoneActionPerformed**(ActionEvent evt)	192
private void	**btn_ExitActionPerformed**(ActionEvent evt)	192
private void	**btn_RemoveActionPerformed**(ActionEvent evt)	192
private void	**cmb_ClustersItemStateChanged**(ItemEvent evt)	192
private void	**fillCombos**()	192
HashMap<String,Vector<String>>	**getAssignments**()	192
private void	**getClustersFromFile**(String xlFile) Reads the clusters' data from an Excel file with a specific format	192
ExecuteAssignClusters	**getMyApp**()	192
private void	**initComponents**() This method is called from within the constructor to initialize the form.	192
boolean	**isDone**()	192
void	**loadContent**(String file, HashMap<String,Vector<jade.core.AID>> ag) Loads the Excel clusters file with the getClustersFromFile(String) method and calls the fillCombos() method.	192
void	**setDone**(boolean done)	192
void	**setMyApp**(ExecuteAssignClusters myApp)	192
private void	**updateTable**(String clusterName) Supporting method to update the table model	192
private void	**UpdateTree**(HashMap<String,Vector<String>> m) Supporting method to refresh the graphical interface	192

Field Detail

private ExecuteAssignClusters **myApp**

private HashMap<String,Vector<Double>> **clusters**

private HashMap<String,Vector<jade.core.AID>> **agents**

private HashMap<String,Vector<String>> **assignments** `private boolean` **`done`**

`private JButton` **`btn_Assign`**

`private JButton` **`btn_Done`**

`private JButton` **`btn_Exit`**

`private JButton` **`btn_Remove`**

`private JComboBox` **`cmb_Agents`**

`private JComboBox` **`cmb_Clusters`**

`private JScrollPane` **`jScrollPane1`**

`private JScrollPane` **`jScrollPane2`**

`private JTree` **`jtr_Assignments`**

`private JLabel` **`lbl_SelectAgents`**

`private JLabel` **`lbl_SelectCluster`**

`private JTable` **`tbl_ClusterData`**

Constructor Detail

`public` **`AssignClustersGUI`**`()`

Creates new form AssignClustersGUI

Method Detail

```
public void loadContent(String file,
                        HashMap<String,Vector<jade.core.AID>> ag)
```

Loads the Excel clusters file with the `getClustersFromFile(String)` method and calls the `fillCombos()` method.

`private void` **`initComponents`**`()`

This method is called from within the constructor to initialize the form. WARNING: Do NOT modify this code. The content of this method is always regenerated by the Form Editor.

`private void` **`btn_AssignActionPerformed`**`(ActionEvent evt)`

`private void` **`btn_RemoveActionPerformed`**`(ActionEvent evt)`

`private void` **`btn_DoneActionPerformed`**`(ActionEvent evt)`

`private void` **`btn_ExitActionPerformed`**`(ActionEvent evt)`

`public void` **`setMyApp`**`(ExecuteAssignClusters myApp)`

`public ExecuteAssignClusters` **`getMyApp`**`()`

`private void` **`updateTable`**`(String clusterName)`

Supporting method to update the table model

`private void` **`getClustersFromFile`**`(String xlFile)`

Reads the clusters' data from an Excel file with a specific format

`private void` **`fillCombos`**`()`

`private void` **`assignCluster`**`()`

Assign a cluster to the selected agent, and removes the assigned cluster from the clusters set.

`private void` **`UpdateTree`**`(HashMap<String,Vector<String>> m)`

Supporting method to refresh the graphical interface

`private void` **`cmb_ClustersItemStateChanged`**`(ItemEvent evt)`

`public void` **`setDone`**`(boolean done)`

`public boolean` **`isDone`**`()`

`public HashMap<String,Vector<String>>` **`getAssignments`**`()`

Class GetExcelDataByRangeName

generic

```
java.lang.Object
  └ java.awt.Component
      └ java.awt.Container
          └ java.awt.Window
              └ java.awt.Frame
                  └ javax.swing.JFrame
                      └ generic.GetExcelDataByRangeName
```

All Implemented Interfaces:

Accessible, ImageObserver, MenuContainer, RootPaneContainer, Serializable, TransferHandler.HasGetTransferHandler, WindowConstants

```
public class GetExcelDataByRangeName
extends JFrame
```

Reads some specified cell areas (ranges) from an Excel File.

Author:

Pavlos Delias

Field Summary		Page
private DefaultListModel	allFieldsModel	194
private JButton	btn_Browse	194
private JButton	btn_Exit	194
private JButton	btn_OK	194
private JButton	btn_Select	194
private File	excelFile	194
private JFileChooser	fc	194
private JFileChooser	jFileChooser1	194
private JScrollPane	jScrollPane1	194
private JScrollPane	jScrollPane2	194
private JLabel	lbl_FilePath	194
private JList	lst_AllFields	194
private JList	lst_SelectedFields	194
private GetDataForTAM	myApp	194
private int	result	194
private DefaultListModel	selectedFieldsModel	194
private static long	serialVersionUID	194
private JTextField	txt_FilePath	194

Constructor Summary	Page
GetExcelDataByRangeName() Creates new form GetExcelDataByRangeName	194

Method Summary		Page
private void	**btn_BrowseActionPerformed**(ActionEvent evt)	194
private void	**btn_ExitActionPerformed**(ActionEvent evt)	194
private void	**btn_OKActionPerformed**(ActionEvent evt)	194
private void	**btn_SelectActionPerformed**(ActionEvent evt)	194
DefaultListModel	**getAllFieldsModel**()	195
GetDataForTAM	**getMyApp**()	195
private void	**getRangesFromFile**() Retrieves the named ranges from the excel file and update the jList components	194
int	**getResult**()	195
private void	**getSelectedRanges**() Fills the application's List with the selected elements	194
private void	**initComponents**()	194
void	**initListModels**()	194
void	**setAllFieldsModel**(DefaultListModel allFieldsModel)	195
void	**setMyApp**(GetDataForTAM myApp)	195
void	**setResult**(int result)	195

Field Detail

private static final long **serialVersionUID**

private GetDataForTAM **myApp**

private JFileChooser **fc**

private File **excelFile**

private int **result**

private DefaultListModel **allFieldsModel**

private DefaultListModel **selectedFieldsModel**

private JButton **btn_Browse**

private JButton **btn_Exit**

private JButton **btn_OK**

private JButton **btn_Select**

private JFileChooser **jFileChooser1**

private JScrollPane **jScrollPane1**

private JScrollPane **jScrollPane2**

private JLabel **lbl_FilePath**

private JList **lst_AllFields**

private JList **lst_SelectedFields**

private JTextField **txt_FilePath**

Constructor Detail

public **GetExcelDataByRangeName**()

Creates new form GetExcelDataByRangeName

Method Detail

private void **initComponents**()

private void **btn_BrowseActionPerformed**(ActionEvent evt)

private void **btn_OKActionPerformed**(ActionEvent evt)

private void **btn_ExitActionPerformed**(ActionEvent evt)

private void **btn_SelectActionPerformed**(ActionEvent evt)

private void **getRangesFromFile**()

Retrieves the named ranges from the excel file and update the jList components

private void **getSelectedRanges**()

Fills the application's List with the selected elements

public void **initListModels**()

```
public GetDataForTAM getMyApp()
```
```
public void setMyApp(GetDataForTAM myApp)
```
```
public DefaultListModel getAllFieldsModel()
```
```
public void setAllFieldsModel(DefaultListModel allFieldsModel)
```
```
public void setResult(int result)
```
```
public int getResult()
```

Class MarketingDirectorGui

generic

```
java.lang.Object
  └ java.awt.Component
      └ java.awt.Container
          └ java.awt.Window
              └ java.awt.Frame
                  └ javax.swing.JFrame
                      └ generic.MarketingDirectorGui
```

All Implemented Interfaces:

Accessible, ImageObserver, MenuContainer, RootPaneContainer, Serializable, TransferHandler.HasGetTransferHandler, WindowConstants

```
public class MarketingDirectorGui
extends JFrame
```

A supportive GUI to help Marketing Director functions (e.g., upload the checklist file)

Author:

Delias Pavlos

Field Summary		Page
private JButton	**btn_Browse**	196
private JButton	**btn_Exit**	196
private JButton	**btn_OK**	196
private JFileChooser	**fc**	196
private JLabel	**lbl_FilePath**	196
private MarketingDirector	**myAgent**	196
private JTextField	**txt_FilePath**	196

Constructor Summary	Page
MarketingDirectorGui() Creates new form MarketingDirectorGui	196
MarketingDirectorGui(MarketingDirector agent)	196

Method Summary		Page
private void	**btn_BrowseActionPerformed**(ActionEvent evt)	196
private void	**btn_ExitActionPerformed**(ActionEvent evt)	196
private void	**btn_OKActionPerformed**(ActionEvent evt)	196
private void	**initComponents**() This method is called from within the constructor to initialize the form.	196

static void	**main**(String[] args)	196

Field Detail

private MarketingDirector **myAgent**

private JFileChooser **fc**

private JButton **btn_Browse**

private JButton **btn_Exit**

private JButton **btn_OK**

private JLabel **lbl_FilePath**

private JTextField **txt_FilePath**

Constructor Detail

public **MarketingDirectorGui**()

Creates new form MarketingDirectorGui

public **MarketingDirectorGui**(MarketingDirector agent)

Method Detail

private void **initComponents**()

This method is called from within the constructor to initialize the form. WARNING: Do NOT modify this code. The content of this method is always regenerated by the Form Editor.

private void **btn_BrowseActionPerformed**(ActionEvent evt)

private void **btn_OKActionPerformed**(ActionEvent evt)

private void **btn_ExitActionPerformed**(ActionEvent evt)

public static void **main**(String[] args)

Class MarketingROI

generic

```
java.lang.Object
  └ java.awt.Component
      └ java.awt.Container
          └ java.awt.Window
              └ java.awt.Frame
                  └ javax.swing.JFrame
                      └ generic.MarketingROI
```

All Implemented Interfaces:

Accessible, ImageObserver, MenuContainer, RootPaneContainer, Serializable, TransferHandler.HasGetTransferHandler, WindowConstants

public class **MarketingROI**
extends JFrame

A supportive GUI to help create a Return on Investment report.

Author:

Pavlos Delias

Field Summary		Page
private JButton	**btn_Calculate**	198
private JButton	**btn_CreateReport**	198
private JButton	**btn_Exit**	198

private JButton	btn_Save	198
private double	costCustomer	198
private double	costPiece	198
private double	costResponse	198
static int	CREATE	198
static int	CREATE_AND_SAVE	198
private JFileChooser	fc	198
private JScrollPane	jScrollPane1	198
private JScrollPane	jScrollPane2	198
private JSeparator	jSeparator1	198
private JLabel	lbl_ConversionRate	198
private JLabel	lbl_NumberPieces	198
private JLabel	lbl_ProfitSale	198
private JLabel	lbl_ResponseRate	198
private JLabel	lbl_TotalCosts	198
private ExecuteROI	myApp	198
private int	numBuyers	198
private int	numResponders	198
private int	result	198
private double	ROI	198
static int	SUCCESS	198
private JTable	tbl_Results	198
private double	totalProfit	198
private JTextField	txt_ConversionRate	198
private JTextField	txt_NumberPieces	198
private JTextField	txt_ProfitSale	198
private JTextField	txt_ResponseRate	198
private JTextField	txt_TotalCosts	198

Constructor Summary	Page
MarketingROI() Creates new form MarketingROI	198

Method Summary		Page
private void	btn_CalculateActionPerformed(ActionEvent evt)	198
private void	btn_CreateReportActionPerformed(ActionEvent evt)	199
private void	btn_ExitActionPerformed(ActionEvent evt)	199
private void	btn_SaveActionPerformed(ActionEvent evt)	199

private void	**calculate**() Calculates some ROI metrics based on GUI input data.	199
private void	**createReport**(int action) Creates a report in a .doc format using a document template and GUI's data.	199
ExecuteROI	**getMyApp**()	199
int	**getResult**()	199
private void	**initComponents**() This method is called from within the constructor to initialize the form.	198
static void	**main**(String[] args)	199
void	**setMyApp**(ExecuteROI myApp)	199
void	**setResult**(int result)	199
private void	**showResults**() Refresh table model	199

Field Detail

private JFileChooser **fc**

private int **numResponders**

private int **numBuyers**

private double **costResponse**

private double **costCustomer**

private double **totalProfit**

private double **costPiece**

private double **ROI**

static final int **CREATE**

static final int **CREATE_AND_SAVE**

public static final int **SUCCESS**

private int **result**

private ExecuteROI **myApp**

private JButton **btn_Calculate**

private JButton **btn_CreateReport**

private JButton **btn_Exit**

private JScrollPane **jScrollPane1**

private JScrollPane **jScrollPane2**

private JSeparator **jSeparator1**

private JButton **btn_Save**

private JLabel **lbl_ConversionRate**

private JLabel **lbl_NumberPieces**

private JLabel **lbl_ProfitSale**

private JLabel **lbl_ResponseRate**

private JLabel **lbl_TotalCosts**

private JTable **tbl_Results**

private JTextField **txt_ConversionRate**

private JTextField **txt_NumberPieces**

private JTextField **txt_ProfitSale**

private JTextField **txt_ResponseRate**

private JTextField **txt_TotalCosts**

Constructor Detail

public **MarketingROI**()

Creates new form MarketingROI

Method Detail

private void **initComponents**()

This method is called from within the constructor to initialize the form. WARNING: Do NOT modify this code. The content of this method is always regenerated by the Form Editor.

private void **btn_CalculateActionPerformed**(ActionEvent evt) `private void btn_CreateReportActionPerformed(ActionEvent evt)`

`private void btn_ExitActionPerformed(ActionEvent evt)`

`private void btn_SaveActionPerformed(ActionEvent evt)`

`public static void main(String[] args)`

Parameters:

`args` - the command line arguments

`private void calculate()`

Calculates some ROI metrics based on GUI input data. The metrics calculated are visible to a Table

`private void showResults()`

Refresh table model

`private void createReport(int action)`

Creates a report in a .doc format using a document template and GUI's data.

`public void setResult(int result)`

`public int getResult()`

`public void setMyApp(ExecuteROI myApp)`

`public ExecuteROI getMyApp()`

Class MediaDecisionsGUI

generic

```
java.lang.Object
  └ java.awt.Component
      └ java.awt.Container
          └ java.awt.Window
              └ java.awt.Frame
                  └ javax.swing.JFrame
                      └ generic.MediaDecisionsGUI
```

All Implemented Interfaces:

Accessible, ImageObserver, MenuContainer, RootPaneContainer, Serializable, TransferHandler.HasGetTransferHandler, WindowConstants

```
public class MediaDecisionsGUI
extends JFrame
```

A supportive GUI to specify media requirements for every cluster (customer segment). The specified requirements may be saved to a text file

Author:

Pavlos Delias

Nested Class Summary		Page
private class	**MediaDecisionsGUI.Cluster** An inner class used by the `MediaDecisionsGUI` to represent the cluster notion	202
static enum	**MediaDecisionsGUI.MediaFormat**	204

Field Summary		Page
private JButton	**btn_Assign**	201
private JButton	**btn_Browse**	201
private JButton	**btn_Done**	201
private JButton	**btn_Publish**	201

private HashMap<String,MediaDecisionsGUI.Cluster>	**clusters**	201
private JComboBox	**cmb_Clusters**	201
private boolean	**done**	201
private File	**excelFile**	201
private JFileChooser	**fc**	201
private JScrollPane	**jScrollPane1**	201
private JLabel	**lbl_Budget**	201
private JLabel	**lbl_MediaFormat**	201
private JLabel	**lbl_Quantity**	201
private JLabel	**lbl_Select**	201
private JLabel	**lbl_SelectCluster**	201
private boolean	**published**	201
private File	**publishFile**	201
private JRadioButton	**rdb_Brochure**	201
private JRadioButton	**rdb_Catalog**	201
private JRadioButton	**rdb_Flyer**	201
private JRadioButton	**rdb_Guift**	201
private ButtonGroup	**rdb_MediaFormat**	201
private JTable	**tbl_ClusterData**	201
private JTextField	**txt_Budget**	201
private JTextField	**txt_FileName**	201
private JTextField	**txt_Quantity**	201

Constructor Summary	**Page**
MediaDecisionsGUI() Creates new form MediaDecisionsGUI	201

Method Summary		**Page**
private void	**assignToCluster**(String clusterName) Cluster parameters are set	202
private void	**btn_AssignActionPerformed**(ActionEvent evt)	201
private void	**btn_BrowseActionPerformed**(ActionEvent evt)	201
private void	**btn_DoneActionPerformed**(ActionEvent evt)	201
private void	**btn_PublishActionPerformed**(ActionEvent evt)	201
private void	**cmb_ClustersItemStateChanged**(ItemEvent evt)	201
void	**createTextFile**() Creates a text files that contains all the media requirements for all clusters	202
private void	**fillCombo**() Updates combo box data	202
private void	**getClustersFromFile**(File xl) Read an Excel file and gets cluster-related data	201
File	**getPublishFile**()	202
private void	**initComponents**() This method is called from within the constructor to initialize the form.	201
boolean	**isDone**()	202
boolean	**isPublished**()	202

static void	**main**(String[] args)	201
void	**setDone**(boolean done)	202
void	**setPublished**(boolean published)	202
void	**setPublishFile**(File publishFile)	202
private void	**txt_FileNameActionPerformed**(ActionEvent evt)	201
private void	**updateTable**(String clusterName) Updates the table that presents the cluster's parameters	202

Field Detail

private JFileChooser **fc**

private File **excelFile**

private File **publishFile**

private boolean **done**

private boolean **published**

private HashMap<String,MediaDecisionsGUI.Cluster> **clusters**

private JButton **btn_Assign**

private JButton **btn_Browse**

private JButton **btn_Done**

private JButton **btn_Publish**

private JComboBox **cmb_Clusters**

private JScrollPane **jScrollPane1**

private JLabel **lbl_Budget**

private JLabel **lbl_MediaFormat**

private JLabel **lbl_Quantity**

private JLabel **lbl_Select**

private JLabel **lbl_SelectCluster**

private JRadioButton **rdb_Brochure**

private JRadioButton **rdb_Catalog**

private JRadioButton **rdb_Flyer**

private JRadioButton **rdb_Guift**

private ButtonGroup **rdb_MediaFormat**

private JTable **tbl_ClusterData**

private JTextField **txt_Budget**

private JTextField **txt_FileName**

private JTextField **txt_Quantity**

Constructor Detail

public **MediaDecisionsGUI**()

Creates new form MediaDecisionsGUI

Method Detail

private void **initComponents**()

This method is called from within the constructor to initialize the form. WARNING: Do NOT modify this code. The content of this method is always regenerated by the Form Editor.

private void **txt_FileNameActionPerformed**(ActionEvent evt)

private void **btn_BrowseActionPerformed**(ActionEvent evt)

private void **btn_AssignActionPerformed**(ActionEvent evt)

private void **btn_DoneActionPerformed**(ActionEvent evt)

private void **btn_PublishActionPerformed**(ActionEvent evt)

private void **cmb_ClustersItemStateChanged**(ItemEvent evt)

public static void **main**(String[] args)

Parameters:

args - the command line arguments

private void **getClustersFromFile**(File xl)

Read an Excel file and gets cluster-related data

`private void` **`fillCombo`**`()`

Updates combo box data

`private void` **`updateTable`**`(String clusterName)`

Updates the table that presents the cluster's parameters

`private void` **`assignToCluster`**`(String clusterName)`

Cluster parameters are set

```
public void createTextFile()
                    throws IOException
```

Creates a text files that contains all the media requirements for all clusters

Throws:

`IOException`

`public void` **`setDone`**`(boolean done)`

`public boolean` **`isDone`**`()`

`public void` **`setPublished`**`(boolean published)`

`public boolean` **`isPublished`**`()`

`public void` **`setPublishFile`**`(File publishFile)`

`public File` **`getPublishFile`**`()`

Class MediaDecisionsGUI.Cluster

generic

```
java.lang.Object
  └ generic.MediaDecisionsGUI.Cluster
```

Enclosing class:

MediaDecisionsGUI

```
private class MediaDecisionsGUI.Cluster
extends Object
```

An inner class used by the `MediaDecisionsGUI` to represent the cluster notion

Author:

Pavlos Delias

Field Summary		Page
private double	**budget**	203
private String	**character**	203
private MediaDecisionsGUI.MediaFormat	**format**	203
private String	**name**	203
private double	**percentage**	203
private HashMap<String,Object>	**propertiesSet**	203
private int	**quantity**	203
private int	**size**	203

Constructor Summary	Page
MediaDecisionsGUI.Cluster()	203

Method Summary		Page
double	getBudget()	203
String	getCharacter()	203
MediaDecisionsGUI.MediaFormat	getFormat()	203
String	getName()	203
double	getPercentage()	203
int	getQuantity()	203
int	getSize()	203
String	publish()	203
void	setBudget(double budget)	203
void	setCharacter(String character)	203
void	setFormat(MediaDecisionsGUI.MediaFormat format)	203
void	setName(String name)	203
void	setPercentage(double percentage)	203
void	setQuantity(int quantity)	203
void	setSize(int size)	203

Field Detail

private String **name**

private String **character**

private int **size**

private double **percentage**

private MediaDecisionsGUI.MediaFormat **format**

private int **quantity**

private double **budget**

private HashMap<String,Object> **propertiesSet**

Constructor Detail

public **MediaDecisionsGUI.Cluster**()

Method Detail

public void **setName**(String name)

public String **getName**()

public void **setCharacter**(String character)

public String **getCharacter**()

public void **setSize**(int size)

public int **getSize**()

public void **setPercentage**(double percentage)

public double **getPercentage**()

public void **setFormat**(MediaDecisionsGUI.MediaFormat format)

public MediaDecisionsGUI.MediaFormat **getFormat**()

public void **setQuantity**(int quantity)

public int **getQuantity**()

public void **setBudget**(double budget)

public double **getBudget**()

public String **publish**()

Returns:

String represenation of cluster in form of 'Name#000#format'

Enum MediaDecisionsGUI.MediaFormat

generic

```
java.lang.Object
  └ java.lang.Enum<MediaDecisionsGUI.MediaFormat>
      └ generic.MediaDecisionsGUI.MediaFormat
```

All Implemented Interfaces:

Comparable<MediaDecisionsGUI.MediaFormat>, Serializable

Enclosing class:

MediaDecisionsGUI

```
public static enum MediaDecisionsGUI.MediaFormat
extends Enum<MediaDecisionsGUI.MediaFormat>
```

Enum Constant Summary	Page
BROCHURE	204
CATALOG	204
FLYER	204
GUIFT	204
UNSET	204

Constructor Summary		Page
private	**MediaDecisionsGUI.MediaFormat**()	204

Method Summary		Page
static MediaDecisionsGUI.MediaFormat	**valueOf**(String name)	204
static MediaDecisionsGUI.MediaFormat[]	**values**()	204

Enum Constant Detail

public static final MediaDecisionsGUI.MediaFormat **BROCHURE**

public static final MediaDecisionsGUI.MediaFormat **FLYER**

public static final MediaDecisionsGUI.MediaFormat **CATALOG**

public static final MediaDecisionsGUI.MediaFormat **GUIFT**

public static final MediaDecisionsGUI.MediaFormat **UNSET**

Constructor Detail

private **MediaDecisionsGUI.MediaFormat**()

Method Detail

public static MediaDecisionsGUI.MediaFormat[] **values**()

public static MediaDecisionsGUI.MediaFormat **valueOf**(String name)

Class Product

generic

```
java.lang.Object
  └ generic.Product
```

```
public class Product
extends Object
```

Field Summary		Page
private String	**checkListFile**	205
private boolean	**CheckListLoaded**	205
private ProductManager	**myManager**	205
private String	**name**	205

Constructor Summary	Page
Product()	205

Method Summary		Page
String	**getCheckListFile**()	205
String	**getName**()	205
boolean	**isCheckListLoaded**()	205
void	**setCheckListFile**(String checkListFile)	205
void	**setCheckListLoaded**(boolean checkListLoaded)	205
void	**setName**(String name)	205

Field Detail

private String **name**

private ProductManager **myManager**

private boolean **CheckListLoaded**

private String **checkListFile**

Constructor Detail

public **Product**()

Method Detail

public void **setName**(String name)

public String **getName**()

public void **setCheckListLoaded**(boolean checkListLoaded)

public boolean **isCheckListLoaded**()

public void **setCheckListFile**(String checkListFile)

public String **getCheckListFile**()

Class ReviewDraftGUI

generic

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Frame
                  └javax.swing.JFrame
                      └generic.ReviewDraftGUI
```

All Implemented Interfaces:

Accessible, ImageObserver, MenuContainer, RootPaneContainer, Serializable, TransferHandler.HasGetTransferHandler, WindowConstants

public class **ReviewDraftGUI**
extends JFrame

A supportive GUI to help marketing communicator review artwork drafts. The review can be saved in a .doc format

Author:

Pavlos Delias

Field Summary		Page
static int	ACCEPT	207
private ButtonGroup	buttonGroup1	207
private JCheckBox	chk_Generate	207
private JButton	cmd_Exit	207
private JButton	cmd_OK	207
private JFileChooser	fc	207
private JLabel	jLabel1	207
private JScrollPane	jScrollPane1	207
private JScrollPane	jScrollPane2	207
private JScrollPane	jScrollPane3	207
private JScrollPane	jScrollPane4	207
private JScrollPane	jScrollPane5	207
private JScrollPane	jScrollPane6	207
private JLabel	lbl_AdEasy	207
private JLabel	lbl_Benefit	208
private JLabel	lbl_Brand	208
private JLabel	lbl_Identified	208
private JLabel	lbl_Illustration	208
private JLabel	lbl_MessageClear	208
private jade.core.AID	MC	207
static int	NEEDS_WORK	207
private JRadioButton	rdb_Accept	208
private JRadioButton	rdb_Reject	208
private String	reportFileName	207
private int	result	207
private boolean	reviewed	207
private JSlider	sld_AdEasy	208
private JSlider	sld_Benefit	208
private JSlider	sld_Brand	208
private JSlider	sld_Identified	208

private JSlider	**sld_Illustration**	208
private JSlider	**sld_MessageClear**	208
private JTextArea	**txt_adEasy**	208
private JTextArea	**txt_Benefit**	208
private JTextArea	**txt_Brand**	208
private JTextArea	**txt_Identified**	208
private JTextArea	**txt_Illustration**	208
private JTextArea	**txt_MessageClear**	208

Constructor Summary	Page
ReviewDraftGUI() Creates new form ReviewDraftGUI	208

Method Summary		Page
private void	**cmd_ExitActionPerformed**(ActionEvent evt)	208
private void	**cmd_OKActionPerformed**(ActionEvent evt)	208
private void	**createReport**() Creates a report document based on the GUI data.	208
String	**getReportFileName**()	208
int	**getResult**()	208
private void	**initComponents**() This method is called from within the constructor to initialize the form.	208
boolean	**isReviewed**()	208
void	**setMC**(jade.core.AID mC)	208
void	**setReportFileName**(String reportFileName)	208
void	**setResult**(int result)	208
void	**setReviewed**(boolean reviewed)	208

Field Detail

private JFileChooser **fc**

private String **reportFileName**

private jade.core.AID **MC**

private int **result**

private boolean **reviewed**

public static final int **ACCEPT**

public static final int **NEEDS_WORK**

private ButtonGroup **buttonGroup1**

private JCheckBox **chk_Generate**

private JButton **cmd_Exit**

private JButton **cmd_OK**

private JLabel **jLabel1**

private JScrollPane **jScrollPane1**

private JScrollPane **jScrollPane2**

private JScrollPane **jScrollPane3**

private JScrollPane **jScrollPane4**

private JScrollPane **jScrollPane5**

private JScrollPane **jScrollPane6**

private JLabel **lbl_AdEasy**

private JLabel **lbl_Benefit**

private JLabel **lbl_Brand**

private JLabel **lbl_Identified**

private JLabel **lbl_Illustration**

private JLabel **lbl_MessageClear**

private JRadioButton **rdb_Accept**

private JRadioButton **rdb_Reject**

private JSlider **sld_AdEasy**

private JSlider **sld_Benefit**

private JSlider **sld_Brand**

private JSlider **sld_Identified**

private JSlider **sld_Illustration**

private JSlider **sld_MessageClear**

private JTextArea **txt_Benefit**

private JTextArea **txt_Brand**

private JTextArea **txt_Illustration**

private JTextArea **txt_MessageClear**

private JTextArea **txt_adEasy**

private JTextArea **txt_Identified**

Constructor Detail

public **ReviewDraftGUI**()

Creates new form ReviewDraftGUI

Method Detail

private void **initComponents**()

This method is called from within the constructor to initialize the form. WARNING: Do NOT modify this code. The content of this method is always regenerated by the Form Editor.

private void **cmd_ExitActionPerformed**(ActionEvent evt)

private void **cmd_OKActionPerformed**(ActionEvent evt)

private void **createReport**()

Creates a report document based on the GUI data. The user is prompted to save the report.

public void **setResult**(int result)

public int **getResult**()

public void **setReviewed**(boolean reviewed)

public boolean **isReviewed**()

public void **setReportFileName**(String reportFileName)

public String **getReportFileName**()

public void **setMC**(jade.core.AID mC)

Package marketing.wf.gui

Class Summary		Page
DBGUIUtils	A supportive class to handle main application's GUI interactions with the database.	209
GUIAgent	The agent behind the main application's GUI.	211
MarketingWFAboutDialog		212
marketingWFMainGUI	The main application's GUI.	214
marketingWFMainGUI.FilteredStream	An auxiliary class to support printing the logs to the GUI Logger.	224
ParametersPanel	A GUI supportive class.	225
ParametersPanel.Row		227

Class DBGUIUtils

marketing.wf.gui

```
java.lang.Object
  └marketing.wf.gui.DBGUIUtils
```

```
public class DBGUIUtils
extends Object
```

A supportive class to handle main application's GUI interactions with the database. Connections, Statements and results set are defined per method

Author:
Pavlos Delias

Field Summary		Page
Connection	conn	210
Statement	ins	210
private marketingWFMainGUI	myGui	210
ResultSet	rs	210
Statement	stmt	210

Constructor Summary	Page
DBGUIUtils(marketingWFMainGUI gui)	210

Method Summary		Page
boolean	checkStateRequirements(String state, String process) Performs a check if all requirements (files) necessary to begin the process instance from the specified state exist.	211
String[]	getAvailableTemplates() Queries the database with "SELECT process_type.Name FROM process_type;"	210
String[]	getExistingProcesses(int typeId) Get all process instances that belong to the specified process type	210
marketingWFMainGUI	getMyGui()	211

int	**getProcessId**(String name) Returns a integer with the process instance id	*210*
String[]	**getStatesOfProcess**(String processName) Based on the process template, the possible states that an instance of this template may be found are returned.	*210*
int	**getTypeId**(String typeName) Queries the database with SELECT process_type.Id FROM process_type WHERE process_type.Name = 'typeName';	*210*
HashMap<String,String>	**getWFProperties**(String state) Based on the state specified, the workflow that should be started is identified, and the appropriate performer types are returned	*211*
void	**insertNewProcess**(String name, String template) Inserts a new process instance for a specific process template.	*210*
void	**setMyGui**(marketingWFMainGUI myGui)	*211*

Field Detail

`private marketingWFMainGUI` **`myGui`**

`Connection` **`conn`**

`Statement` **`stmt`**

`Statement` **`ins`**

`ResultSet` **`rs`**

Constructor Detail

`public` **`DBGUIUtils`**`(marketingWFMainGUI gui)`

Method Detail

`public int` **`getTypeId`**`(String typeName)`

Queries the database with SELECT process_type.Id FROM process_type WHERE process_type.Name = 'typeName';

Returns:

int process type Id

`public String[]` **`getAvailableTemplates`**`()`

Queries the database with "SELECT process_type.Name FROM process_type;"

Returns:

An Array of Strings, each specifying a process template

```
public void insertNewProcess(String name,
                             String template)
```

Inserts a new process instance for a specific process template.

`public int` **`getProcessId`**`(String name)`

Returns a integer with the process instance id

Returns:

int The process instance id

`public String[]` **`getExistingProcesses`**`(int typeId)`

Get all process instances that belong to the specified process type

Returns:

An array of Strings

`public String[]` **`getStatesOfProcess`**`(String processName)`

Based on the process template, the possible states that an instance of this template may be found are returned.

Returns:
An array of Strings, specifying the states of the process

```
public boolean checkStateRequirements(String state,
                                      String process)
```

Performs a check if all requirements (files) necessary to begin the process instance from the specified state exist.

Parameters:
`state` - the state - milestone to start execution from
`process` - the process type

Returns:
An answer to the question are requirements fulfilled?

```
public HashMap<String,String> getWFProperties(String state)
```

Based on the state specified, the workflow that should be started is identified, and the appropriate performer types are returned

Returns:
A map containing the appropriate performer types

```
public void setMyGui(marketingWFMainGUI myGui)
```

```
public marketingWFMainGUI getMyGui()
```

Class GUIAgent

marketing.wf.gui

```
java.lang.Object
  └ jade.core.Agent
      └ marketing.wf.gui.GUIAgent
```

All Implemented Interfaces:
Runnable, jade.util.leap.Serializable, Serializable, jade.core.TimerListener

---

```
public class GUIAgent
extends jade.core.Agent
```

The agent behind the main application's GUI. During its setup:Registers the ontologies (`ContactCenterOntology`, `com.tilab.wade.ca.ontology.DeploymentOntology`, `com.tilab.wade.cfa.ontology.ConfigurationOntology`.Retrieves the configuration Agent from the WADE platformRetrieves the Controller agents form the WADE platformGets associated with the GUI

Author:
Pavlos Delias

Nested classes/interfaces inherited from class jade.core.Agent
`Agent.Interrupted`

Field Summary		Page
private jade.domain.FIPAAgentManagement.DFAgentDescription	**caTemplate**	212
private jade.core.AID	**cfa**	212
private marketingWFMainGUI	**myGUI**	212

Fields inherited from class jade.core.Agent
`AP_ACTIVE, AP_DELETED, AP_IDLE, AP_INITIATED, AP_MAX, AP_MIN, AP_SUSPENDED, AP_WAITING,`

```
D_ACTIVE, D_MAX, D_MIN, D_RETIRED, D_SUSPENDED, D_UNKNOWN, MSG_QUEUE_CLASS
```

Constructor Summary	Page
GUIAgent()	212

Method Summary		Page
jade.domain.FIPAAgentManagement.DFAgentDescription	**getCaTemplate**()	212
jade.core.AID	**getCfa**()	212
void	**retrieveStatus**()	212
protected void	**setup**()	212

Methods inherited from class jade.core.Agent
addBehaviour, afterClone, afterMove, beforeClone, beforeMove, blockingReceive, blockingReceive, blockingReceive, blockingReceive, changeStateTo, clean, createMessageQueue, doActivate, doClone, doDelete, doMove, doSuspend, doTimeOut, doWait, doWait, doWake, getAgentState, getAID, getAMS, getArguments, getBootProperties, getContainerController, getContentManager, getCurQueueSize, getDefaultDF, getHap, getHelper, getLocalName, getName, getO2AObject, getProperty, getQueueSize, getState, here, isRestarting, join, notifyChangeBehaviourState, notifyRestarted, postMessage, putBack, putO2AObject, receive, receive, removeBehaviour, removeTimer, restartLater, restore, restoreBufferedState, run, send, setArguments, setEnabledO2ACommunication, setGenerateBehaviourEvents, setO2AManager, setQueueSize, takeDown, waitUntilStarted, write

Field Detail

private jade.core.AID **cfa**

private jade.domain.FIPAAgentManagement.DFAgentDescription **caTemplate**

private marketingWFMainGUI **myGUI**

Constructor Detail

public **GUIAgent**()

Method Detail

protected void **setup**()

Overrides:

setup in class jade.core.Agent

public void **retrieveStatus**()

public jade.core.AID **getCfa**()

public jade.domain.FIPAAgentManagement.DFAgentDescription **getCaTemplate**()

Class MarketingWFAboutDialog

marketing.wf.gui

```
java.lang.Object
  └ java.awt.Component
      └ java.awt.Container
          └ java.awt.Window
              └ java.awt.Dialog
                  └ javax.swing.JDialog
                      └ marketing.wf.gui.MarketingWFAboutDialog
```

All Implemented Interfaces:

Accessible, ImageObserver, MenuContainer, RootPaneContainer, Serializable, TransferHandler.HasGetTransferHandler, WindowConstants

public class **MarketingWFAboutDialog**
extends JDialog

Author:
Pavlos Delias

Field Summary		Page
private JButton	cmd_Close	213
private JScrollPane	jScrollPane1	213
private JLabel	lbl_desc	213
private JLabel	lbl_Logo	213
private JLabel	lbl_Title	213
private JTextArea	txt_Desc	213

Constructor Summary	Page
MarketingWFAboutDialog(Frame parent, boolean modal) Creates new form MarketingWFAboutDialog	213

Method Summary		Page
private void	cmd_CloseActionPerformed(ActionEvent evt)	213
private void	initComponents() This method is called from within the constructor to initialize the form.	213
static void	main(String[] args)	213

Field Detail

`private JButton` **`cmd_Close`**

`private JScrollPane` **`jScrollPane1`**

`private JLabel` **`lbl_Logo`**

`private JLabel` **`lbl_Title`**

`private JLabel` **`lbl_desc`**

`private JTextArea` **`txt_Desc`**

Constructor Detail

```
public MarketingWFAboutDialog(Frame parent,
                              boolean modal)
```

Creates new form MarketingWFAboutDialog

Method Detail

`private void` **`initComponents`**`()`

This method is called from within the constructor to initialize the form. WARNING: Do NOT modify this code. The content of this method is always regenerated by the Form Editor.

`private void` **`cmd_CloseActionPerformed`**`(ActionEvent evt)`

`public static void` **`main`**`(String[] args)`

Parameters:

`args` - the command line arguments

Class marketingWFMainGUI

marketing.wf.gui

```
java.lang.Object
  └ java.awt.Component
      └ java.awt.Container
          └ java.awt.Window
              └ java.awt.Frame
                  └ javax.swing.JFrame
                      └ marketing.wf.gui.marketingWFMainGUI
```

All Implemented Interfaces:

Accessible, ImageObserver, MenuContainer, RootPaneContainer, Serializable, TransferHandler.HasGetTransferHandler, WindowConstants, com.tilab.wade.dispatcher.WorkflowResultListener

```
public class marketingWFMainGUI
extends JFrame
implements com.tilab.wade.dispatcher.WorkflowResultListener
```

The main application's GUI.

Author:

Pavlos Delias

Nested Class Summary		Page
class	**marketingWFMainGUI.FilteredStream** An auxiliary class to support printing the logs to the GUI Logger.	224

Field Summary		Page
private jade.core.AID	**applicationEngine**	219
PrintStream	**aPrintStream**	219
private HashMap<com.tilab.wade.commons.AgentType,Vector<jade.core.AID>>	**availableAgents** Hold the available agents per type	219
private ButtonGroup	**btnGrp_NewProcess**	219
private boolean	**catchErrors**	219
private jade.domain.FIPAAgentManagement.DFAgentDescription	**caTemplate**	219
private jade.core.AID	**cfa**	219
static long	**CFA_TIMEOUT**	219
private JComboBox	**cmb_DefineTemplate**	219
private JComboBox	**cmb_ExistingProcessName**	219
private JComboBox	**cmb_ExistingProcessState**	219
private JButton	**cmd_CheckStateReqs**	219
private JButton	**cmd_ExportConfiguration**	219
private JButton	**cmd_ImportConfiguration**	219
private JButton	**cmd_NewProcess**	219
private JButton	**cmd_OpenJadeConf**	219
private JButton	**cmd_OpenPlatformConfFile**	219
private JButton	**cmd_OpenWADEtypes**	219
private JButton	**cmd_RefreshAgents**	219

private JButton	cmd_RunWF	219
private JButton	cmd_SaveConfiguration	219
private JButton	cmd_SaveManagementFiles	219
private JButton	cmd_SelectPerformer	219
private JButton	cmd_StartDaemon	219
private JButton	cmd_StartMain	219
private JButton	cmd_StartPlatform	219
private JButton	cmd_StopPlatform	219
private int	cnt	219
private int	currentProcessId	219
private com.tilab.wade.dispatcher.DispatchingCapabilities	dc	219
private JFileChooser	GUIfc	219
private MonitoringWFService.MonitoringWFHelperImpl	helper A Service Helper for the monitoring Service	219
private JMenu	jMenu1	220
private JMenu	jMenu2	220
private JMenuBar	jMenuBar1	220
private JScrollPane	jScrollPane1	220
private JScrollPane	jScrollPane2	220
private JScrollPane	jScrollPane3	220
private JScrollPane	jScrollPane4	220
private JTree	jtr_Performers	220
private int	launcherCounter	219
private JLabel	lbl_ActiveConfiguration	220
private JLabel	lbl_AppropriatePerformer	220
private JLabel	lbl_CheckReqsResult	220
private JLabel	lbl_DefineTemplate	220
private JLabel	lbl_ExistingProcessName	220
private JLabel	lbl_ExistingProcessState	220
private JLabel	lbl_Logo	220
private JLabel	lbl_NewProcessName	220
private JLabel	lbl_Performer	220
private JLabel	lbl_PlatformStatus	220
private boolean	logFile	219
private String	logFileName	219
private JMenuItem	mnu_About	220
private JMenuItem	mnu_GetStatus	220
private JMenu	mnu_Help	220
private JMenuItem	mnu_OpenApi	220
private JMenuItem	mnu_SaveLog	220
private JMenuItem	mnu_Test1	220
private JMenuItem	mnu_Test2	220
private JMenu	mnu_Testing	220
private GUIAgent	myAgent	219
private DBGUIUtils	myDB	219
private ParametersPanel	parametersPanel	219

private String	platformStatus	219
private JPanel	pnl_Architecture	220
private JPanel	pnl_Configuration	220
private JPanel	pnl_InnerPlatform	220
private JPanel	pnl_Logger	220
private JPanel	pnl_Management	220
private JScrollPane	pnl_Parameters	220
private JPanel	pnl_Performers	220
private JPanel	pnl_Platform	220
private JPanel	pnl_Process	220
private JPanel	pnl_WFLauncher	220
private JPanel	pnl_Workflows	220
private JRadioButton	rdb_ExistingProcess	220
private JRadioButton	rdb_NewProcess	220
private JTabbedPane	tab_Sections	220
private JTextField	txt_ActiveConfiguration	220
private JTextField	txt_AppropriatePerformer	220
private JEditorPane	txt_Editor	220
private JTextArea	txt_Logger	220
private JTextField	txt_NewProcessName	220
private JTextField	txt_Performer	220
private JTextField	txt_PlatformStatus	220
private JTextArea	txt_WFEvents	220
private String	workflowExecutionId	219
private String	workflowToRun	219

Constructor Summary	Page
marketingWFMainGUI() Creates new form marketingWFMainGUI	220

Method Summary		Page
private String	buildConversationalId()	223
private void	cmb_ExistingProcessNameActionPerformed(ActionEvent evt)	221
private void	cmb_ExistingProcessStateActionPerformed(ActionEvent evt)	221
private void	cmd_CheckStateReqsActionPerformed(ActionEvent evt)	221
private void	cmd_ExportConfigurationActionPerformed(ActionEvent evt)	220
private void	cmd_ImportConfigurationActionPerformed(ActionEvent evt)	220
private void	cmd_NewProcessActionPerformed(ActionEvent evt)	221
private void	cmd_OpenJadeConfActionPerformed(ActionEvent evt)	221
private void	cmd_OpenPlatformConfFileActionPerformed(ActionEvent evt)	221
private void	cmd_OpenWADEtypesActionPerformed(ActionEvent evt)	221
private void	cmd_RefreshAgentsActionPerformed(ActionEvent evt)	221
private void	cmd_RunWFActionPerformed(ActionEvent evt)	221
private void	cmd_SaveConfigurationActionPerformed(ActionEvent evt)	220
private void	cmd_SaveManagementFilesActionPerformed(ActionEvent evt)	221
private void	cmd_SelectPerformerActionPerformed(ActionEvent evt)	221

private void	**cmd_StartDaemonActionPerformed**(ActionEvent evt)	221
private void	**cmd_StartMainActionPerformed**(ActionEvent evt)	221
private void	**cmd_StartPlatformActionPerformed**(ActionEvent evt)	221
private void	**cmd_StopPlatformActionPerformed**(ActionEvent evt)	221
private void	**configureAgentReferences**() The GUI sets up its reference with the configuration agent and the application engine agent (ApplicationEngineAgent.	221
jade.lang.acl.ACLMessage	**createAppEngineRequest**(jade.content.AgentAction action) Prepares an ACLMessage to be send to the Application Engine Agent	223
jade.lang.acl.ACLMessage	**createCfaRequest**(jade.content.AgentAction action) Prepares an ACLMessage to be send to the Configuration Agent	223
void	**fillStatesCombo**() Queries the database and finds process states according to process type.	222
private Vector<com.tilab.wade.commons.AgentType>	**getAllAgentTypes**() Gets all the agent types that are defined within the types.xml file.	223
private void	**getAvailableAgents**(Vector<com.tilab.wade.commons.AgentType> types) Gets all the agents that exist in the platform, grouping them by their type.	223
private void	**getAvailableTemplates**() Queries the database to get the available process types, and publish them to the respective combobox	221
int	**getCurrentProcessId**()	224
MonitoringWFService.MonitoringWFHelperImpl	**getHelper**()	224
GUIAgent	**getMyAgent**()	224
DBGUIUtils	**getMyDB**()	224
String	**getPlatformStatus**()	224
private jade.util.leap.List	**getWorkflowParameters**(String workflowName) It communicates with the Controller Agent of the local container to get the parameters that are specified by the workflow definition (class)	223
private void	**getWorkflowProperties**() For a specified state of a process, it gets the workflow class that it should be performed and it stores it into the workflowToRun field.	223
void	**handleAssignedId**(jade.core.AID executor, String executionId)	224
void	**handleCheckRequirements**() This method is called when the "Check Requirements" button is pressed.	222
private void	**handleException**(String op)	224
private void	**handleException**(String op, Exception e)	224
void	**handleExecutionCompleted**(jade.util.leap.List results, jade.core.AID executor, String executionId)	224
void	**handleExecutionError**(com.tilab.wade.performer.ontology.ExecutionError er, jade.core.AID executor, String executionId)	224
void	**handleExistingProcessNameSelected**() This process is called whenever the user selects a process instance from the corresponded comboBox.	222
void	**handleExistingProcessSelection**() This method is called when the radio button "Existing Process" is selected.	221
void	**handleExportConfiguration**(String configurationName, String configurationDesc, boolean override) This method is called when the "Export Configuration" button is pressed.	222

void	**handleImportConfiguration**() This method is called when the "Import Configuration" button is pressed.	222
void	**handleLoadError**(String reason)	224
void	**handleNewProcessAdded**() This method is called when the "Submit" button of the Workflow tab is pressed.	222
void	**handleNewProcessSelection**() This method is called when the radio button "New process" is selected.	221
void	**handleNotificationError**(jade.core.AID executor, String executionId)	224
void	**handleRunWorkflow**(String wf) This method starts execution of the workflow class specified in the parameters.	222
void	**handleSaveConfiguration**() This method is called when the "Save Configuration" button is pressed.	222
void	**handleSelectPerformer**() This method is called when the "Select Performer" button of the workflows Tab is pressed.	222
void	**handleShutdownPlatform**() This method is called when the "Shutdown Platform" button is pressed.	222
void	**handleStartBoot**() This method is called when the button "Start Boot Daemon" is pressed.	221
void	**handleStartMain**() This method is called when the "Start Main Container" button is pressed.	221
void	**handleStartupPlatform**() This method is called when the "Start Platform" button is pressed.	221
private void	**initComponents**() This method is called from within the constructor to initialize the form.	220
private void	**jtr_PerformersValueChanged**(TreeSelectionEvent evt)	221
void	**log**(String s)	223
static void	**main**(String[] args)	221
private void	**mnu_AboutActionPerformed**(ActionEvent evt)	221
private void	**mnu_GetStatusActionPerformed**(ActionEvent evt)	221
private void	**mnu_OpenApiActionPerformed**(ActionEvent evt)	221
private void	**mnu_SaveLogActionPerformed**(ActionEvent evt)	221
private void	**rdb_ExistingProcessActionPerformed**(ActionEvent evt)	221
private void	**rdb_NewProcessActionPerformed**(ActionEvent evt)	221
void	**saveLogFile**() Saves the Logger panel content to a file.	224
private String	**selectConfiguration**() Opens a dialog to select a platform's configuration file from the default configuration directory.	222
private void	**serveNewContactCenter**() This method is called when a new process of the ContactCenter type is submitted.	221
private void	**serveNewDirectMail**() This method is called when a new process of the DirectMail type is submitted.	221
void	**setCurrentProcessId**(int currentProcessId)	224
void	**setHelper**(MonitoringWFService.MonitoringWFHelperImpl helper)	224

void	**setMyAgent**(GUIAgent myAgent)	224
void	**setMyDB**(DBGUIUtils myDB)	224
void	**setPlatformStatus**(String status)	224
private void	**setProcessId2Engine**(int id)	223
private void	**setProcessId2Monitor**(int id)	223
private void	**startJADE**() Starts the JADE platform.	221
private void	**updateTree**(HashMap<com.tilab.wade.commons.AgentType,Vector<jade.core.AID>> map) A GUI supportive method to update the tree of the workflows Tab, that presents all the available agents, grouped by type.	223

Field Detail

static final long **CFA_TIMEOUT**

private int **cnt**

private String **platformStatus**

private GUIAgent **myAgent**

private jade.core.AID **cfa**

private jade.core.AID **applicationEngine**

private jade.domain.FIPAAgentManagement.DFAgentDescription **caTemplate**

private int **launcherCounter**

private String **workflowToRun**

private ParametersPanel **parametersPanel**

private com.tilab.wade.dispatcher.DispatchingCapabilities **dc**

private String **workflowExecutionId**

private int **currentProcessId**

private MonitoringWFService.MonitoringWFHelperImpl **helper**

A Service Helper for the monitoring Service

private DBGUIUtils **myDB**

private HashMap<com.tilab.wade.commons.AgentType,Vector<jade.core.AID>> **availableAgents**

Hold the available agents per type

private boolean **catchErrors**

private boolean **logFile**

private String **logFileName**

PrintStream **aPrintStream**

private JFileChooser **GUIfc**

private ButtonGroup **btnGrp_NewProcess**

private JComboBox **cmb_DefineTemplate**

private JComboBox **cmb_ExistingProcessName**

private JComboBox **cmb_ExistingProcessState**

private JButton **cmd_CheckStateReqs**

private JButton **cmd_ExportConfiguration**

private JButton **cmd_ImportConfiguration**

private JButton **cmd_NewProcess**

private JButton **cmd_OpenJadeConf**

private JButton **cmd_OpenWADEtypes**

private JButton **cmd_OpenPlatformConfFile**

private JButton **cmd_RefreshAgents**

private JButton **cmd_RunWF**

private JButton **cmd_SaveConfiguration**

private JButton **cmd_SaveManagementFiles**

private JButton **cmd_SelectPerformer**

private JButton **cmd_StartDaemon**

private JButton **cmd_StartMain**

private JButton **cmd_StartPlatform**

private JButton **cmd_StopPlatform**

private JMenu **jMenu1**

private JMenu **jMenu2**

private JMenuBar **jMenuBar1**

private JScrollPane **jScrollPane1**

private JScrollPane **jScrollPane2**

private JScrollPane **jScrollPane3**

private JScrollPane **jScrollPane4**

private JTree **jtr_Performers**

private JLabel **lbl_ActiveConfiguration**

private JLabel **lbl_AppropriatePerformer**

private JLabel **lbl_CheckReqsResult**

private JLabel **lbl_DefineTemplate**

private JLabel **lbl_ExistingProcessName**

private JLabel **lbl_ExistingProcessState**

private JLabel **lbl_NewProcessName**

private JLabel **lbl_Performer**

private JLabel **lbl_PlatformStatus**

private JLabel **lbl_Logo**

private JMenuItem **mnu_About**

private JMenuItem **mnu_GetStatus**

private JMenu **mnu_Help**

private JMenuItem **mnu_OpenApi**

private JMenuItem **mnu_SaveLog**

private JMenuItem **mnu_Test1**

private JMenuItem **mnu_Test2**

private JMenu **mnu_Testing**

private JPanel **pnl_Architecture**

private JPanel **pnl_Configuration**

private JPanel **pnl_InnerPlatform**

private JPanel **pnl_Logger**

private JPanel **pnl_Management**

private JScrollPane **pnl_Parameters**

private JPanel **pnl_Performers**

private JPanel **pnl_Platform**

private JPanel **pnl_Process**

private JPanel **pnl_WFLauncher**

private JPanel **pnl_Workflows**

private JRadioButton **rdb_ExistingProcess**

private JRadioButton **rdb_NewProcess**

private JTabbedPane **tab_Sections**

private JTextField **txt_ActiveConfiguration**

private JTextField **txt_AppropriatePerformer**

private JEditorPane **txt_Editor**

private JTextArea **txt_Logger**

private JTextField **txt_NewProcessName**

private JTextField **txt_Performer**

private JTextField **txt_PlatformStatus**

private JTextArea **txt_WFEvents**

Constructor Detail

public **marketingWFMainGUI**()

Creates new form marketingWFMainGUI

Method Detail

private void **initComponents**()

This method is called from within the constructor to initialize the form. WARNING: Do NOT modify this code. The content of this method is always regenerated by the Form Editor.

private void **cmd_ImportConfigurationActionPerformed**(ActionEvent evt)

private void **cmd_ExportConfigurationActionPerformed**(ActionEvent evt)

private void **cmd_SaveConfigurationActionPerformed**(ActionEvent evt) `private void` **`cmd_StartMainActionPerformed`**`(ActionEvent evt)`

`private void` **`cmd_StartDaemonActionPerformed`**`(ActionEvent evt)`

`private void` **`rdb_NewProcessActionPerformed`**`(ActionEvent evt)`

`private void` **`rdb_ExistingProcessActionPerformed`**`(ActionEvent evt)`

`private void` **`cmd_NewProcessActionPerformed`**`(ActionEvent evt)`

`private void` **`cmd_CheckStateReqsActionPerformed`**`(ActionEvent evt)`

`private void` **`cmd_StartPlatformActionPerformed`**`(ActionEvent evt)`

`private void` **`cmd_StopPlatformActionPerformed`**`(ActionEvent evt)`

`private void` **`cmb_ExistingProcessNameActionPerformed`**`(ActionEvent evt)`

`private void` **`cmd_SelectPerformerActionPerformed`**`(ActionEvent evt)`

`private void` **`cmd_RefreshAgentsActionPerformed`**`(ActionEvent evt)`

`private void` **`jtr_PerformersValueChanged`**`(TreeSelectionEvent evt)`

`private void` **`cmd_RunWFActionPerformed`**`(ActionEvent evt)`

`private void` **`cmb_ExistingProcessStateActionPerformed`**`(ActionEvent evt)`

`private void` **`mnu_SaveLogActionPerformed`**`(ActionEvent evt)`

`private void` **`mnu_GetStatusActionPerformed`**`(ActionEvent evt)`

`private void` **`mnu_OpenApiActionPerformed`**`(ActionEvent evt)`

`private void` **`mnu_AboutActionPerformed`**`(ActionEvent evt)`

`private void` **`cmd_OpenJadeConfActionPerformed`**`(ActionEvent evt)`

`private void` **`cmd_OpenWADEtypesActionPerformed`**`(ActionEvent evt)`

`private void` **`cmd_SaveManagementFilesActionPerformed`**`(ActionEvent evt)`

`private void` **`cmd_OpenPlatformConfFileActionPerformed`**`(ActionEvent evt)`

`public static void` **`main`**`(String[] args)`

Parameters:

`args` - the command line arguments

`private void` **`startJADE`**`()`

Starts the JADE platform. To adjust platform's properties, a .properties file is used.

`private void` **`configureAgentReferences`**`()`

The GUI sets up its reference with the configuration agent and the application engine agent (`ApplicationEngineAgent`.

`private void` **`getAvailableTemplates`**`()`

Queries the database to get the available process types, and publish them to the respective combobox

`private void` **`serveNewContactCenter`**`()`

This method is called when a new process of the ContactCenter type is submitted.

`private void` **`serveNewDirectMail`**`()`

This method is called when a new process of the DirectMail type is submitted.

`public void` **`handleStartMain`**`()`

This method is called when the "Start Main Container" button is pressed. Once the main container is started, the button is disabled, i.e., users can not start a second Main Container

`public void` **`handleStartupPlatform`**`()`

This method is called when the "Start Platform" button is pressed. It actually sends a REQUEST message to the Configuration Agent.

`public void` **`handleExistingProcessSelection`**`()`

This method is called when the radio button "Existing Process" is selected. It fetches available process instances of the specified process type, and it enables / disables GUI controls.

`public void` **`handleNewProcessSelection`**`()`

This method is called when the radio button "New process" is selected. It enables / disables GUI controls.

`public void` **`handleStartBoot`**`()`

This method is called when the button "Start Boot Daemon" is pressed. It start the Boot Daemon on the local host, taking as arguments the agents types file (types.xml) and the root configuration directory. Once the Daemon is started, the button is disabled.

`public void` **`handleImportConfiguration`**`()`

This method is called when the "Import Configuration" button is pressed. It actually sends a REQUEST message to the Configuration Agent.

`public void` **`handleSaveConfiguration`**`()`

This method is called when the "Save Configuration" button is pressed. It actually sends a REQUEST message to the Configuration Agent.

```
public void handleExportConfiguration(String configurationName,
                                      String configurationDesc,
                                      boolean override)
```

This method is called when the "Export Configuration" button is pressed. It opens a dialog to get the necessary input information. Ultimately, it sends a REQUEST message to the Configuration Agent.

`public void` **`handleShutdownPlatform`**`()`

This method is called when the "Shutdown Platform" button is pressed. It open a dialog to prompt the user if he wishes a soft shutdown or not. It ultimately sends a REQUEST message to the Configuration Agent.

`public void` **`handleNewProcessAdded`**`()`

This method is called when the "Submit" button of the Workflow tab is pressed. It registers a new process instance with the specified name and type with the database, and it starts serving the new process instance execution, according to the process type.

`public void` **`handleExistingProcessNameSelected`**`()`

This process is called whenever the user selects a process instance from the corresponded comboBox. It queries the database to get the process instance id and notifies the GUI, the Application Engine and the monitor service.

`public void` **`handleSelectPerformer`**`()`

This method is called when the "Select Performer" button of the workflows Tab is pressed. It sets the workflow to-be-performer to the selected agent.

`public void` **`handleCheckRequirements`**`()`

This method is called when the "Check Requirements" button is pressed. It queries the DB to check if the required documents to begin the selected state exist for the specific process instance.

`public void` **`handleRunWorkflow`**`(String wf)`

This method starts execution of the workflow class specified in the parameters. The performer is specified by another method (`handleSelectPerformer()` and the workflow parameters are specified through the GUI interface.

Parameters:

`wf` - - The workflow class to be executed

`public void` **`fillStatesCombo`**`()`

Queries the database and finds process states according to process type.

```
private String selectConfiguration()
                            throws Exception
```

Opens a dialog to select a platform's configuration file from the default configuration directory. Ultimately, it sends a REQUEST message to the Configuration Agent, which performs the task.

Returns:

String - Configuration name

Throws:

`Exception`

```
private Vector<com.tilab.wade.commons.AgentType> getAllAgentTypes()
                                                   throws Exception
```

Gets all the agent types that are defined within the types.xml file.

Returns:
Vector of AgentType

Throws:
`Exception`

```
private void getAvailableAgents(Vector<com.tilab.wade.commons.AgentType> types)
```

Gets all the agents that exist in the platform, grouping them by their type.

```
private void updateTree(HashMap<com.tilab.wade.commons.AgentType,Vector<jade.core.AID>> map)
```

A GUI supportive method to update the tree of the workflows Tab, that presents all the available agents, grouped by type.

```
private void getWorkflowProperties()
```

For a specified state of a process, it gets the workflow class that it should be performed and it stores it into the `workflowToRun` field. Additionally it find the appropriate performer type and it publishes it to the `txt_Performer` field.

```
private jade.util.leap.List getWorkflowParameters(String workflowName)
                                       throws Exception
```

It communicates with the Controller Agent of the local container to get the parameters that are specified by the workflow definition (class)

Parameters:
`workflowName` - the workflow class

Returns:
List - the parameters list

Throws:
`Exception`

```
private synchronized String buildConversationalId()
```

```
private void setProcessId2Monitor(int id)
```

```
private void setProcessId2Engine(int id)
```

```
void log(String s)
```

```
synchronized jade.lang.acl.ACLMessage createCfaRequest(jade.content.AgentAction action)
                                      throws jade.content.onto.OntologyException,
                                             jade.content.lang.Codec.CodecException
```

Prepares an ACLMessage to be send to the Configuration Agent

Parameters:
`action` - - The action that is requested for execution. Every action is specified in the package

Returns:
ACLMessage A REQUEST message

Throws:
`jade.content.onto.OntologyException`
`jade.content.lang.Codec.CodecException`
`Codec.CodecException`

```
synchronized jade.lang.acl.ACLMessage createAppEngineRequest(jade.content.AgentAction action)
                                            throws jade.content.onto.OntologyExcepti
on,
                                                   jade.content.lang.Codec.CodecExce
ption
```

Prepares an ACLMessage to be send to the Application Engine Agent

Parameters:
`action` - The action that is requested for execution. Every action is specified in the package

Returns:
ACLMessage A REQUEST message

Throws:
`jade.content.onto.OntologyException`
`jade.content.lang.Codec.CodecException`
`Codec.CodecException`

```
private void handleException(String op,
                             Exception e)
```

`private void handleException(String op)`

`public GUIAgent getMyAgent()`

`public void setMyAgent(GUIAgent myAgent)`

`public void setMyDB(DBGUIUtils myDB)`

`public DBGUIUtils getMyDB()`

`public void saveLogFile()`

Saves the Logger panel content to a file.

```
public void handleAssignedId(jade.core.AID executor,
                             String executionId)
```

Specified by:
`handleAssignedId` in interface `com.tilab.wade.dispatcher.WorkflowResultListener`

```
public void handleExecutionCompleted(jade.util.leap.List results,
                                     jade.core.AID executor,
                                     String executionId)
```

Specified by:
`handleExecutionCompleted` in interface `com.tilab.wade.dispatcher.WorkflowResultListener`

```
public void handleExecutionError(com.tilab.wade.performer.ontology.ExecutionError er,
                                 jade.core.AID executor,
                                 String executionId)
```

Specified by:
`handleExecutionError` in interface `com.tilab.wade.dispatcher.WorkflowResultListener`

`public void handleLoadError(String reason)`

Specified by:
`handleLoadError` in interface `com.tilab.wade.dispatcher.WorkflowResultListener`

```
public void handleNotificationError(jade.core.AID executor,
                                    String executionId)
```

Specified by:
`handleNotificationError` in interface `com.tilab.wade.dispatcher.WorkflowResultListener`

`public void setCurrentProcessId(int currentProcessId)`

`public int getCurrentProcessId()`

`public void setHelper(MonitoringWFService.MonitoringWFHelperImpl helper)`

`public MonitoringWFService.MonitoringWFHelperImpl getHelper()`

`public void setPlatformStatus(String status)`

`public String getPlatformStatus()`

Class marketingWFMainGUI.FilteredStream

marketing.wf.gui

```
java.lang.Object
  └ java.io.OutputStream
      └ java.io.FilterOutputStream
          └ marketing.wf.gui.marketingWFMainGUI.FilteredStream
```

All Implemented Interfaces:
Closeable, Flushable

Enclosing class:

[marketingWFMainGUI]

```
class marketingWFMainGUI.FilteredStream
extends FilterOutputStream
```

An auxiliary class to support printing the logs to the GUI Logger.

Author:

Pavlos Delias

Constructor Summary	**Page**
marketingWFMainGUI.FilteredStream(OutputStream aStream)	225

Method Summary		**Page**
void	**write**(byte[] b)	225
void	**write**(byte[] b, int off, int len)	225

Constructor Detail

```
public marketingWFMainGUI.FilteredStream(OutputStream aStream)
```

Method Detail

```
public void write(byte[] b)
           throws IOException
```

Overrides:

`write` in class `FilterOutputStream`

Throws:

`IOException`

```
public void write(byte[] b,
                  int off,
                  int len)
           throws IOException
```

Overrides:

`write` in class `FilterOutputStream`

Throws:

`IOException`

Class ParametersPanel

marketing.wf.gui

```
java.lang.Object
  └ java.awt.Component
      └ java.awt.Container
          └ javax.swing.JComponent
              └ javax.swing.JScrollPane
                  └ marketing.wf.gui.ParametersPanel
```

All Implemented Interfaces:

Accessible, ImageObserver, MenuContainer, ScrollPaneConstants, Serializable, TransferHandler.HasGetTransferHandler

```
public class ParametersPanel
extends JScrollPane
```

A GUI supportive class. It is used to handle the workflow parameters

Author:
Pavlos Delias

Nested Class Summary		Page
private class	**ParametersPanel.Row**	227

Field Summary		Page
private marketingWFMainGUI	**launcherGUI**	226
private AbstractTableModel	**model**	226
private TableCellRenderer	**renderer**	226
private List<ParametersPanel.Row>	**rows**	226
private JTable	**table**	226

Constructor Summary	Page
ParametersPanel(marketingWFMainGUI launcherGUI)	226

Method Summary		Page
boolean	**checkInputParameters**()	226
private String	**getParameterMode**(int mode)	226
jade.util.leap.List	**getParameters**()	226
void	**reset**()	226
void	**setFieldsEnabled**(boolean enabled)	226
void	**setParameters**(jade.util.leap.List parameters)	226
void	**setResult**(jade.util.leap.List parameters)	226

Field Detail

private JTable **table**

private AbstractTableModel **model**

private TableCellRenderer **renderer**

private List<ParametersPanel.Row> **rows**

private marketingWFMainGUI **launcherGUI**

Constructor Detail

public **ParametersPanel**(marketingWFMainGUI launcherGUI)

Method Detail

public void **setParameters**(jade.util.leap.List parameters)

public jade.util.leap.List **getParameters**()

boolean **checkInputParameters**()

public void **setResult**(jade.util.leap.List parameters)

private String **getParameterMode**(int mode)

void **setFieldsEnabled**(boolean enabled)

void **reset**()

Class ParametersPanel.Row

marketing.wf.gui

```
java.lang.Object
  └─ marketing.wf.gui.ParametersPanel.Row
```

Enclosing class:

ParametersPanel

```
private class ParametersPanel.Row
extends Object
```

Field Summary		Page
private JLabel	**mode**	227
private JLabel	**name**	227
private com.tilab.wade.performer.descriptors.Parameter	**parameter**	227
private JLabel	**type**	227
private TableCellEditor	**valueEditor**	227
private JComponent	**valueShower**	227

Constructor Summary	Page
ParametersPanel.Row(com.tilab.wade.performer.descriptors.Parameter parameter)	227

Method Summary		Page
JLabel	**getMode**()	227
JLabel	**getName**()	227
com.tilab.wade.performer.descriptors.Parameter	**getParameter**()	227
JLabel	**getType**()	227
JComponent	**getValue**()	227
TableCellEditor	**getValueEditor**()	227
void	**resetValue**()	227

Field Detail

`private com.tilab.wade.performer.descriptors.Parameter` **`parameter`**

`private JLabel` **`name`**

`private JLabel` **`type`**

`private JLabel` **`mode`**

`private JComponent` **`valueShower`**

`private TableCellEditor` **`valueEditor`**

Constructor Detail

`public` **`ParametersPanel.Row`**`(com.tilab.wade.performer.descriptors.Parameter parameter)`

Method Detail

`public void` **`resetValue`**`()`

`public com.tilab.wade.performer.descriptors.Parameter` **`getParameter`**`()`

`public JComponent` **`getValue`**`()`

`public TableCellEditor` **`getValueEditor`**`()`

`public JLabel` **`getName`**`()`

`public JLabel` **`getType`**`()`

`public JLabel` **`getMode`**`()`

Package monitoring

Interface Summary		Page
MonitoringWFHelper	This interface allow agents to interact directly with the MonitoringWF Service.	228
MonitoringWFSlice		234

Class Summary		Page
MonitoringWFProxy	This is a class whose instances are proxies to a remote slice.	229
MonitoringWFService	A kernel service used to semantically register all messages that are exchanged among agent to the database.	229
MonitoringWFService.MonitoringWFHelperImpl		232
MonitoringWFService.MonitoringWFSliceImpl		232
MonitoringWFService.OutgoingMontioringWFFilter	The filters do the actual work for a service.	233

Interface MonitoringWFHelper

monitoring

All Superinterfaces:

jade.core.ServiceHelper

All Known Implementing Classes:

MonitoringWFService.MonitoringWFHelperImpl

```
public interface MonitoringWFHelper
extends jade.core.ServiceHelper
```

This interface allow agents to interact directly with the MonitoringWF Service.

Author:

Pavlos Delias

Method Summary		Page
void	**setProcessId**(int id)	228

Methods inherited from interface jade.core.ServiceHelper
init

Method Detail

```
void setProcessId(int id)
```

Class MonitoringWFProxy

monitoring

```
java.lang.Object
  └ jade.core.SliceProxy
      └ monitoring.MonitoringWFProxy
```

All Implemented Interfaces:

MonitoringWFSlice, jade.util.leap.Serializable, Serializable, jade.core.Service.Slice

---

```
public class MonitoringWFProxy
extends jade.core.SliceProxy
implements MonitoringWFSlice
```

This is a class whose instances are proxies to a remote slice. When the `MonitoringWFService` needs to interact with a slice on a remote node it first retrieves a proxy to that slice and then invokes the required methods. The proxy has the main purpose of converting method calls into proper horizontal commands that will be sent to the remote slice.

Author:

Pavlos Delias

Fields inherited from interface monitoring.MonitoringWFSlice
`H_MONITORMESSAGE`

Constructor Summary	Page
`MonitoringWFProxy()`	229

Method Summary		Page
`void`	`monitorMessage(jade.lang.acl.ACLMessage msg)`	229

Methods inherited from class jade.core.SliceProxy
`getNode, getService, serve, setNode`

Constructor Detail

```
public MonitoringWFProxy()
```

Method Detail

```
public void monitorMessage(jade.lang.acl.ACLMessage msg)
                    throws jade.core.IMTPException
```

Specified by:

`monitorMessage` in interface `MonitoringWFSlice`

Throws:

`jade.core.IMTPException`

Class MonitoringWFService

monitoring

```
java.lang.Object
  └ jade.core.BaseService
      └ monitoring.MonitoringWFService
```

All Implemented Interfaces:

jade.core.Service

```
public class MonitoringWFService
extends jade.core.BaseService
```

A kernel service used to semantically register all messages that are exchanged among agent to the database.

Author:

Pavlos Delias

Nested Class Summary

	Nested Class Summary	Page
class	**MonitoringWFService.MonitoringWFHelperImpl**	232
class	**MonitoringWFService.MonitoringWFSliceImpl**	232
class	**MonitoringWFService.OutgoingMontioringWFFilter** The filters do the actual work for a service.	233

Nested classes/interfaces inherited from interface jade.core.Service

`Service.Slice, Service.SliceProxy`

Field Summary

	Field Summary	Page
static String	**APPLICATION_RUN**	231
private String	**applicationRun**	231
private jade.core.ServiceHelper	**helper**	231
private jade.core.Service.Slice	**localSlice**	231
static String	**NAME**	231
private jade.core.Filter	**outFilter**	231
private int	**processId**	231
static String	**VERBOSE**	231
private boolean	**verbose**	231

Fields inherited from class jade.core.BaseService

`MAIN_SLICE, myFinder, myLogger, THIS_SLICE`

Fields inherited from interface jade.core.Service

`ADOPTED_NODE, DEAD_NODE, DEAD_PLATFORM_MANAGER, DEAD_REPLICA, DEAD_SLICE, NEW_NODE, NEW_REPLICA, NEW_SLICE, REATTACHED, RECONNECTED`

Constructor Summary

Constructor Summary	Page
MonitoringWFService()	231

Method Summary

	Method Summary	Page
void	**boot**(jade.core.Profile p)	231
jade.core.Filter	**getCommandFilter**(boolean direction)	231
jade.core.ServiceHelper	**getHelper**(jade.core.Agent a)	231
Class<MonitoringWFSlice>	**getHorizontalInterface**()	231
jade.core.Service.Slice	**getLocalSlice**()	231
String	**getName**()	231
int	**getProcessId**()	232

void	**insertMSG2DB**(jade.lang.acl.ACLMessage msg, jade.core.AID receiverAID)	232
void	**setMyProcessId**(int processId)	232

Methods inherited from class jade.core.BaseService
addAlias, clearCachedSlice, createInvokator, dump, getAllSlices, getAMSBehaviour, getCommandSink, getFreshSlice, getIMTPManager, getLocalNode, getNumberOfSlices, getOwnedCommands, getSlice, init, lookupAlias, shutdown, stringifySlice, submit

Field Detail

public static final String **NAME**

public static final String **VERBOSE**

public static final String **APPLICATION_RUN**

private boolean **verbose**

private int **processId**

private String **applicationRun**

private jade.core.Filter **outFilter**

private jade.core.Service.Slice **localSlice**

private jade.core.ServiceHelper **helper**

Constructor Detail

public **MonitoringWFService**()

Method Detail

public String **getName**()

Specified by:

getName in interface jade.core.Service

---

public void **boot**(jade.core.Profile p)
throws jade.core.ServiceException

Specified by:

boot in interface jade.core.Service

Overrides:

boot in class jade.core.BaseService

Throws:

jade.core.ServiceException

---

public jade.core.Filter **getCommandFilter**(boolean direction)

Specified by:

getCommandFilter in interface jade.core.Service

Overrides:

getCommandFilter in class jade.core.BaseService

---

public jade.core.ServiceHelper **getHelper**(jade.core.Agent a)

Specified by:

getHelper in interface jade.core.Service

Overrides:

getHelper in class jade.core.BaseService

---

public Class<MonitoringWFSlice> **getHorizontalInterface**()

Specified by:

getHorizontalInterface in interface jade.core.Service

Overrides:

getHorizontalInterface in class jade.core.BaseService

---

public jade.core.Service.Slice **getLocalSlice**()

Specified by:

getLocalSlice in interface jade.core.Service

Overrides:

getLocalSlice in class jade.core.BaseService

```
public void setMyProcessId(int processId)
```

```
public int getProcessId()
```

```
public void insertMSG2DB(jade.lang.acl.ACLMessage msg,
                         jade.core.AID receiverAID)
```

Class MonitoringWFService.MonitoringWFHelperImpl

monitoring

```
java.lang.Object
  └ monitoring.MonitoringWFService.MonitoringWFHelperImpl
```

All Implemented Interfaces:

MonitoringWFHelper, jade.core.ServiceHelper

Enclosing class:

MonitoringWFService

```
public class MonitoringWFService.MonitoringWFHelperImpl
extends Object
implements MonitoringWFHelper
```

Constructor Summary	*Page*
MonitoringWFService.MonitoringWFHelperImpl()	*232*

Method Summary		*Page*
void	**init**(jade.core.Agent a)	*232*
void	**setProcessId**(int id)	*232*

Constructor Detail

```
public MonitoringWFService.MonitoringWFHelperImpl()
```

Method Detail

```
public void setProcessId(int id)
```

Specified by:

`setProcessId` in interface `MonitoringWFHelper`

```
public void init(jade.core.Agent a)
```

Specified by:

`init` in interface `jade.core.ServiceHelper`

Class MonitoringWFService.MonitoringWFSliceImpl

monitoring

```
java.lang.Object
  └ monitoring.MonitoringWFService.MonitoringWFSliceImpl
```

All Implemented Interfaces:

jade.util.leap.Serializable, Serializable, jade.core.Service.Slice

Enclosing class:

MonitoringWFService

```
public class MonitoringWFService.MonitoringWFSliceImpl
extends Object
implements jade.core.Service.Slice
```

Constructor Summary	*Page*
MonitoringWFService.MonitoringWFSliceImpl()	233

Method Summary		*Page*
jade.core.Node	**getNode**()	233
jade.core.Service	**getService**()	233
jade.core.VerticalCommand	**serve**(jade.core.HorizontalCommand cmd)	233

Constructor Detail

`public MonitoringWFService.MonitoringWFSliceImpl()`

Method Detail

```
public jade.core.Node getNode()
                       throws jade.core.ServiceException
```

Specified by:

`getNode` in interface `jade.core.Service.Slice`

Throws:

`jade.core.ServiceException`

`public jade.core.Service getService()`

Specified by:

`getService` in interface `jade.core.Service.Slice`

`public jade.core.VerticalCommand serve(jade.core.HorizontalCommand cmd)`

Specified by:

`serve` in interface `jade.core.Service.Slice`

Class MonitoringWFService.OutgoingMontioringWFFilter

monitoring

```
java.lang.Object
  └ jade.core.Filter
      └ monitoring.MonitoringWFService.OutgoingMontioringWFFilter
```

Enclosing class:

MonitoringWFService

```
public class MonitoringWFService.OutgoingMontioringWFFilter
extends jade.core.Filter
```

The filters do the actual work for a service. This one check if the VerticalCommand is the one that the SErvice is supposed to serve, and if yes, it sends the message to the mainSlice `MonitoringWFSlice` to store it in the database

Author:

Pavlos Delias

Fields inherited from class jade.core.Filter
FIRST, INCOMING, LAST, OUTGOING

Constructor Summary	Page
MonitoringWFService.OutgoingMontioringWFFilter()	234

Method Summary		Page
boolean	**accept**(jade.core.VerticalCommand cmd)	234

Methods inherited from class jade.core.Filter
getPreferredPosition, isBlocking, isSkipping, postProcess, setBlocking, setPreferredPosition, setSkipping

Constructor Detail

`public` **`MonitoringWFService.OutgoingMontioringWFFilter`**`()`

Method Detail

`public boolean` **`accept`**`(jade.core.VerticalCommand cmd)`

Overrides:

`accept` in class `jade.core.Filter`

Interface MonitoringWFSlice

monitoring

All Superinterfaces:

jade.util.leap.Serializable, Serializable, jade.core.Service.Slice

All Known Implementing Classes:

MonitoringWFProxy

```
public interface MonitoringWFSlice
extends jade.core.Service.Slice
```

Field Summary		Page
String	**H_MONITORMESSAGE**	234

Method Summary		Page
void	**monitorMessage**(jade.lang.acl.ACLMessage msg)	234

Methods inherited from interface jade.core.Service.Slice
getNode, getService, serve

Field Detail

`public static final String` **`H_MONITORMESSAGE`**

Method Detail

```
void monitorMessage(jade.lang.acl.ACLMessage msg)
             throws jade.core.IMTPException
```

Throws:

`jade.core.IMTPException`

Package ontology

Interface Summary		Page
ContactCenterVocabulary		238

Class Summary		Page
AddWorklist		235
ContactCenterOntology		236
Read		240
ReceiveMails		241
RequestsOf		241
SendMailBatch		242
SetProcess		243
Todo		244

Class AddWorklist

ontology

```
java.lang.Object
  └ ontology.AddWorklist
```

All Implemented Interfaces:

jade.content.AgentAction, jade.content.Concept, jade.content.ContentElement, jade.util.leap.Serializable, Serializable, jade.content.Term

```
public class AddWorklist
extends Object
implements jade.content.AgentAction
```

Field Summary		Page
private jade.core.AID	agent	235
private Worklist	list	235

Constructor Summary	Page
AddWorklist()	236

Method Summary		Page
jade.core.AID	getAgent()	236
Worklist	getList()	236
void	setAgent(jade.core.AID agent)	236
void	setList(Worklist list)	236

Field Detail

`private jade.core.AID agent`

`private Worklist list`

Constructor Detail

`public AddWorklist()`

Method Detail

`public void setAgent(jade.core.AID agent)`

`public jade.core.AID getAgent()`

`public void setList(Worklist list)`

`public Worklist getList()`

Class ContactCenterOntology

ontology

```
java.lang.Object
  └ jade.content.onto.Ontology
      └ ontology.ContactCenterOntology
```

All Implemented Interfaces:

ContactCenterVocabulary, jade.util.leap.Serializable, Serializable

```
public class ContactCenterOntology
extends jade.content.onto.Ontology
implements ContactCenterVocabulary
```

Field Summary		Page
private static jade.content.onto.Ontology	**theInstance** The singleton instance of this ontology	237

Fields inherited from interface ontology.ContactCenterVocabulary

ADD_WORKLIST, ADD_WORKLIST_2AGENT, ADD_WORKLIST_LIST, APPLICATION_RUN, CONVERSATION_ID, IN_REPLY_TO, MAIL, MAIL_BATCH, MAIL_BATCH_FILE, MAIL_BATCH_ITEMS, MAIL_CONTENT, MAIL_DURATION, MAIL_FT, MAIL_ST, MAIL_TYPE, ONTOLOGY, ONTOLOGY_NAME, PERFORMATIVE, PERFORMATIVE_NAME, PROCESS_ID, READ, READ_FILE, RECEIVE_MAILS, RECEIVE_MAILS_PASS, RECEIVE_MAILS_SERVER, RECEIVE_MAILS_USER, RECEIVER, RECEIVER_AGENT, REQUESTS_OF, REQUESTS_OF_AGENT, REQUESTS_TO, REQUESTS_TO_AGENT, SEND_MAIL_BATCH, SEND_MAIL_BATCH_FILENAME, SEND_MAIL_BATCH_ITEMS, SEND_MAIL_BATCH_TO_AGENT, SENDER, SENDER_NAME, SENDER_TYPE, SET_PROCESS, SET_PROCESS_ID, TASK, TASK_NAME, TASK_ST, TIMESTAMP, TODO, TODO_ITEM, WORKLIST, WORKLIST_FILE, WORKLIST_TASKS

Constructor Summary	Page
ContactCenterOntology()	237

Method Summary		Page
private void	**createAllActions**()	237
private void	**createAllConcepts**()	237
private void	**defineAllConcepts**()	237
private void	**defineMailBatchConcept**()	237
private void	**defineMailConcept**()	237
private void	**definePerformativeConcept**()	237
private void	**defineReceiverConcept**()	237
private void	**defineSenderConcept**()	237
private void	**defineTaskConcept**()	238
private void	**defineWorklistConcept**()	238
static jade.content.onto.Ontology	**getInstance**()	237

Methods inherited from class jade.content.onto.Ontology
`add, add, checkIsTerm, createConceptSlotFunction, fromObject, fromObject, getActionNames, getClassForElement, getConceptNames, getIntrospector, getName, getOwnActionNames, getOwnConceptNames, getOwnPredicateNames, getPredicateNames, getSchema, getSchema, toObject, toObject, toString, useConceptSlotsAsFunctions`

Field Detail

`private static final jade.content.onto.Ontology` **`theInstance`**

The singleton instance of this ontology

Constructor Detail

`public` **`ContactCenterOntology`**`()`

Method Detail

`public static final jade.content.onto.Ontology` **`getInstance`**`()`

```
private void createAllConcepts()
                        throws jade.content.onto.OntologyException
```

Throws:

`jade.content.onto.OntologyException`

```
private void createAllActions()
                       throws jade.content.onto.OntologyException
```

Throws:

`jade.content.onto.OntologyException`

```
private void defineAllConcepts()
                        throws jade.content.onto.OntologyException
```

Throws:

`jade.content.onto.OntologyException`

```
private void defineSenderConcept()
                          throws jade.content.onto.OntologyException
```

Throws:

`jade.content.onto.OntologyException`

```
private void defineReceiverConcept()
                            throws jade.content.onto.OntologyException
```

Throws:

`jade.content.onto.OntologyException`

```
private void definePerformativeConcept()
                                throws jade.content.onto.OntologyException
```

Throws:

`jade.content.onto.OntologyException`

```
private void defineMailConcept()
                        throws jade.content.onto.OntologyException
```

Throws:

`jade.content.onto.OntologyException`

```
private void defineMailBatchConcept()
                             throws jade.content.onto.OntologyException
```

Throws:

`jade.content.onto.OntologyException`

```
private void defineTaskConcept()
                        throws jade.content.onto.OntologyException
```

Throws:

`jade.content.onto.OntologyException`

```
private void defineWorklistConcept()
                            throws jade.content.onto.OntologyException
```

Throws:

`jade.content.onto.OntologyException`

Interface ContactCenterVocabulary

ontology

All Known Implementing Classes:

ContactCenterOntology

```
public interface ContactCenterVocabulary
```

Field Summary		Page
String	ADD_WORKLIST	239
String	ADD_WORKLIST_2AGENT	239
String	ADD_WORKLIST_LIST	239
String	APPLICATION_RUN	239
String	CONVERSATION_ID	239
String	IN_REPLY_TO	239
String	MAIL	239
String	MAIL_BATCH	239
String	MAIL_BATCH_FILE	239
String	MAIL_BATCH_ITEMS	239
String	MAIL_CONTENT	239
String	MAIL_DURATION	239
String	MAIL_FT	239
String	MAIL_ST	239
String	MAIL_TYPE	239
String	ONTOLOGY	239
String	ONTOLOGY_NAME	239
String	PERFORMATIVE	239
String	PERFORMATIVE_NAME	239
String	PROCESS_ID	239
String	READ	240
String	READ_FILE	240
String	RECEIVE_MAILS	240
String	RECEIVE_MAILS_PASS	240
String	RECEIVE_MAILS_SERVER	240
String	RECEIVE_MAILS_USER	240
String	RECEIVER	239
String	RECEIVER_AGENT	239
String	REQUESTS_OF	240
String	REQUESTS_OF_AGENT	240

String	REQUESTS_TO	240
String	REQUESTS_TO_AGENT	240
String	SEND_MAIL_BATCH	240
String	SEND_MAIL_BATCH_FILENAME	240
String	SEND_MAIL_BATCH_ITEMS	240
String	SEND_MAIL_BATCH_TO_AGENT	240
String	SENDER	239
String	SENDER_NAME	239
String	SENDER_TYPE	239
String	SET_PROCESS	240
String	SET_PROCESS_ID	240
String	TASK	239
String	TASK_NAME	239
String	TASK_ST	239
String	TIMESTAMP	239
String	TODO	240
String	TODO_ITEM	240
String	WORKLIST	239
String	WORKLIST_FILE	239
String	WORKLIST_TASKS	239

Field Detail

public static final String **ONTOLOGY_NAME**

public static final String **SENDER**

public static final String **RECEIVER**

public static final String **SENDER_NAME**

public static final String **SENDER_TYPE**

public static final String **RECEIVER_AGENT**

public static final String **PERFORMATIVE**

public static final String **PERFORMATIVE_NAME**

public static final String **TIMESTAMP**

public static final String **APPLICATION_RUN**

public static final String **PROCESS_ID**

public static final String **IN_REPLY_TO**

public static final String **ONTOLOGY**

public static final String **CONVERSATION_ID**

public static final String **MAIL**

public static final String **MAIL_TYPE**

public static final String **MAIL_ST**

public static final String **MAIL_FT**

public static final String **MAIL_DURATION**

public static final String **MAIL_CONTENT**

public static final String **MAIL_BATCH**

public static final String **MAIL_BATCH_ITEMS**

public static final String **MAIL_BATCH_FILE**

public static final String **TASK**

public static final String **TASK_NAME**

public static final String **TASK_ST**

public static final String **WORKLIST**

public static final String **WORKLIST_FILE**

public static final String **WORKLIST_TASKS**

public static final String **ADD_WORKLIST**

public static final String **ADD_WORKLIST_2AGENT**

public static final String **ADD_WORKLIST_LIST**

```
public static final String TODO
public static final String TODO_ITEM
public static final String REQUESTS_OF
public static final String REQUESTS_OF_AGENT
public static final String REQUESTS_TO
public static final String REQUESTS_TO_AGENT
public static final String RECEIVE_MAILS
public static final String RECEIVE_MAILS_USER
public static final String RECEIVE_MAILS_PASS
public static final String RECEIVE_MAILS_SERVER
public static final String SEND_MAIL_BATCH
public static final String SEND_MAIL_BATCH_TO_AGENT
public static final String SEND_MAIL_BATCH_FILENAME
public static final String SEND_MAIL_BATCH_ITEMS
public static final String READ
public static final String READ_FILE
public static final String SET_PROCESS
public static final String SET_PROCESS_ID
```

Class Read

ontology

```
java.lang.Object
  └ ontology.Read
```

All Implemented Interfaces:

jade.content.AgentAction, jade.content.Concept, jade.content.ContentElement, jade.util.leap.Serializable, Serializable, jade.content.Term

```
public class Read
extends Object
implements jade.content.AgentAction
```

Field Summary		Page
private String	file	240

Constructor Summary	Page
Read()	240

Method Summary		Page
String	getFile()	240
void	setFile(String file)	240

Field Detail

```
private String file
```

Constructor Detail

```
public Read()
```

Method Detail

```
public void setFile(String file)
public String getFile()
```

Class ReceiveMails

ontology

```
java.lang.Object
  └ ontology.ReceiveMails
```

All Implemented Interfaces:

jade.content.AgentAction, jade.content.Concept, jade.content.ContentElement, jade.util.leap.Serializable, Serializable, jade.content.Term

```
public class ReceiveMails
extends Object
implements jade.content.AgentAction
```

Field Summary		Page
private String	password	241
private String	server	241
private String	user	241

Constructor Summary	Page
ReceiveMails()	241

Method Summary		Page
String	getPassword()	241
String	getServer()	241
String	getUser()	241
void	setPassword(String password)	241
void	setServer(String server)	241
void	setUser(String user)	241

Field Detail

private String **user**

private String **password**

private String **server**

Constructor Detail

public **ReceiveMails**()

Method Detail

public void **setUser**(String user)

public String **getUser**()

public void **setPassword**(String password)

public String **getPassword**()

public void **setServer**(String server)

public String **getServer**()

Class RequestsOf

ontology

```
java.lang.Object
  └ ontology.RequestsOf
```

All Implemented Interfaces:

jade.content.AgentAction, jade.content.Concept, jade.content.ContentElement, jade.util.leap.Serializable, Serializable, jade.content.Term

```
public class RequestsOf
extends Object
implements jade.content.AgentAction
```

Field Summary		Page
private jade.core.AID	**agent**	242

Constructor Summary	Page
RequestsOf()	242

Method Summary		Page
jade.core.AID	**getAgent**()	242
void	**setAgent**(jade.core.AID agent)	242

Field Detail

`private jade.core.AID agent`

Constructor Detail

`public RequestsOf()`

Method Detail

`public void setAgent(jade.core.AID agent)`

`public jade.core.AID getAgent()`

Class SendMailBatch

ontology

```
java.lang.Object
  └ ontology.SendMailBatch
```

All Implemented Interfaces:

jade.content.AgentAction, jade.content.Concept, jade.content.ContentElement, jade.util.leap.Serializable, Serializable, jade.content.Term

```
public class SendMailBatch
extends Object
implements jade.content.AgentAction
```

Field Summary		Page
private String	**fileName**	243
private jade.core.AID	**toAgent**	243

Constructor Summary	Page
SendMailBatch()	243

Method Summary		Page
String	**getFileName**()	243
jade.core.AID	**getToAgent**()	243
void	**setFileName**(String fileName)	243
void	**setToAgent**(jade.core.AID toAgent)	243

Field Detail

private jade.core.AID **toAgent**

private String **fileName**

Constructor Detail

public **SendMailBatch**()

Method Detail

public void **setToAgent**(jade.core.AID toAgent)

public jade.core.AID **getToAgent**()

public void **setFileName**(String fileName)

public String **getFileName**()

Class SetProcess

ontology

```
java.lang.Object
  └─ ontology.SetProcess
```

All Implemented Interfaces:

jade.content.AgentAction, jade.content.Concept, jade.content.ContentElement, jade.util.leap.Serializable, Serializable, jade.content.Term

```
public class SetProcess
extends Object
implements jade.content.AgentAction
```

Field Summary		Page
private int	**id**	243

Constructor Summary	Page
SetProcess()	243

Method Summary		Page
int	**getId**()	243
void	**setId**(int id)	243

Field Detail

private int **id**

Constructor Detail

public **SetProcess**()

Method Detail

public void **setId**(int id)

public int **getId**()

Class Todo

ontology

```
java.lang.Object
  └ ontology.Todo
```

All Implemented Interfaces:

jade.content.AgentAction, jade.content.Concept, jade.content.ContentElement, jade.util.leap.Serializable, Serializable, jade.content.Term

```
public class Todo
extends Object
implements jade.content.AgentAction
```

Field Summary		Page
private String	item	244

Constructor Summary	Page
Todo()	244

Method Summary		Page
String	getItem()	244
void	setItem(String item)	244

Field Detail

`private String item`

Constructor Detail

`public Todo()`

Method Detail

`public String getItem()`

`public void setItem(String item)`

Package ontology.beans

Class Summary		Page
Mail		245
MailBatch		247
Performative		248
Receiver		249
Sender		249
Task		250
Worklist		251

Enum Summary		Page
Mail.MailType		246

Class Mail

ontology.beans

```
java.lang.Object
  └ ontology.beans.Mail
```

All Implemented Interfaces:

jade.content.Concept, jade.util.leap.Serializable, Serializable, jade.content.Term

Direct Known Subclasses:

MailBatch

```
public class Mail
extends Object
implements jade.content.Concept
```

Nested Class Summary		Page
static enum	Mail.MailType	246

Field Summary		Page
private String	content	246
private long	duration	246
private String	finishTime	246
private String	startTime	246
private String	type	246

Constructor Summary	Page
Mail()	246

Method Summary		Page
String	getContent()	246
long	getDuration()	246
String	getFinishTime()	246
String	getStartTime()	246
String	getType()	246
void	setContent(String content)	246
void	setDuration(long duration)	246
void	setFinishTime(String finishTime)	246
void	setStartTime(String startTime)	246
void	setType(String type)	246

Field Detail

private String **type**

private String **startTime**

private String **finishTime**

private long **duration**

private String **content**

Constructor Detail

public **Mail**()

Method Detail

public void **setType**(String type)

public String **getType**()

public void **setStartTime**(String startTime)

public String **getStartTime**()

public void **setFinishTime**(String finishTime)

public String **getFinishTime**()

public void **setContent**(String content)

public String **getContent**()

public void **setDuration**(long duration)

public long **getDuration**()

Enum Mail.MailType

ontology.beans

```
java.lang.Object
  └ java.lang.Enum<Mail.MailType>
      └ ontology.beans.Mail.MailType
```

All Implemented Interfaces:

Comparable<Mail.MailType>, Serializable

Enclosing class:

Mail

```
public static enum Mail.MailType
extends Enum<Mail.MailType>
```

Enum Constant Summary	Page
ERROR	247
GENERAL	247

INSTALLATION	247
SPECS	247
TROUBLESHOOTING	247
WARRANTY	247

Constructor Summary		Page
private	**Mail.MailType**()	247

Method Summary		Page
static Mail.MailType	**valueOf**(String name)	247
static Mail.MailType[]	**values**()	247

Enum Constant Detail

`public static final Mail.MailType` **WARRANTY**

`public static final Mail.MailType` **INSTALLATION**

`public static final Mail.MailType` **TROUBLESHOOTING**

`public static final Mail.MailType` **ERROR**

`public static final Mail.MailType` **SPECS**

`public static final Mail.MailType` **GENERAL**

Constructor Detail

`private` **Mail.MailType**()

Method Detail

`public static Mail.MailType[]` **values**()

`public static Mail.MailType` **valueOf**(String name)

Class MailBatch

ontology.beans

```
java.lang.Object
  └ ontology.beans.Mail
      └ ontology.beans.MailBatch
```

All Implemented Interfaces:

jade.content.Concept, jade.util.leap.Serializable, Serializable, jade.content.Term

```
public class MailBatch
extends Mail
```

Nested classes/interfaces inherited from class ontology.beans.Mail
Mail.MailType

Field Summary		Page
private String	**file**	248
private jade.util.leap.List	**items**	248

Constructor Summary	Page
MailBatch()	248

Method Summary		Page
String	**getFile**()	248
jade.util.leap.List	**getItems**()	248
void	**setFile**(String file)	248
void	**setItems**(jade.util.leap.List items)	248

Methods inherited from class ontology.beans.Mail
getContent, getDuration, getFinishTime, getStartTime, getType, setContent, setDuration, setFinishTime, setStartTime, setType

Field Detail

`private String` **`file`**

`private jade.util.leap.List` **`items`**

Constructor Detail

`public` **`MailBatch`**`()`

Method Detail

`public void` **`setFile`**`(String file)`

`public String` **`getFile`**`()`

`public void` **`setItems`**`(jade.util.leap.List items)`

`public jade.util.leap.List` **`getItems`**`()`

Class Performative

ontology.beans

```
java.lang.Object
  └ ontology.beans.Performative
```

All Implemented Interfaces:

jade.content.Concept, jade.util.leap.Serializable, Serializable, jade.content.Term

```
public class Performative
extends Object
implements jade.content.Concept
```

Field Summary		Page
private String	**name**	248

Constructor Summary	Page
Performative()	248

Method Summary		Page
String	**getName**()	249
void	**setName**(String name)	249

Field Detail

`private String` **`name`**

Constructor Detail

`public` **`Performative`**`()`

Method Detail

`public void setName(String name)`

`public String getName()`

Class Receiver

ontology.beans

```
java.lang.Object
  └ ontology.beans.Receiver
```

All Implemented Interfaces:

jade.content.Concept, jade.util.leap.Serializable, Serializable, jade.content.Term

```
public class Receiver
extends Object
implements jade.content.Concept
```

Field Summary		Page
private jade.core.AID	agent	249

Constructor Summary	Page
Receiver()	249

Method Summary		Page
jade.core.AID	getReceiverAgent()	249
void	setReceiverAgent(jade.core.AID receiverAgent)	249

Field Detail

`private jade.core.AID agent`

Constructor Detail

`public Receiver()`

Method Detail

`public void setReceiverAgent(jade.core.AID receiverAgent)`

`public jade.core.AID getReceiverAgent()`

Class Sender

ontology.beans

```
java.lang.Object
  └ ontology.beans.Sender
```

All Implemented Interfaces:

jade.content.Concept, jade.util.leap.Serializable, Serializable, jade.content.Term

```
public class Sender
extends Object
implements jade.content.Concept
```

Field Summary		Page
private String	name	250

private String	type	250

Constructor Summary	Page
Sender()	250

Method Summary		Page
String	getName()	250
String	getType()	250
void	setName(String name)	250
void	setType(String type)	250

Field Detail

private String **name**

private String **type**

Constructor Detail

public **Sender**()

Method Detail

public void **setName**(String name)

public String **getName**()

public void **setType**(String type)

public String **getType**()

Class Task

ontology.beans

```
java.lang.Object
  └ ontology.beans.Task
```

All Implemented Interfaces:

jade.content.Concept, jade.util.leap.Serializable, Serializable, jade.content.Term

Direct Known Subclasses:

Worklist

```
public class Task
extends Object
implements jade.content.Concept
```

Field Summary		Page
private String	name	251
private int	startTime	251

Constructor Summary	Page
Task()	251

Method Summary		Page
String	getName()	251
int	getStartTime()	251

void	**setName**(String name)	251
void	**setStartTime**(int startTime)	251

Field Detail

private String **name**

private int **startTime**

Constructor Detail

public **Task**()

Method Detail

public void **setName**(String name)

public String **getName**()

public void **setStartTime**(int startTime)

public int **getStartTime**()

Class Worklist

ontology.beans

```
java.lang.Object
  └ ontology.beans.Task
      └ ontology.beans.Worklist
```

All Implemented Interfaces:

jade.content.Concept, jade.util.leap.Serializable, Serializable, jade.content.Term

public class **Worklist**
extends Task

Field Summary		Page
private String	**file**	251
private jade.util.leap.List	**tasks**	251

Constructor Summary	Page
Worklist()	251

Method Summary		Page
String	**getFile**()	252
jade.util.leap.List	**getTasks**()	252
void	**setFile**(String file)	252
void	**setTasks**(jade.util.leap.List tasks)	252

Methods inherited from class ontology.beans.Task
getName, getStartTime, setName, setStartTime

Field Detail

private String **file**

private jade.util.leap.List **tasks**

Constructor Detail

public **Worklist**()

Method Detail

`public void` **`setFile`**`(String file)`

`public String` **`getFile`**`()`

`public void` **`setTasks`**`(jade.util.leap.List tasks)`

`public jade.util.leap.List` **`getTasks`**`()`

Package util

Class Summary		Page
CheckForMails	A Behavior to check periodically for mails.	253
ModifyDFDescription	A behaviour that is used to modify the agent's service description by adding a property	254
WordProcessing	Native interface to Word for Windows.	255

Class CheckForMails

util

```
java.lang.Object
  └ jade.core.behaviours.Behaviour
      └ jade.core.behaviours.SimpleBehaviour
          └ jade.core.behaviours.TickerBehaviour
              └ util.CheckForMails
```

All Implemented Interfaces:

jade.util.leap.Serializable, Serializable

```
public class CheckForMails
extends jade.core.behaviours.TickerBehaviour
```

A Behavior to check periodically for mails. It actually finds the reference to the Application Engine Agent and then it sends to him a request through the `createAppEngineRequest(AgentAction)` method.

Author:

Pavlos Delias

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
`Behaviour.RunnableChangedEvent`

Field Summary		Page
`jade.core.AID`	**`applicationEngine`**	254

Fields inherited from class jade.core.behaviours.Behaviour
`myAgent, myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING`

Constructor Summary	Page
`CheckForMails``(jade.core.Agent a, long period)`	254

Method Summary		Page
`jade.lang.acl.ACLMessage`	**`createAppEngineRequest`**`(jade.content.AgentAction action)` Creates a messages that requests from the Application Engine Agent to perform a `ReceiveMails` action.	254
`protected void`	**`onTick`**`()`	254

Methods inherited from class jade.core.behaviours.TickerBehaviour
`action, done, getTickCount, onStart, reset, reset, stop`

Methods inherited from class jade.core.behaviours.Behaviour
`actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, handle, handleBlockEvent, handleRestartEvent, isRunnable, onEnd, restart, root, setAgent, setBehaviourName, setDataStore, setExecutionState`

Field Detail

`jade.core.AID` **`applicationEngine`**

Constructor Detail

```
public CheckForMails(jade.core.Agent a,
                     long period)
```

Method Detail

```
protected void onTick()
```

Overrides:

`onTick` in class `jade.core.behaviours.TickerBehaviour`

```
synchronized jade.lang.acl.ACLMessage createAppEngineRequest(jade.content.AgentAction action)
                                                      throws jade.content.onto.OntologyExcepti
on,
                                                             jade.content.lang.Codec.CodecExce
ption
```

Creates a messages that requests from the Application Engine Agent to perform a `ReceiveMails` action.

Returns:

ACLMessage - a REQUEST message

Throws:

`jade.content.onto.OntologyException`
`jade.content.lang.Codec.CodecException`
`Codec.CodecException`

Class ModifyDFDescription

util

```
java.lang.Object
  └ jade.core.behaviours.Behaviour
      └ jade.core.behaviours.SimpleBehaviour
          └ jade.core.behaviours.OneShotBehaviour
              └ util.ModifyDFDescription
```

All Implemented Interfaces:

jade.util.leap.Serializable, Serializable

```
public class ModifyDFDescription
extends jade.core.behaviours.OneShotBehaviour
```

A behaviour that is used to modify the agent's service description by adding a property

Author:

Pavlos Delias

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
`Behaviour.RunnableChangedEvent`

Field Summary

		Page
private jade.core.Agent	**myAgent**	255
private jade.domain.FIPAAgentManagement.Property	**toAdd**	255

Fields inherited from class jade.core.behaviours.Behaviour

myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING

Constructor Summary

	Page
ModifyDFDescription(jade.core.Agent A, jade.domain.FIPAAgentManagement.Property p)	255

Method Summary

		Page
void	**action**()	255

Methods inherited from class jade.core.behaviours.OneShotBehaviour

done

Methods inherited from class jade.core.behaviours.SimpleBehaviour

reset

Methods inherited from class jade.core.behaviours.Behaviour

actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, handle, handleBlockEvent, handleRestartEvent, isRunnable, onEnd, onStart, restart, root, setAgent, setBehaviourName, setDataStore, setExecutionState

Field Detail

```
private jade.core.Agent myAgent
```

```
private jade.domain.FIPAAgentManagement.Property toAdd
```

Constructor Detail

```
public ModifyDFDescription(jade.core.Agent A,
                           jade.domain.FIPAAgentManagement.Property p)
```

Method Detail

```
public void action()
```

Overrides:

`action` in class `jade.core.behaviours.Behaviour`

Class WordProcessing

util

```
java.lang.Object
  └ util.WordProcessing
```

```
public class WordProcessing
extends Object
```

Native interface to Word for Windows. Simple version as presented via internet. To create a new document and to serve bookmarks by your java application code like this:

```
WordProcessing.createNewDocumentFromTemplate("SampleTemplate");
WordProcessing
```

```
   .typeTextAtBookmark("AddressLine1", "O'Reilly & Associated, Inc.");
 WordProcessing.typeTextAtBookmark("AddressLine2", "Mr Miller");
 WordProcessing.typeTextAtBookmark("AddressLine3", "101 Moris Street");
 WordProcessing.typeTextAtBookmark("AddressLine4", "Sebastopol, CA 95472-9902");
 WordProcessing.typeTextAtBookmark("Salutation", "Dear Mr Miller,");
 WordProcessing.exec();
```

Author:

Christoph Mueller

Field Summary		Page
private static boolean	**noteNotMatchingBookmarks**	257
private static File	**wordInput**	257
private static FileWriter	**wordInputWriter**	257

Constructor Summary	Page
WordProcessing()	257

Method Summary		Page
static void	**cancel**() Cancels the word processing.	258
static void	**changeDocumentDirectory**(String documentDirectory) Sets the document directory for future document saving.	258
static void	**closeDocument**() Closes the active document.	258
private static void	**closeWordInput**()	259
private static String	**code**(String stringToCode)	257
static void	**createNewDocumentFromTemplate**(String templateName) Creates a new document based on the desired template.	257
static void	**createNewDocumentFromTemplateToSelectByUser**() Triggers to the template selection dialog and creates a new document based on the chosen template.	257
static boolean	**exec**() Starts the execution of the above instructions.	258
static void	**executeMacro**(String macroName) Executes an arbitrary WordBasic macro.	258
private static boolean	**openWordInput**()	259
private static void	**output**(String key, String value)	259
static void	**printAndForget**() Prints the document on the standard printer and closes the document without saving.	258
static void	**printAndForget**(String printerName) Prints the document on the specified printer and closes the document without saving.	258
static void	**printToPrinterToSelectByUserAndForget**() Triggers to the printer selection dialog, prints the document on the selected printer and closes the document without saving.	258
static void	**quitApplication**() Quits the word processing application.	258

static void	**quitApplicationAfterWaiting**(int milliseconds) Quits the word processing application after a pause.	*258*
private static String	**replaceAll**(String stringToManipulate, String stringToReplace, String replaceString)	*258*
static void	**saveDocumentAs**(String documentName) Saves the active document using the indicated name (usually without extension).	*258*
static void	**saveDocumentAsAndClose**(String documentName) Saves the active document using the indicated name and closes it.	*258*
static void	**setNoteNotMatchingBookmarks**(boolean noteNotMatchingBookmarks) Set the warning flag about not matching bookmarks Decides whether the user shall be informed that the template didn't include certain bookmarks.	*257*
static void	**typeTextAtBookmark**(String bookmark, String textToType) Goes to the specified bookmark and types the desired text.	*257*
static void	**typeTextAtBookmark**(String bookmark, String[] linesToType) Goes to the specified bookmark and types the desired text with line feed.	*257*

Field Detail

`private static final boolean` **`noteNotMatchingBookmarks`**

`private static File` **`wordInput`**

`private static FileWriter` **`wordInputWriter`**

Constructor Detail

`public` **`WordProcessing`**`()`

Method Detail

`public static void` **`createNewDocumentFromTemplateToSelectByUser`**`()`

Triggers to the template selection dialog and creates a new document based on the chosen template.

`public static void` **`createNewDocumentFromTemplate`**`(String templateName)`

Creates a new document based on the desired template.

Parameters:

`templateName` - the name of the template to be used

`public static void` **`setNoteNotMatchingBookmarks`**`(boolean noteNotMatchingBookmarks)`

Set the warning flag about not matching bookmarks Decides whether the user shall be informed that the template didn't include certain bookmarks.

Parameters:

`noteNotMatchingBookmarks` - whether the user should be warned

```
public static void typeTextAtBookmark(String bookmark,
                                      String textToType)
```

Goes to the specified bookmark and types the desired text.

Parameters:

`bookmark` - the bookmark where text type starts
`textToType` - the text to be included

```
public static void typeTextAtBookmark(String bookmark,
                                      String[] linesToType)
```

Goes to the specified bookmark and types the desired text with line feed.

Parameters:

`bookmark` - the bookmark where text type starts
`linesToType` - the lines to be included

`private static String` **`code`**`(String stringToCode)`

```
private static synchronized String replaceAll(String stringToManipulate,
                                              String stringToReplace,
                                              String replaceString)
```

`public static void` **`changeDocumentDirectory`**`(String documentDirectory)`

Sets the document directory for future document saving.

Parameters:

`documentDirectory` - the name of the directory

`public static void` **`saveDocumentAs`**`(String documentName)`

Saves the active document using the indicated name (usually without extension).

Parameters:

`documentName` - the name of the document

`public static void` **`saveDocumentAsAndClose`**`(String documentName)`

Saves the active document using the indicated name and closes it.

Parameters:

`documentName` - the name of the document

`public static void` **`closeDocument`**`()`

Closes the active document.

`public static void` **`printAndForget`**`()`

Prints the document on the standard printer and closes the document without saving.

`public static void` **`printAndForget`**`(String printerName)`

Prints the document on the specified printer and closes the document without saving.

Parameters:

`printerName` - the name of the desired printer

`public static void` **`printToPrinterToSelectByUserAndForget`**`()`

Triggers to the printer selection dialog, prints the document on the selected printer and closes the document without saving.

`public static void` **`executeMacro`**`(String macroName)`

Executes an arbitrary WordBasic macro.

Parameters:

`macroName` - the name of the macro to be executed

`public static void` **`quitApplication`**`()`

Quits the word processing application.

`public static void` **`quitApplicationAfterWaiting`**`(int milliseconds)`

Quits the word processing application after a pause. This gives the word processing time to finish e.g. a print job. This avoids dialogs by the word processing system wether the print job is to stop

Parameters:

`milliseconds` - waiting time in milliseconds prior leaving application

---

`public static boolean` **`exec`**`()`

Starts the execution of the above instructions. (This stacking is particularly helpful at large numbers of standard letters.) Always use use this as the last method of a sequence.

`public static void` **`cancel`**`()`

Cancels the word processing.

```
private static void output(String key,
                           String value)
```

```
private static boolean openWordInput()
```

```
private static void closeWordInput()
```

Package util.objects

Class Summary		Page
ApplicationFile	An auxiliary method to facilitate file functions.	260
CustomerRecord	An supportive class to represent a Customer Record as a JAVA object.	261
Offer	A supportive class to represent an vendor's Offer as a JAVA object.	262

Class ApplicationFile

util.objects

```
java.lang.Object
  └ util.objects.ApplicationFile
```

```
public class ApplicationFile
extends Object
```

An auxiliary method to facilitate file functions.

Author:
Pavlos Delias

Field Summary		Page
private String	myPath	260

Constructor Summary	Page
ApplicationFile()	260

Method Summary		Page
String	getMyPath()	260
static String	returnEscapedPath(String in) This method accepts a filename as input and it returns the same filename with escaped characters.	260
void	setMyPath(String myPath)	260

Field Detail

`private String myPath`

Constructor Detail

`public ApplicationFile()`

Method Detail

`public static String returnEscapedPath(String in)`

This method accepts a filename as input and it returns the same filename with escaped characters.

Returns:
String - The filename containing the escaped characters for backslashes.

`public void setMyPath(String myPath)`

`public String getMyPath()`

Class CustomerRecord

util.objects

```
java.lang.Object
  └ util.objects.CustomerRecord
```

public class **CustomerRecord**
extends Object

An supportive class to represent a Customer Record as a JAVA object.

Author:
Pavlos Delias

Field Summary

Field Summary		Page
private int	channel	261
private int	ID	261
private String	name	261
private int	processingTime	261

Constructor Summary

Constructor Summary	Page
CustomerRecord()	261

Method Summary

Method Summary		Page
int	getChannel()	261
int	getID()	261
String	getName()	261
int	getProcessingTime()	261
void	setChannel(int channel)	261
void	setID(int ID)	261
void	setName(String name)	261
void	setProcessingTime(int processingTime)	262

Field Detail

private int **ID**

private String **name**

private int **channel**

private int **processingTime**

Constructor Detail

public **CustomerRecord**()

Method Detail

public int **getID**()

public void **setID**(int ID)

public int **getChannel**()

public void **setChannel**(int channel)

public String **getName**()

public void **setName**(String name)

public int **getProcessingTime**()

`public void setProcessingTime(int processingTime)`

Class Offer

util.objects

```
java.lang.Object
  └ util.objects.Offer
```

```
public class Offer
extends Object
```

A supportive class to represent an vendor's Offer as a JAVA object.

Author:
Pavlos Delias

Field Summary		Page
private MediaDecisionsGUI.MediaFormat	**format**	262
private int	**quantity**	262

Constructor Summary	Page
Offer(int q, MediaDecisionsGUI.MediaFormat f)	262
Offer(int q, String format)	262

Method Summary		Page
MediaDecisionsGUI.MediaFormat	**getFormat**()	262
int	**getQuantity**()	262
void	**setFormat**(MediaDecisionsGUI.MediaFormat format)	262
void	**setQuantity**(int quantity)	262
void	**setStringFormat**(String f)	262

Field Detail

`private int quantity`

`private MediaDecisionsGUI.MediaFormat format`

Constructor Detail

```
public Offer(int q,
             MediaDecisionsGUI.MediaFormat f)
```

```
public Offer(int q,
             String format)
```

Method Detail

`public void setQuantity(int quantity)`

`public int getQuantity()`

`public void setFormat(MediaDecisionsGUI.MediaFormat format)`

`public MediaDecisionsGUI.MediaFormat getFormat()`

`public void setStringFormat(String f)`

Package util.ws

Interface Summary		Page
CalculateVendorOffer		263
CalculateVendorOfferService		265
ContactCRM		269
ContactCRMService		271

Class Summary		Page
CalculateVendorOfferPortBindingStub		264
CalculateVendorOfferServiceDescriptor		266
CalculateVendorOfferServiceLocator		267
ContactCRMPortBindingStub		270
ContactCRMServiceDescriptor		272
ContactCRMServiceLocator		273
CrmResult		275
MediaFormat		277

Interface CalculateVendorOffer

util.ws

All Superinterfaces:

Remote

All Known Implementing Classes:

CalculateVendorOfferPortBindingStub

```
public interface CalculateVendorOffer
extends Remote
```

Method Summary		Page
double	**calculateOffer**(int quantity, MediaFormat format, int myStyle)	263

Method Detail

```
double calculateOffer(int quantity,
                      MediaFormat format,
                      int myStyle)
               throws RemoteException
```

Throws:

RemoteException

Class CalculateVendorOfferPortBindingStub

util.ws

```
java.lang.Object
  └ org.apache.axis.client.Stub
      └ util.ws.CalculateVendorOfferPortBindingStub
```

All Implemented Interfaces:

CalculateVendorOffer, Remote, Stub

```
public class CalculateVendorOfferPortBindingStub
extends org.apache.axis.client.Stub
implements CalculateVendorOffer
```

Field Summary		Page
static org.apache.axis.description.OperationDesc[]	**_operations**	264
private Vector	**cachedDeserFactories**	264
private Vector	**cachedSerClasses**	264
private Vector	**cachedSerFactories**	264
private Vector	**cachedSerQNames**	264

Fields inherited from class org.apache.axis.client.Stub
_call, cachedEndpoint, cachedPassword, cachedPortName, cachedProperties, cachedTimeout, cachedUsername, maintainSession, maintainSessionSet, service

Constructor Summary	Page
CalculateVendorOfferPortBindingStub()	265
CalculateVendorOfferPortBindingStub(URL endpointURL, Service service)	265
CalculateVendorOfferPortBindingStub(Service service)	265

Method Summary		Page
private static void	**_initOperationDesc1**()	265
double	**calculateOffer**(int quantity, MediaFormat format, int myStyle)	265
protected org.apache.axis.client.Call	**createCall**()	265

Methods inherited from class org.apache.axis.client.Stub
_createCall, _getCall, _getProperty, _getPropertyNames, _getService, _setProperty, addAttachment, clearAttachments, clearHeaders, extractAttachments, firstCall, getAttachments, getHeader, getHeaders, getPassword, getPortName, getResponseHeader, getResponseHeaders, getResponseHeaders, getTimeout, getUsername, removeProperty, setAttachments, setHeader, setHeader, setMaintainSession, setPassword, setPortName, setPortName, setRequestHeaders, setTimeout, setUsername

Field Detail

private Vector **cachedSerClasses**

private Vector **cachedSerQNames**

private Vector **cachedSerFactories**

private Vector **cachedDeserFactories**

static org.apache.axis.description.OperationDesc[] **_operations**

Constructor Detail

```
public CalculateVendorOfferPortBindingStub()
                                    throws org.apache.axis.AxisFault
```

```
public CalculateVendorOfferPortBindingStub(URL endpointURL,
                                           Service service)
                                    throws org.apache.axis.AxisFault
```

```
public CalculateVendorOfferPortBindingStub(Service service)
                                    throws org.apache.axis.AxisFault
```

Method Detail

```
private static void _initOperationDesc1()
```

```
protected org.apache.axis.client.Call createCall()
                                          throws RemoteException
```

Throws:

RemoteException

```
public double calculateOffer(int quantity,
                             MediaFormat format,
                             int myStyle)
                      throws RemoteException
```

Specified by:

calculateOffer in interface CalculateVendorOffer

Throws:

RemoteException

Interface CalculateVendorOfferService

util.ws

All Superinterfaces:

Service

All Known Implementing Classes:

CalculateVendorOfferServiceLocator

```
public interface CalculateVendorOfferService
extends Service
```

Method Summary		Page
CalculateVendorOffer	**getCalculateVendorOfferPort**()	265
CalculateVendorOffer	**getCalculateVendorOfferPort**(URL portAddress)	265
String	**getCalculateVendorOfferPortAddress**()	265

Method Detail

```
String getCalculateVendorOfferPortAddress()
```

```
CalculateVendorOffer getCalculateVendorOfferPort()
                                          throws ServiceException
```

Throws:

ServiceException

```
CalculateVendorOffer getCalculateVendorOfferPort(URL portAddress)
                                          throws ServiceException
```

Throws:

ServiceException

Class CalculateVendorOfferServiceDescriptor

util.ws

```
java.lang.Object
  └ com.tilab.wade.performer.descriptors.webservice.ServiceDescriptor
      └ util.ws.CalculateVendorOfferServiceDescriptor
```

public class **CalculateVendorOfferServiceDescriptor**
extends com.tilab.wade.performer.descriptors.webservice.ServiceDescriptor

Field Summary		Page
CalculateVendorOfferServiceLocator	**locator**	266
static String	**NAMESPACE**	266
static String	**SERVICE_NAME**	266

Fields inherited from class com.tilab.wade.performer.descriptors.webservice.ServiceDescriptor
portDescriptors, SERVICE_DESCRIPTOR_SUFFIX

Constructor Summary	Page
CalculateVendorOfferServiceDescriptor()	266

Method Summary		Page
Remote	**getService**()	266
String	**getServiceName**()	266
private void	**setcalculateOfferParameters**(jade.util.leap.List formalParams)	267
void	**setEndpointAddress**(String endpointAddress)	267

Methods inherited from class com.tilab.wade.performer.descriptors.webservice.ServiceDescriptor
addOperationDescriptor, addPortDescriptor, getOperationDescriptor, getOperationNames, getPortDescriptor, getPortNames, invoke

Field Detail

public static final String **SERVICE_NAME**

public static final String **NAMESPACE**

CalculateVendorOfferServiceLocator **locator**

Constructor Detail

public **CalculateVendorOfferServiceDescriptor**()

Method Detail

```
public Remote getService()
                  throws ServiceException
```

Overrides:

getService in class com.tilab.wade.performer.descriptors.webservice.ServiceDescriptor

Throws:

ServiceException

public String **getServiceName**()

Overrides:

getServiceName in class com.tilab.wade.performer.descriptors.webservice.ServiceDescriptor

```
public void setEndpointAddress(String endpointAddress)
```

Overrides:

`setEndpointAddress` in class `com.tilab.wade.performer.descriptors.webservice.ServiceDescriptor`

```
private void setcalculateOfferParameters(jade.util.leap.List formalParams)
```

Class CalculateVendorOfferServiceLocator

util.ws

```
java.lang.Object
  └ org.apache.axis.client.Service
      └ util.ws.CalculateVendorOfferServiceLocator
```

All Implemented Interfaces:

CalculateVendorOfferService, Referenceable, Serializable, Service

```
public class CalculateVendorOfferServiceLocator
extends org.apache.axis.client.Service
implements CalculateVendorOfferService
```

Nested classes/interfaces inherited from class org.apache.axis.client.Service
Service.HandlerRegistryImpl

Field Summary		Page
private String	**CalculateVendorOfferPort_address**	268
private String	**CalculateVendorOfferPortWSDDServiceName**	268
private HashSet	**ports**	268

Fields inherited from class org.apache.axis.client.Service
_call

Constructor Summary	Page
CalculateVendorOfferServiceLocator()	268
CalculateVendorOfferServiceLocator(String wsdlLoc, QName sName)	268
CalculateVendorOfferServiceLocator(org.apache.axis.EngineConfiguration config)	268

Method Summary		Page
CalculateVendorOffer	**getCalculateVendorOfferPort**()	268
CalculateVendorOffer	**getCalculateVendorOfferPort**(URL portAddress)	268
String	**getCalculateVendorOfferPortAddress**()	268
String	**getCalculateVendorOfferPortWSDDServiceName**()	268
Remote	**getPort**(Class serviceEndpointInterface) For the given interface, get the stub implementation.	268
Remote	**getPort**(QName portName, Class serviceEndpointInterface) For the given interface, get the stub implementation.	269
Iterator	**getPorts**()	269
QName	**getServiceName**()	269
void	**setCalculateVendorOfferPortEndpointAddress**(String address)	268
void	**setCalculateVendorOfferPortWSDDServiceName**(String name)	268

void	**setEndpointAddress**(String portName, String address) Set the endpoint address for the specified port name.	269
void	**setEndpointAddress**(QName portName, String address) Set the endpoint address for the specified port name.	269

Methods inherited from class org.apache.axis.client.Service
createCall, createCall, createCall, createCall, getAxisClient, getCacheWSDL, getCall, getCalls, getEngine, getEngineConfiguration, getHandlerRegistry, getMaintainSession, getPort, getReference, getTypeMappingRegistry, getWSDLDocumentLocation, getWSDLParser, getWSDLService, setCacheWSDL, setEngine, setEngineConfiguration, setMaintainSession, setTypeMappingRegistry, setTypeMappingVersion

Field Detail

private String **CalculateVendorOfferPort_address**

private String **CalculateVendorOfferPortWSDDServiceName**

private HashSet **ports**

Constructor Detail

public **CalculateVendorOfferServiceLocator**()

public **CalculateVendorOfferServiceLocator**(org.apache.axis.EngineConfiguration config)

```
public CalculateVendorOfferServiceLocator(String wsdlLoc,
                                          QName sName)
                                   throws ServiceException
```

Method Detail

public String **getCalculateVendorOfferPortAddress**()

Specified by:

getCalculateVendorOfferPortAddress in interface CalculateVendorOfferService

public String **getCalculateVendorOfferPortWSDDServiceName**()

public void **setCalculateVendorOfferPortWSDDServiceName**(String name)

```
public CalculateVendorOffer getCalculateVendorOfferPort()
                                                 throws ServiceException
```

Specified by:

getCalculateVendorOfferPort in interface CalculateVendorOfferService

Throws:

ServiceException

```
public CalculateVendorOffer getCalculateVendorOfferPort(URL portAddress)
                                                 throws ServiceException
```

Specified by:

getCalculateVendorOfferPort in interface CalculateVendorOfferService

Throws:

ServiceException

public void **setCalculateVendorOfferPortEndpointAddress**(String address)

```
public Remote getPort(Class serviceEndpointInterface)
               throws ServiceException
```

For the given interface, get the stub implementation. If this service has no port for the given interface, then ServiceException is thrown.

Specified by:

getPort in interface Service

Overrides:

getPort in class org.apache.axis.client.Service

Throws:

ServiceException

```
public Remote getPort(QName portName,
                      Class serviceEndpointInterface)
               throws ServiceException
```

For the given interface, get the stub implementation. If this service has no port for the given interface, then ServiceException is thrown.

Specified by:
`getPort` in interface `Service`

Overrides:
`getPort` in class `org.apache.axis.client.Service`

Throws:
`ServiceException`

```
public QName getServiceName()
```

Specified by:
`getServiceName` in interface `Service`

Overrides:
`getServiceName` in class `org.apache.axis.client.Service`

```
public Iterator getPorts()
```

Specified by:
`getPorts` in interface `Service`

Overrides:
`getPorts` in class `org.apache.axis.client.Service`

```
public void setEndpointAddress(String portName,
                               String address)
                        throws ServiceException
```

Set the endpoint address for the specified port name.

Throws:
`ServiceException`

```
public void setEndpointAddress(QName portName,
                               String address)
                        throws ServiceException
```

Set the endpoint address for the specified port name.

Throws:
`ServiceException`

Interface ContactCRM

util.ws

All Superinterfaces:
Remote

All Known Implementing Classes:
ContactCRMPortBindingStub

```
public interface ContactCRM
extends Remote
```

Method Summary		Page
`CrmResult`	**`getResult`**`(String customerName)`	*270*

Method Detail

```
CrmResult getResult(String customerName)
            throws RemoteException
```

Throws:

`RemoteException`

Class ContactCRMPortBindingStub

util.ws

```
java.lang.Object
  └ org.apache.axis.client.Stub
      └ util.ws.ContactCRMPortBindingStub
```

All Implemented Interfaces:

ContactCRM, Remote, Stub

```
public class ContactCRMPortBindingStub
extends org.apache.axis.client.Stub
implements ContactCRM
```

Field Summary		Page
static org.apache.axis.description.OperationDesc[]	**_operations**	271
private Vector	**cachedDeserFactories**	271
private Vector	**cachedSerClasses**	271
private Vector	**cachedSerFactories**	271
private Vector	**cachedSerQNames**	271

Fields inherited from class org.apache.axis.client.Stub

_call, cachedEndpoint, cachedPassword, cachedPortName, cachedProperties, cachedTimeout, cachedUsername, maintainSession, maintainSessionSet, service

Constructor Summary	Page
ContactCRMPortBindingStub()	271
ContactCRMPortBindingStub(URL endpointURL, Service service)	271
ContactCRMPortBindingStub(Service service)	271

Method Summary		Page
private static void	**_initOperationDesc1**()	271
protected org.apache.axis.client.Call	**createCall**()	271
CrmResult	**getResult**(String customerName)	271

Methods inherited from class org.apache.axis.client.Stub

_createCall, _getCall, _getProperty, _getPropertyNames, _getService, _setProperty, addAttachment, clearAttachments, clearHeaders, extractAttachments, firstCall, getAttachments, getHeader, getHeaders, getPassword, getPortName, getResponseHeader, getResponseHeaders, getResponseHeaders, getTimeout, getUsername, removeProperty, setAttachments, setHeader, setHeader, setMaintainSession, setPassword, setPortName, setPortName, setRequestHeaders, setTimeout, setUsername

Field Detail

```
private Vector cachedSerClasses
```
```
private Vector cachedSerQNames
```
```
private Vector cachedSerFactories
```
```
private Vector cachedDeserFactories
```
```
static org.apache.axis.description.OperationDesc[] _operations
```

Constructor Detail

```
public ContactCRMPortBindingStub()
                          throws org.apache.axis.AxisFault
```
```
public ContactCRMPortBindingStub(URL endpointURL,
                                 Service service)
                          throws org.apache.axis.AxisFault
```
```
public ContactCRMPortBindingStub(Service service)
                          throws org.apache.axis.AxisFault
```

Method Detail

```
private static void _initOperationDesc1()
```
```
protected org.apache.axis.client.Call createCall()
                                        throws RemoteException
```

Throws:
`RemoteException`

```
public CrmResult getResult(String customerName)
                    throws RemoteException
```

Specified by:
`getResult` in interface `ContactCRM`

Throws:
`RemoteException`

Interface ContactCRMService

util.ws

All Superinterfaces:
Service

All Known Implementing Classes:
ContactCRMServiceLocator

```
public interface ContactCRMService
extends Service
```

Method Summary		Page
ContactCRM	**getContactCRMPort**()	271
ContactCRM	**getContactCRMPort**(URL portAddress)	271
String	**getContactCRMPortAddress**()	271

Method Detail

```
String getContactCRMPortAddress()
```
```
ContactCRM getContactCRMPort()
                      throws ServiceException
```

Throws:
`ServiceException`

```
ContactCRM getContactCRMPort(URL portAddress)
                      throws ServiceException
```

Throws:
`ServiceException`

Class ContactCRMServiceDescriptor

[util.ws]

```
java.lang.Object
  └ com.tilab.wade.performer.descriptors.webservice.ServiceDescriptor
      └ util.ws.ContactCRMServiceDescriptor
```

```
public class ContactCRMServiceDescriptor
extends com.tilab.wade.performer.descriptors.webservice.ServiceDescriptor
```

Field Summary		Page
ContactCRMServiceLocator	**locator**	272
static String	**NAMESPACE**	272
static String	**SERVICE_NAME**	272

Fields inherited from class com.tilab.wade.performer.descriptors.webservice.ServiceDescriptor
portDescriptors, SERVICE_DESCRIPTOR_SUFFIX

Constructor Summary	Page
ContactCRMServiceDescriptor()	272

Method Summary		Page
Remote	**getService**()	272
String	**getServiceName**()	273
void	**setEndpointAddress**(String endpointAddress)	273
private void	**setgetResultParameters**(jade.util.leap.List formalParams)	273

Methods inherited from class com.tilab.wade.performer.descriptors.webservice.ServiceDescriptor
addOperationDescriptor, addPortDescriptor, getOperationDescriptor, getOperationNames, getPortDescriptor, getPortNames, invoke

Field Detail

`public static final String` **SERVICE_NAME**

`public static final String` **NAMESPACE**

`ContactCRMServiceLocator` **locator**

Constructor Detail

`public` **ContactCRMServiceDescriptor**()

Method Detail

```
public Remote getService()
                  throws ServiceException
```

Overrides:
`getService` in class
`com.tilab.wade.performer.descriptors.webservice.ServiceDescriptor`

Throws:
`ServiceException`

```
public String getServiceName()
```

Overrides:

`getServiceName` in class `com.tilab.wade.performer.descriptors.webservice.ServiceDescriptor`

```
public void setEndpointAddress(String endpointAddress)
```

Overrides:

`setEndpointAddress` in class `com.tilab.wade.performer.descriptors.webservice.ServiceDescriptor`

```
private void setgetResultParameters(jade.util.leap.List formalParams)
```

Class ContactCRMServiceLocator

util.ws

```
java.lang.Object
  └ org.apache.axis.client.Service
      └ util.ws.ContactCRMServiceLocator
```

All Implemented Interfaces:

ContactCRMService, Referenceable, Serializable, Service

```
public class ContactCRMServiceLocator
extends org.apache.axis.client.Service
implements ContactCRMService
```

Nested classes/interfaces inherited from class org.apache.axis.client.Service
`Service.HandlerRegistryImpl`

Field Summary		Page
private String	**ContactCRMPort_address**	274
private String	**ContactCRMPortWSDDServiceName**	274
private HashSet	**ports**	274

Fields inherited from class org.apache.axis.client.Service
`_call`

Constructor Summary	Page
ContactCRMServiceLocator()	274
ContactCRMServiceLocator(String wsdlLoc, QName sName)	274
ContactCRMServiceLocator(org.apache.axis.EngineConfiguration config)	274

Method Summary		Page
ContactCRM	**getContactCRMPort**()	274
ContactCRM	**getContactCRMPort**(URL portAddress)	274
String	**getContactCRMPortAddress**()	274
String	**getContactCRMPortWSDDServiceName**()	274
Remote	**getPort**(Class serviceEndpointInterface) For the given interface, get the stub implementation.	274
Remote	**getPort**(QName portName, Class serviceEndpointInterface) For the given interface, get the stub implementation.	275

Iterator	**getPorts**()	275
QName	**getServiceName**()	275
void	**setContactCRMPortEndpointAddress**(String address)	274
void	**setContactCRMPortWSDDServiceName**(String name)	274
void	**setEndpointAddress**(String portName, String address) Set the endpoint address for the specified port name.	275
void	**setEndpointAddress**(QName portName, String address) Set the endpoint address for the specified port name.	275

Methods inherited from class org.apache.axis.client.Service
createCall, createCall, createCall, createCall, getAxisClient, getCacheWSDL, getCall, getCalls, getEngine, getEngineConfiguration, getHandlerRegistry, getMaintainSession, getPort, getReference, getTypeMappingRegistry, getWSDLDocumentLocation, getWSDLParser, getWSDLService, setCacheWSDL, setEngine, setEngineConfiguration, setMaintainSession, setTypeMappingRegistry, setTypeMappingVersion

Field Detail

`private String` **`ContactCRMPort_address`**

`private String` **`ContactCRMPortWSDDServiceName`**

`private HashSet` **`ports`**

Constructor Detail

`public` **`ContactCRMServiceLocator`**`()`

`public` **`ContactCRMServiceLocator`**`(org.apache.axis.EngineConfiguration config)`

```
public ContactCRMServiceLocator(String wsdlLoc,
                                QName sName)
                         throws ServiceException
```

Method Detail

`public String` **`getContactCRMPortAddress`**`()`

Specified by:

getContactCRMPortAddress in interface ContactCRMService

`public String` **`getContactCRMPortWSDDServiceName`**`()`

`public void` **`setContactCRMPortWSDDServiceName`**`(String name)`

```
public ContactCRM getContactCRMPort()
                             throws ServiceException
```

Specified by:

getContactCRMPort in interface ContactCRMService

Throws:

ServiceException

```
public ContactCRM getContactCRMPort(URL portAddress)
                             throws ServiceException
```

Specified by:

getContactCRMPort in interface ContactCRMService

Throws:

ServiceException

`public void` **`setContactCRMPortEndpointAddress`**`(String address)`

```
public Remote getPort(Class serviceEndpointInterface)
               throws ServiceException
```

For the given interface, get the stub implementation. If this service has no port for the given interface, then ServiceException is thrown.

Specified by:

getPort in interface Service

Overrides:

getPort in class org.apache.axis.client.Service

Throws:
`ServiceException`

```
public Remote getPort(QName portName,
                      Class serviceEndpointInterface)
               throws ServiceException
```

For the given interface, get the stub implementation. If this service has no port for the given interface, then ServiceException is thrown.

Specified by:
`getPort` in interface `Service`

Overrides:
`getPort` in class `org.apache.axis.client.Service`

Throws:
`ServiceException`

```
public QName getServiceName()
```

Specified by:
`getServiceName` in interface `Service`

Overrides:
`getServiceName` in class `org.apache.axis.client.Service`

```
public Iterator getPorts()
```

Specified by:
`getPorts` in interface `Service`

Overrides:
`getPorts` in class `org.apache.axis.client.Service`

```
public void setEndpointAddress(String portName,
                               String address)
                        throws ServiceException
```

Set the endpoint address for the specified port name.

Throws:
`ServiceException`

```
public void setEndpointAddress(QName portName,
                               String address)
                        throws ServiceException
```

Set the endpoint address for the specified port name.

Throws:
`ServiceException`

Class CrmResult

util.ws

```
java.lang.Object
  └─util.ws.CrmResult
```

All Implemented Interfaces:
Serializable

```
public class CrmResult
extends Object
implements Serializable
```

Field Summary		Page
static String	_COLD	276
static String	_FAILURE	276
static String	_HOT	276
static String	_PENDING	276
private static HashMap	_table_	276
private String	_value_	276
static CrmResult	COLD	276
static CrmResult	FAILURE	276
static CrmResult	HOT	276
static CrmResult	PENDING	276
private static org.apache.axis.description.TypeDesc	typeDesc	276

Constructor Summary		Page
protected	**CrmResult**(String value)	276

Method Summary		Page
boolean	**equals**(Object obj)	277
static CrmResult	**fromString**(String value)	277
static CrmResult	**fromValue**(String value)	276
static org.apache.axis.encoding.Deserializer	**getDeserializer**(String mechType, Class _javaType, QName _xmlType)	277
static org.apache.axis.encoding.Serializer	**getSerializer**(String mechType, Class _javaType, QName _xmlType)	277
static org.apache.axis.description.TypeDesc	**getTypeDesc**() Return type metadata object	277
String	**getValue**()	276
int	**hashCode**()	277
Object	**readResolve**()	277
String	**toString**()	277

Field Detail

private String **_value_**

private static HashMap **_table_**

public static final String **_PENDING**

public static final String **_HOT**

public static final String **_COLD**

public static final String **_FAILURE**

public static final CrmResult **PENDING**

public static final CrmResult **HOT**

public static final CrmResult **COLD**

public static final CrmResult **FAILURE**

private static org.apache.axis.description.TypeDesc **typeDesc**

Constructor Detail

protected **CrmResult**(String value)

Method Detail

public String **getValue**()

public static CrmResult **fromValue**(String value)
throws IllegalArgumentException

Throws:

IllegalArgumentException

```
public static CrmResult fromString(String value)
                            throws IllegalArgumentException
```

Throws:

`IllegalArgumentException`

```
public boolean equals(Object obj)
```

Overrides:

`equals` in class `Object`

```
public int hashCode()
```

Overrides:

`hashCode` in class `Object`

```
public String toString()
```

Overrides:

`toString` in class `Object`

```
public Object readResolve()
                   throws ObjectStreamException
```

Throws:

`ObjectStreamException`

```
public static org.apache.axis.encoding.Serializer getSerializer(String mechType,
                                                                Class _javaType,
                                                                QName _xmlType)
```

```
public static org.apache.axis.encoding.Deserializer getDeserializer(String mechType,
                                                                    Class _javaType,
                                                                    QName _xmlType)
```

```
public static org.apache.axis.description.TypeDesc getTypeDesc()
```

Return type metadata object

Class MediaFormat

util.ws

```
java.lang.Object
  └ util.ws.MediaFormat
```

All Implemented Interfaces:

Serializable

```
public class MediaFormat
extends Object
implements Serializable
```

Field Summary

Field Summary		Page
static String	BROCHURE	278
static String	CATALOG	278
static String	FLYER	278
static String	GUIFT	278
private static HashMap	table	278
static String	UNSET	278
private String	value	278
static MediaFormat	BROCHURE	278
static MediaFormat	CATALOG	278
static MediaFormat	FLYER	278

static MediaFormat	**GUIFT**	278
private static org.apache.axis.description.TypeDesc	**typeDesc**	278
static MediaFormat	**UNSET**	278

Constructor Summary	Page
MediaFormat(String value)	278

Method Summary		Page
boolean	**equals**(Object obj)	278
static MediaFormat	**fromString**(String value)	278
static MediaFormat	**fromValue**(String value)	278
static org.apache.axis.encoding.Deserializer	**getDeserializer**(String mechType, Class _javaType, QName _xmlType)	279
static org.apache.axis.encoding.Serializer	**getSerializer**(String mechType, Class _javaType, QName _xmlType)	279
static org.apache.axis.description.TypeDesc	**getTypeDesc**() Return type metadata object	279
String	**getValue**()	278
int	**hashCode**()	279
Object	**readResolve**()	279
String	**toString**()	279

Field Detail

private String **_value_**

private static HashMap **_table_**

public static final String **_BROCHURE**

public static final String **_FLYER**

public static final String **_CATALOG**

public static final String **_GUIFT**

public static final String **_UNSET**

public static final MediaFormat **BROCHURE**

public static final MediaFormat **FLYER**

public static final MediaFormat **CATALOG**

public static final MediaFormat **GUIFT**

public static final MediaFormat **UNSET**

private static org.apache.axis.description.TypeDesc **typeDesc**

Constructor Detail

public **MediaFormat**(String value)

Method Detail

public String **getValue**()

public static MediaFormat **fromValue**(String value)
throws IllegalArgumentException

Throws:
IllegalArgumentException

public static MediaFormat **fromString**(String value)
throws IllegalArgumentException

Throws:
IllegalArgumentException

public boolean **equals**(Object obj)

Overrides:
equals in class Object

`public int` **`hashCode`**`()`

Overrides:

`hashCode` in class `Object`

`public String` **`toString`**`()`

Overrides:

`toString` in class `Object`

```
public Object readResolve()
                   throws ObjectStreamException
```

Throws:

`ObjectStreamException`

```
public static org.apache.axis.encoding.Serializer getSerializer(String mechType,
                                                                Class _javaType,
                                                                QName _xmlType)
```

```
public static org.apache.axis.encoding.Deserializer getDeserializer(String mechType,
                                                                    Class _javaType,
                                                                    QName _xmlType)
```

`public static org.apache.axis.description.TypeDesc` **`getTypeDesc`**`()`

Return type metadata object

Package util.ws.crm

Interface Summary		Page
GetCustomerDataFromCRM		280
GetCustomerDataFromCRMService		282

Class Summary		Page
GetCustomerDataFromCRMPortBindingStub		280
GetCustomerDataFromCRMServiceDescriptor		282
GetCustomerDataFromCRMServiceLocator		283

Interface GetCustomerDataFromCRM

util.ws.crm

All Superinterfaces:

Remote

All Known Implementing Classes:

GetCustomerDataFromCRMPortBindingStub

```
public interface GetCustomerDataFromCRM
extends Remote
```

Method Summary		Page
String	**getRecord**(int parameter)	280

Method Detail

```
String getRecord(int parameter)
         throws RemoteException
```

Throws:

RemoteException

Class GetCustomerDataFromCRMPortBindingStub

util.ws.crm

```
java.lang.Object
  └ org.apache.axis.client.Stub
      └ util.ws.crm.GetCustomerDataFromCRMPortBindingStub
```

All Implemented Interfaces:

GetCustomerDataFromCRM, Remote, Stub

```
public class GetCustomerDataFromCRMPortBindingStub
extends org.apache.axis.client.Stub
implements GetCustomerDataFromCRM
```

Field Summary		Page
static org.apache.axis.description.OperationDesc[]	**_operations**	*281*
private Vector	**cachedDeserFactories**	*281*
private Vector	**cachedSerClasses**	*281*
private Vector	**cachedSerFactories**	*281*
private Vector	**cachedSerQNames**	*281*

Fields inherited from class org.apache.axis.client.Stub
_call, cachedEndpoint, cachedPassword, cachedPortName, cachedProperties, cachedTimeout, cachedUsername, maintainSession, maintainSessionSet, service

Constructor Summary	Page
GetCustomerDataFromCRMPortBindingStub()	*281*
GetCustomerDataFromCRMPortBindingStub(URL endpointURL, Service service)	*281*
GetCustomerDataFromCRMPortBindingStub(Service service)	*281*

Method Summary		Page
private static void	**_initOperationDesc1**()	*281*
protected org.apache.axis.client.Call	**createCall**()	*281*
String	**getRecord**(int parameter)	*282*

Methods inherited from class org.apache.axis.client.Stub
_createCall, _getCall, _getProperty, _getPropertyNames, _getService, _setProperty, addAttachment, clearAttachments, clearHeaders, extractAttachments, firstCall, getAttachments, getHeader, getHeaders, getPassword, getPortName, getResponseHeader, getResponseHeaders, getResponseHeaders, getTimeout, getUsername, removeProperty, setAttachments, setHeader, setHeader, setMaintainSession, setPassword, setPortName, setPortName, setRequestHeaders, setTimeout, setUsername

Field Detail

private Vector **cachedSerClasses**

private Vector **cachedSerQNames**

private Vector **cachedSerFactories**

private Vector **cachedDeserFactories**

static org.apache.axis.description.OperationDesc[] **_operations**

Constructor Detail

```
public GetCustomerDataFromCRMPortBindingStub()
                                      throws org.apache.axis.AxisFault
```

```
public GetCustomerDataFromCRMPortBindingStub(URL endpointURL,
                                             Service service)
                                      throws org.apache.axis.AxisFault
```

```
public GetCustomerDataFromCRMPortBindingStub(Service service)
                                      throws org.apache.axis.AxisFault
```

Method Detail

private static void **_initOperationDesc1**()

```
protected org.apache.axis.client.Call createCall()
                                          throws RemoteException
```

Throws:

RemoteException

```
public String getRecord(int parameter)
                 throws RemoteException
```

Specified by:
getRecord in interface GetCustomerDataFromCRM

Throws:
RemoteException

Interface GetCustomerDataFromCRMService

util.ws.crm

All Superinterfaces:
Service

All Known Implementing Classes:
GetCustomerDataFromCRMServiceLocator

```
public interface GetCustomerDataFromCRMService
extends Service
```

Method Summary		Page
GetCustomerDataFromCRM	**getGetCustomerDataFromCRMPort**()	282
GetCustomerDataFromCRM	**getGetCustomerDataFromCRMPort**(URL portAddress)	282
String	**getGetCustomerDataFromCRMPortAddress**()	282

Method Detail

```
String getGetCustomerDataFromCRMPortAddress()
```

```
GetCustomerDataFromCRM getGetCustomerDataFromCRMPort()
                                               throws ServiceException
```

Throws:
ServiceException

```
GetCustomerDataFromCRM getGetCustomerDataFromCRMPort(URL portAddress)
                                               throws ServiceException
```

Throws:
ServiceException

Class GetCustomerDataFromCRMServiceDescriptor

util.ws.crm

```
java.lang.Object
  └ com.tilab.wade.performer.descriptors.webservice.ServiceDescriptor
      └ util.ws.crm.GetCustomerDataFromCRMServiceDescriptor
```

```
public class GetCustomerDataFromCRMServiceDescriptor
extends com.tilab.wade.performer.descriptors.webservice.ServiceDescriptor
```

Field Summary		Page
GetCustomerDataFromCRMServiceLocator	**locator**	283
static String	**NAMESPACE**	283
static String	**SERVICE_NAME**	283

Fields inherited from class com.tilab.wade.performer.descriptors.webservice.ServiceDescriptor
`portDescriptors, SERVICE_DESCRIPTOR_SUFFIX`

Constructor Summary	Page
`GetCustomerDataFromCRMServiceDescriptor()`	283

Method Summary		Page
`Remote`	`getService()`	283
`String`	`getServiceName()`	283
`void`	`setEndpointAddress(String endpointAddress)`	283
`private void`	`setgetRecordParameters(jade.util.leap.List formalParams)`	283

Methods inherited from class com.tilab.wade.performer.descriptors.webservice.ServiceDescriptor
`addOperationDescriptor, addPortDescriptor, getOperationDescriptor, getOperationNames, getPortDescriptor, getPortNames, invoke`

Field Detail

`public static final String` **`SERVICE_NAME`**

`public static final String` **`NAMESPACE`**

`GetCustomerDataFromCRMServiceLocator` **`locator`**

Constructor Detail

`public` **`GetCustomerDataFromCRMServiceDescriptor`**`()`

Method Detail

```
public Remote getService()
                  throws ServiceException
```

Overrides:

`getService` in class `com.tilab.wade.performer.descriptors.webservice.ServiceDescriptor`

Throws:

`ServiceException`

`public String` **`getServiceName`**`()`

Overrides:

`getServiceName` in class `com.tilab.wade.performer.descriptors.webservice.ServiceDescriptor`

`public void` **`setEndpointAddress`**`(String endpointAddress)`

Overrides:

`setEndpointAddress` in class `com.tilab.wade.performer.descriptors.webservice.ServiceDescriptor`

`private void` **`setgetRecordParameters`**`(jade.util.leap.List formalParams)`

Class GetCustomerDataFromCRMServiceLocator

util.ws.crm

```
java.lang.Object
  └ org.apache.axis.client.Service
      └ util.ws.crm.GetCustomerDataFromCRMServiceLocator
```

All Implemented Interfaces:

GetCustomerDataFromCRMService, Referenceable, Serializable, Service

```
public class GetCustomerDataFromCRMServiceLocator
extends org.apache.axis.client.Service
implements GetCustomerDataFromCRMService
```

Nested classes/interfaces inherited from class org.apache.axis.client.Service
Service.HandlerRegistryImpl

Field Summary		Page
private String	**GetCustomerDataFromCRMPort_address**	284
private String	**GetCustomerDataFromCRMPortWSDDServiceName**	284
private HashSet	**ports**	284

Fields inherited from class org.apache.axis.client.Service
_call

Constructor Summary	Page
GetCustomerDataFromCRMServiceLocator()	285
GetCustomerDataFromCRMServiceLocator(String wsdlLoc, QName sName)	285
GetCustomerDataFromCRMServiceLocator(org.apache.axis.EngineConfiguration config)	285

Method Summary		Page
GetCustomerDataFromCRM	**getGetCustomerDataFromCRMPort**()	285
GetCustomerDataFromCRM	**getGetCustomerDataFromCRMPort**(URL portAddress)	285
String	**getGetCustomerDataFromCRMPortAddress**()	285
String	**getGetCustomerDataFromCRMPortWSDDServiceName**()	285
Remote	**getPort**(Class serviceEndpointInterface) For the given interface, get the stub implementation.	285
Remote	**getPort**(QName portName, Class serviceEndpointInterface) For the given interface, get the stub implementation.	285
Iterator	**getPorts**()	286
QName	**getServiceName**()	285
void	**setEndpointAddress**(String portName, String address) Set the endpoint address for the specified port name.	286
void	**setEndpointAddress**(QName portName, String address) Set the endpoint address for the specified port name.	286
void	**setGetCustomerDataFromCRMPortEndpointAddress**(String address)	285
void	**setGetCustomerDataFromCRMPortWSDDServiceName**(String name)	285

Methods inherited from class org.apache.axis.client.Service
createCall, createCall, createCall, createCall, getAxisClient, getCacheWSDL, getCall, getCalls, getEngine, getEngineConfiguration, getHandlerRegistry, getMaintainSession, getPort, getReference, getTypeMappingRegistry, getWSDLDocumentLocation, getWSDLParser, getWSDLService, setCacheWSDL, setEngine, setEngineConfiguration, setMaintainSession, setTypeMappingRegistry, setTypeMappingVersion

Field Detail

```
private String GetCustomerDataFromCRMPort_address
```

```
private String GetCustomerDataFromCRMPortWSDDServiceName
```

```
private HashSet ports
```

Constructor Detail

```
public GetCustomerDataFromCRMServiceLocator()
```

```
public GetCustomerDataFromCRMServiceLocator(org.apache.axis.EngineConfiguration config)
```

```
public GetCustomerDataFromCRMServiceLocator(String wsdlLoc,
                                            QName sName)
                                     throws ServiceException
```

Method Detail

```
public String getGetCustomerDataFromCRMPortAddress()
```

Specified by:
getGetCustomerDataFromCRMPortAddress in interface GetCustomerDataFromCRMService

```
public String getGetCustomerDataFromCRMPortWSDDServiceName()
```

```
public void setGetCustomerDataFromCRMPortWSDDServiceName(String name)
```

```
public GetCustomerDataFromCRM getGetCustomerDataFromCRMPort()
                                                     throws ServiceException
```

Specified by:
getGetCustomerDataFromCRMPort in interface GetCustomerDataFromCRMService
Throws:
ServiceException

```
public GetCustomerDataFromCRM getGetCustomerDataFromCRMPort(URL portAddress)
                                                     throws ServiceException
```

Specified by:
getGetCustomerDataFromCRMPort in interface GetCustomerDataFromCRMService
Throws:
ServiceException

```
public void setGetCustomerDataFromCRMPortEndpointAddress(String address)
```

```
public Remote getPort(Class serviceEndpointInterface)
               throws ServiceException
```

For the given interface, get the stub implementation. If this service has no port for the given interface, then ServiceException is thrown.

Specified by:
getPort in interface Service
Overrides:
getPort in class org.apache.axis.client.Service
Throws:
ServiceException

```
public Remote getPort(QName portName,
                      Class serviceEndpointInterface)
               throws ServiceException
```

For the given interface, get the stub implementation. If this service has no port for the given interface, then ServiceException is thrown.

Specified by:
getPort in interface Service
Overrides:
getPort in class org.apache.axis.client.Service
Throws:
ServiceException

```
public QName getServiceName()
```

Specified by:
getServiceName in interface Service
Overrides:
getServiceName in class org.apache.axis.client.Service

```
public Iterator getPorts()
```

Specified by:

`getPorts` in interface `Service`

Overrides:

`getPorts` in class `org.apache.axis.client.Service`

```
public void setEndpointAddress(String portName,
                               String address)
                        throws ServiceException
```

Set the endpoint address for the specified port name.

Throws:

`ServiceException`

```
public void setEndpointAddress(QName portName,
                               String address)
                        throws ServiceException
```

Set the endpoint address for the specified port name.

Throws:

`ServiceException`

Package workflows

Class Summary		Page
BudgetRF	A workflow class modelling the "Budget Response Factor" business process.	287
CreateJobSchedules	Considers all jobs that request for execution and all the available agents that can perform them and produces a job schedule for each agent, storing it into an Excel File.	290
DirectMailCampaign	The overall Direct mail Campaign process.	294
EstablishTargetMarkets	A workflow class to represent the Establish Target Markets business process.	297
LaunchCampaign	The class to model the actual launching of a marketing campaign.	303
MarketResearch	A workflow to represent the activities of the first phase of a marketing campaign.	308
PreparePiece	The workflow class to model the "Prepare Marketing Piece" business process.	312
QuantifyTAM	A workflow model to represent the "Quantify Total Available Market" business process.	316
ReviewDrafts	The workflow class to model the "Review Drafts of marketing artwork" business process.	320
Segmentation	The workflow class to model the business process "Find Market Segments".	324
SolicitDesign	A workflow class to model the business process "Solicit Vendor to subcontract the artwork design".	327

Enum Summary		Page
EstablishTargetMarkets.failureReason		302

Class BudgetRF

workflows

```
java.lang.Object
  └ jade.core.behaviours.Behaviour
      └ jade.core.behaviours.CompositeBehaviour
          └ jade.core.behaviours.SerialBehaviour
              └ jade.core.behaviours.FSMBehaviour
                  └ com.tilab.wade.performer.WorkflowBehaviour
                      └ workflows.BudgetRF
```

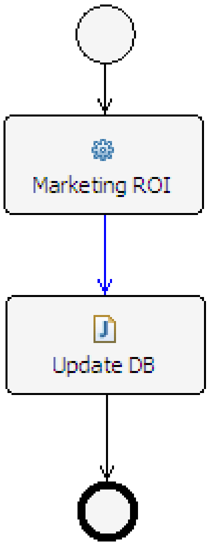
All Implemented Interfaces:

com.tilab.wade.performer.HierarchyNode, jade.util.leap.Serializable, Serializable

---

```
public class BudgetRF
extends com.tilab.wade.performer.WorkflowBehaviour
```

A workflow class modelling the "Budget Response Factor" business process.



Author:

Pavlos Delias

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
`Behaviour.RunnableChangedEvent`

Field Summary		Page
static String	**BUDGETRFTOOLACTIVITY1_ACTIVITY**	289
private jade.core.AID	**marketingCommunicator**	289
private static long	**serialVersionUID**	289
static String	**UPDATEDB_ACTIVITY**	289

Fields inherited from class com.tilab.wade.performer.WorkflowBehaviour
`COLLECT_ASYNCH_SUBFLOWS_STATE, END_STATE, ERROR_STATE, FINAL, formalParams, INITIAL, INITIAL_AND_FINAL, INTERMEDIATE, lastException, myLogger, rootExecutor, START_STATE`

Fields inherited from class jade.core.behaviours.FSMBehaviour
`currentName, lastStates`

Fields inherited from class jade.core.behaviours.CompositeBehaviour
`currentExecuted`

Fields inherited from class jade.core.behaviours.Behaviour
`myAgent, myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING`

Constructor Summary	Page
BudgetRF()	289

Method Summary		Page
private void	**defineActivities**()	289
private void	**defineTransitions**()	290
protected void	**executeBudgetRFToolActivity1**(com.tilab.wade.performer.ApplicationList applications)	290
protected void	**executeUpdateDB**()	290

Methods inherited from class com.tilab.wade.performer.WorkflowBehaviour
changeActivityOrder, checkModifier, checkTermination, commit, configure, deregisterActivity, deregisterTransition, enterInterruptableSection, exitInterruptableSection, fillFormalParameters, fireEvent, getAgent, getBindingManager, getBuildingBlock, getCurrent, getDefaultPriority, getDescriptor, getExecutionContext, getExecutionId, getFormalParameters, getLastErrorEvent, getLastException, getLimit, getModifier, getModifiers, getOutgoingTransitions, getOwner, getRollbackWorkflow, getTracer, getTransactionManager, handleBeginActivity, handleBeginApplication, handleEndActivity, handleEndApplication, handleException, handleInconsistentFSM, handleStateEntered, handleUngroundedParameters, hasJADEDefaultTransition, initRootExecutor, isError, isFireable, isInterrupted, manageBindings, mark, onEnd, onStart, performApplication, performDynamicWebService, performSubflow, performWebService, propagateException, registerActivity, registerActivity, registerTransition, reinit, reset, resume, rollback, setDataStore, setError, setFailureReason, setInterrupted, setUseDataStore, suspend, trace, trace

Methods inherited from class jade.core.behaviours.FSMBehaviour
deregisterDefaultTransition, deregisterState, deregisterTransition, forceTransitionTo, getChildren, getLastExitValue, getName, getPrevious, getState, hasDefaultTransition, registerDefaultTransition, registerDefaultTransition, registerFirstState, registerLastState, registerState, registerTransition, registerTransition, resetStates, scheduleFirst, scheduleNext, stringifyTransitionTable

Methods inherited from class jade.core.behaviours.SerialBehaviour
handle

Methods inherited from class jade.core.behaviours.CompositeBehaviour
action, done, handleBlockEvent, handleRestartEvent, registerAsChild, resetChildren, setAgent

Methods inherited from class jade.core.behaviours.Behaviour
actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, isRunnable, restart, root, setBehaviourName, setExecutionState

Methods inherited from interface com.tilab.wade.performer.HierarchyNode
getBehaviourName, getDataStore, root

Field Detail

private static final long **serialVersionUID**

public static final String **UPDATEDB_ACTIVITY**

public static final String **BUDGETRFTOOLACTIVITY1_ACTIVITY**

private jade.core.AID **marketingCommunicator**

Constructor Detail

public **BudgetRF**()

Method Detail

private void **defineActivities**()

```
protected void executeBudgetRFToolActivity1(com.tilab.wade.performer.ApplicationList applicati
ons)
                                     throws Exception
```

Throws:

Exception

```
protected void executeUpdateDB()
                        throws Exception
```

Throws:

Exception

```
private void defineTransitions()
```

Class CreateJobSchedules

[workflows]

```
java.lang.Object
  └─ jade.core.behaviours.Behaviour
      └─ jade.core.behaviours.CompositeBehaviour
          └─ jade.core.behaviours.SerialBehaviour
              └─ jade.core.behaviours.FSMBehaviour
                  └─ com.tilab.wade.performer.WorkflowBehaviour
                      └─ workflows.CreateJobSchedules
```

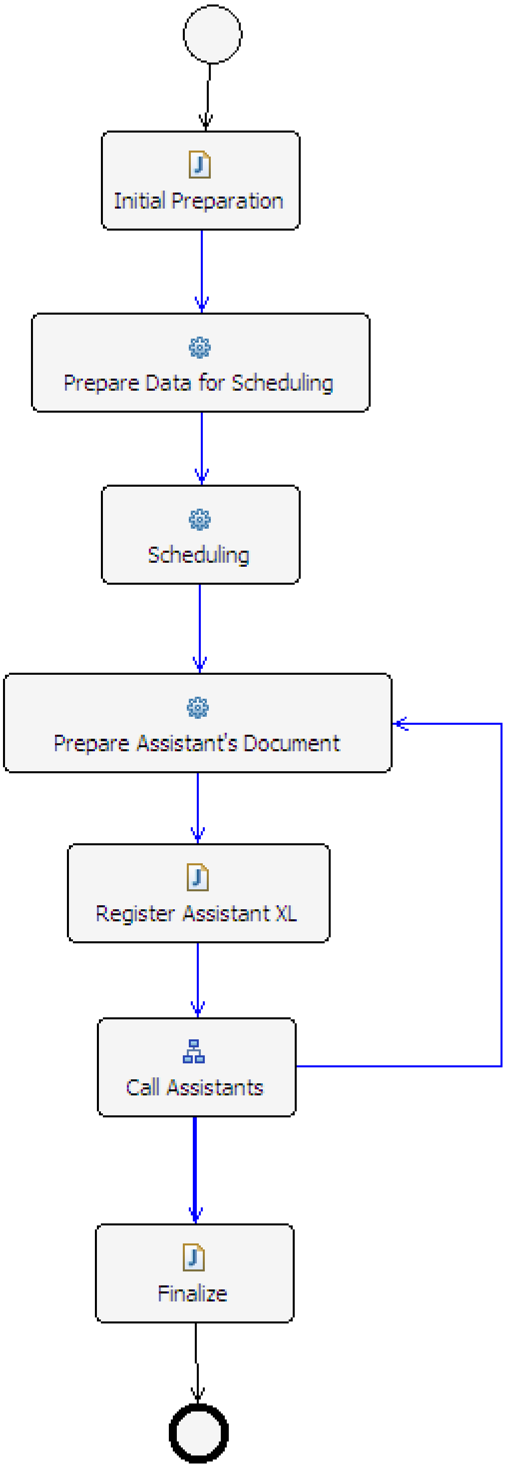
All Implemented Interfaces:

com.tilab.wade.performer.HierarchyNode, jade.util.leap.Serializable, Serializable

---

```
public class CreateJobSchedules
extends com.tilab.wade.performer.WorkflowBehaviour
```

Considers all jobs that request for execution and all the available agents that can perform them and produces a job schedule for each agent, storing it into an Excel File.



Author:
Pavlos Delias

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
`Behaviour.RunnableChangedEvent`

Field Summary

		Page
static String	ASSISTANTDOCUMENT_ACTIVITY	293
static String	ASSISTANTWORKFLOW_ACTIVITY	293
static String	CREATEDATAFORSCHEDULING_ACTIVITY	293
private int	currentAssistant	294
private File	currentAssistantFile	294
static String	FINALIZE_ACTIVITY	293
static String	INITIALIZE_ACTIVITY	293
private int	numOfAssistants	293
private ProductManager	PM	293
static String	REGISTERASSISTANTXL_ACTIVITY	293
static String	SCHEDULING_ACTIVITY	293
private static long	serialVersionUID	293
private Vector<String>	taskNames	293
private Vector<Integer>	taskProcessingTimes	293
private double[][]	taskStartTimes	293
private double[][]	taskToAgents	293

Fields inherited from class com.tilab.wade.performer.WorkflowBehaviour

COLLECT_ASYNCH_SUBFLOWS_STATE, END_STATE, ERROR_STATE, FINAL, formalParams, INITIAL, INITIAL_AND_FINAL, INTERMEDIATE, lastException, myLogger, rootExecutor, START_STATE

Fields inherited from class jade.core.behaviours.FSMBehaviour

currentName, lastStates

Fields inherited from class jade.core.behaviours.CompositeBehaviour

currentExecuted

Fields inherited from class jade.core.behaviours.Behaviour

myAgent, myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING

Constructor Summary

	Page
CreateJobSchedules()	294

Method Summary

		Page
protected boolean	checkAssistantWorkflowToFinalize()	294
private void	defineActivities()	294
private void	defineTransitions()	294
protected void	executeAssistantDocument(com.tilab.wade.performer.ApplicationList applications)	294
protected void	executeAssistantWorkflow(com.tilab.wade.performer.Subflow s)	294
protected void	executeCreateDataForScheduling(com.tilab.wade.performer.ApplicationList applications)	294
protected void	executeFinalize()	294

protected void	**executeInitialize**()	294
protected void	**executeRegisterAssistantXL**()	294
protected void	**executeScheduling**(com.tilab.wade.performer.ApplicationList applications)	294

Methods inherited from class com.tilab.wade.performer.WorkflowBehaviour

changeActivityOrder, checkModifier, checkTermination, commit, configure, deregisterActivity, deregisterTransition, enterInterruptableSection, exitInterruptableSection, fillFormalParameters, fireEvent, getAgent, getBindingManager, getBuildingBlock, getCurrent, getDefaultPriority, getDescriptor, getExecutionContext, getExecutionId, getFormalParameters, getLastErrorEvent, getLastException, getLimit, getModifier, getModifiers, getOutgoingTransitions, getOwner, getRollbackWorkflow, getTracer, getTransactionManager, handleBeginActivity, handleBeginApplication, handleEndActivity, handleEndApplication, handleException, handleInconsistentFSM, handleStateEntered, handleUngroundedParameters, hasJADEDefaultTransition, initRootExecutor, isError, isFireable, isInterrupted, manageBindings, mark, onEnd, onStart, performApplication, performDynamicWebService, performSubflow, performWebService, propagateException, registerActivity, registerActivity, registerTransition, reinit, reset, resume, rollback, setDataStore, setError, setFailureReason, setInterrupted, setUseDataStore, suspend, trace, trace

Methods inherited from class jade.core.behaviours.FSMBehaviour

deregisterDefaultTransition, deregisterState, deregisterTransition, forceTransitionTo, getChildren, getLastExitValue, getName, getPrevious, getState, hasDefaultTransition, registerDefaultTransition, registerDefaultTransition, registerFirstState, registerLastState, registerState, registerTransition, registerTransition, resetStates, scheduleFirst, scheduleNext, stringifyTransitionTable

Methods inherited from class jade.core.behaviours.SerialBehaviour

handle

Methods inherited from class jade.core.behaviours.CompositeBehaviour

action, done, handleBlockEvent, handleRestartEvent, registerAsChild, resetChildren, setAgent

Methods inherited from class jade.core.behaviours.Behaviour

actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, isRunnable, restart, root, setBehaviourName, setExecutionState

Methods inherited from interface com.tilab.wade.performer.HierarchyNode

getBehaviourName, getDataStore, root

Field Detail

private static final long **serialVersionUID**

public static final String **REGISTERASSISTANTXL_ACTIVITY**

public static final String **FINALIZE_ACTIVITY**

public static final String **ASSISTANTWORKFLOW_ACTIVITY**

public static final String **ASSISTANTDOCUMENT_ACTIVITY**

public static final String **SCHEDULING_ACTIVITY**

public static final String **CREATEDATAFORSCHEDULING_ACTIVITY**

public static final String **INITIALIZE_ACTIVITY**

private Vector<String> **taskNames**

private Vector<Integer> **taskProcessingTimes**

private double[][] **taskStartTimes**

private double[][] **taskToAgents**

private ProductManager **PM**

private int **numOfAssistants**

```
private int currentAssistant
```

```
private File currentAssistantFile
```

Constructor Detail

```
public CreateJobSchedules()
```

Method Detail

```
private void defineActivities()
```

```
protected void executeInitialize()
                          throws Exception
```

Throws:
`Exception`

```
private void defineTransitions()
```

```
protected void executeCreateDataForScheduling(com.tilab.wade.performer.ApplicationList applica
tions)
                                       throws Exception
```

Throws:
`Exception`

```
protected void executeScheduling(com.tilab.wade.performer.ApplicationList applications)
                          throws Exception
```

Throws:
`Exception`

```
protected void executeAssistantDocument(com.tilab.wade.performer.ApplicationList applications)
                                 throws Exception
```

Throws:
`Exception`

```
protected void executeAssistantWorkflow(com.tilab.wade.performer.Subflow s)
                                 throws Exception
```

Throws:
`Exception`

```
protected void executeFinalize()
                        throws Exception
```

Throws:
`Exception`

```
protected boolean checkAssistantWorkflowToFinalize()
```

```
protected void executeRegisterAssistantXL()
                                   throws Exception
```

Throws:
`Exception`

Class DirectMailCampaign

workflows

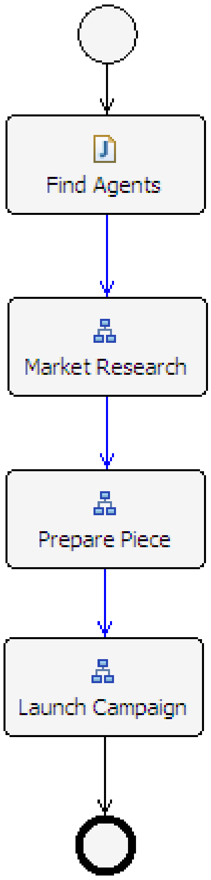
```
java.lang.Object
  └─jade.core.behaviours.Behaviour
      └─jade.core.behaviours.CompositeBehaviour
          └─jade.core.behaviours.SerialBehaviour
              └─jade.core.behaviours.FSMBehaviour
                  └─com.tilab.wade.performer.WorkflowBehaviour
                      └─workflows.DirectMailCampaign
```

All Implemented Interfaces:

com.tilab.wade.performer.HierarchyNode, jade.util.leap.Serializable, Serializable

```
public class DirectMailCampaign
extends com.tilab.wade.performer.WorkflowBehaviour
```

The overall Direct mail Campaign process. It includes the basic steps as subflows.



Author:
Pavlos Delias

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
Behaviour.RunnableChangedEvent

Field Summary		Page
private jade.core.AID	communicator	297
static String	FINDAGENTS_ACTIVITY	297
static String	LAUNCHCAMPAIGN_ACTIVITY	297
private jade.core.AID	manager	297
static String	MARKETRESEARCH_ACTIVITY	297
static String	PREPAREMARKETINGPIECE_ACTIVITY	297

Fields inherited from class com.tilab.wade.performer.WorkflowBehaviour
COLLECT_ASYNCH_SUBFLOWS_STATE, END_STATE, ERROR_STATE, FINAL, formalParams, INITIAL, INITIAL_AND_FINAL, INTERMEDIATE, lastException, myLogger, rootExecutor, START_STATE

Fields inherited from class jade.core.behaviours.FSMBehaviour
`currentName, lastStates`

Fields inherited from class jade.core.behaviours.CompositeBehaviour
`currentExecuted`

Fields inherited from class jade.core.behaviours.Behaviour
`myAgent, myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING`

Constructor Summary	Page
DirectMailCampaign`()`	297

Method Summary		Page
`private void`	**defineActivities**`()`	297
`private void`	**defineTransitions**`()`	297
`protected void`	**executeFindAgents**`()` Contact the Directory Facilitator to get all the necessary references to agents	297
`protected void`	**executeLaunchCampaign**`(com.tilab.wade.performer.Subflow s)`	297
`protected void`	**executeMarketResearch**`(com.tilab.wade.performer.Subflow s)`	297
`protected void`	**executePrepareMarketingPiece**`(com.tilab.wade.performer.Subflow s)`	297

Methods inherited from class com.tilab.wade.performer.WorkflowBehaviour
`changeActivityOrder, checkModifier, checkTermination, commit, configure, deregisterActivity, deregisterTransition, enterInterruptableSection, exitInterruptableSection, fillFormalParameters, fireEvent, getAgent, getBindingManager, getBuildingBlock, getCurrent, getDefaultPriority, getDescriptor, getExecutionContext, getExecutionId, getFormalParameters, getLastErrorEvent, getLastException, getLimit, getModifier, getModifiers, getOutgoingTransitions, getOwner, getRollbackWorkflow, getTracer, getTransactionManager, handleBeginActivity, handleBeginApplication, handleEndActivity, handleEndApplication, handleException, handleInconsistentFSM, handleStateEntered, handleUngroundedParameters, hasJADEDefaultTransition, initRootExecutor, isError, isFireable, isInterrupted, manageBindings, mark, onEnd, onStart, performApplication, performDynamicWebService, performSubflow, performWebService, propagateException, registerActivity, registerActivity, registerTransition, reinit, reset, resume, rollback, setDataStore, setError, setFailureReason, setInterrupted, setUseDataStore, suspend, trace, trace`

Methods inherited from class jade.core.behaviours.FSMBehaviour
`deregisterDefaultTransition, deregisterState, deregisterTransition, forceTransitionTo, getChildren, getLastExitValue, getName, getPrevious, getState, hasDefaultTransition, registerDefaultTransition, registerDefaultTransition, registerFirstState, registerLastState, registerState, registerTransition, registerTransition, resetStates, scheduleFirst, scheduleNext, stringifyTransitionTable`

Methods inherited from class jade.core.behaviours.SerialBehaviour
`handle`

Methods inherited from class jade.core.behaviours.CompositeBehaviour
`action, done, handleBlockEvent, handleRestartEvent, registerAsChild, resetChildren, setAgent`

Methods inherited from class jade.core.behaviours.Behaviour
`actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, isRunnable, restart, root, setBehaviourName, setExecutionState`

Methods inherited from interface com.tilab.wade.performer.HierarchyNode
`getBehaviourName, getDataStore, root`

Field Detail

`public static final String` **`LAUNCHCAMPAIGN_ACTIVITY`**

`public static final String` **`PREPAREMARKETINGPIECE_ACTIVITY`**

`public static final String` **`MARKETRESEARCH_ACTIVITY`**

`private jade.core.AID` **`manager`**

`private jade.core.AID` **`communicator`**

`public static final String` **`FINDAGENTS_ACTIVITY`**

Constructor Detail

`public` **`DirectMailCampaign`**`()`

Method Detail

`private void` **`defineActivities`**`()`

```
protected void executeFindAgents()
                          throws Exception
```

Contact the Directory Facilitator to get all the necessary references to agents

Throws:

`Exception`

```
protected void executeMarketResearch(com.tilab.wade.performer.Subflow s)
                              throws Exception
```

Throws:

`Exception`

`private void` **`defineTransitions`**`()`

```
protected void executePrepareMarketingPiece(com.tilab.wade.performer.Subflow s)
                                     throws Exception
```

Throws:

`Exception`

```
protected void executeLaunchCampaign(com.tilab.wade.performer.Subflow s)
                              throws Exception
```

Throws:

`Exception`

Class EstablishTargetMarkets

[workflows]

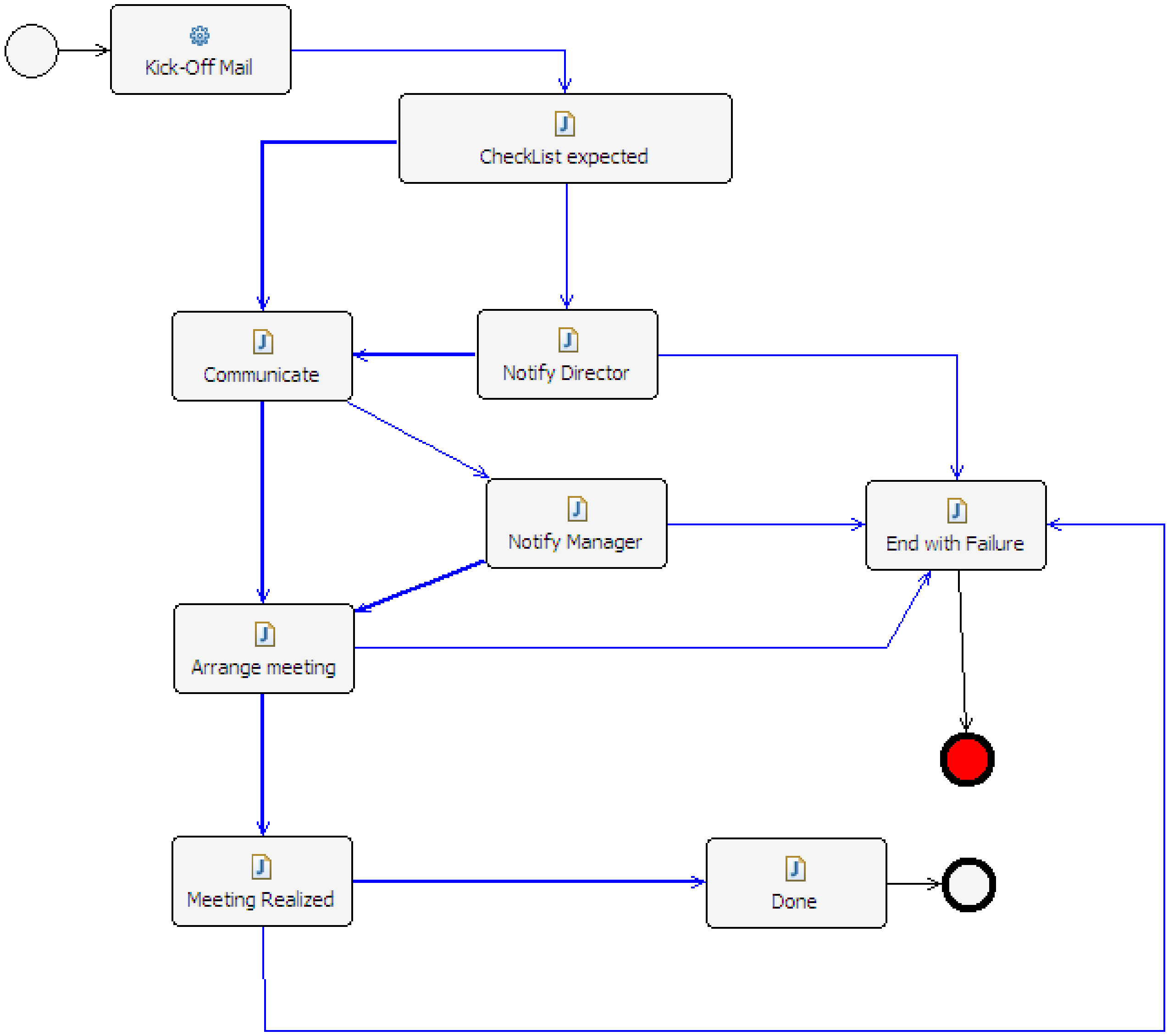
```
java.lang.Object
  └jade.core.behaviours.Behaviour
      └jade.core.behaviours.CompositeBehaviour
          └jade.core.behaviours.SerialBehaviour
              └jade.core.behaviours.FSMBehaviour
                  └com.tilab.wade.performer.WorkflowBehaviour
                      └workflows.EstablishTargetMarkets
```

All Implemented Interfaces:

com.tilab.wade.performer.HierarchyNode, jade.util.leap.Serializable, Serializable

```
public class EstablishTargetMarkets
extends com.tilab.wade.performer.WorkflowBehaviour
```

A workflow class to represent the Establish Target Markets business process. The process must start with a clear target audience in mind: potential buyers of the company's products, current users, deciders, or influencers; individuals, groups, particular publics, or the general public. The target audience can potentially be profiled in terms of the identified market segments Bilateral meetings facilitation activities are also included.



Author:

Pavlos Delias

Nested Class Summary		Page
static enum	**EstablishTargetMarkets.failureReason**	302

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
`Behaviour.RunnableChangedEvent`

Field Summary		Page
static String	**ARRANGEMEETING_ACTIVITY**	301

static String	**CHECKLISTAWARE_ACTIVITY**	*301*
private File	**checkListFile**	*301*
private boolean	**checkListRefined**	*301*
private boolean	**checkListUploaded**	*301*
static String	**COMMUNICATELIST_ACTIVITY**	*301*
static String	**ENDSUCESS_ACTIVITY**	*301*
static String	**FAILURE_ACTIVITY**	*301*
static String	**KICKOFFMAIL_ACTIVITY**	*301*
private MarketingDirector	**MD**	*301*
private String	**meetingConversationId**	*301*
private boolean	**meetingRealized**	*301*
static String	**MEETINGREALIZED_ACTIVITY**	*301*
static String	**NOTIFYDIRECTOR_ACTIVITY**	*301*
static String	**NOTIFYMANAGER_ACTIVITY**	*301*
private jade.core.AID	**productManager**	*301*
private EstablishTargetMarkets.failureReason	**reason**	*301*
private static long	**serialVersionUID**	*301*
private jade.lang.acl.MessageTemplate	**template_CheckList**	*301*
private jade.lang.acl.MessageTemplate	**template_DirectorReply**	*301*
private jade.lang.acl.MessageTemplate	**template_meetingReply**	*301*

Fields inherited from class com.tilab.wade.performer.WorkflowBehaviour

COLLECT_ASYNCH_SUBFLOWS_STATE, END_STATE, ERROR_STATE, FINAL, formalParams, INITIAL, INITIAL_AND_FINAL, INTERMEDIATE, lastException, myLogger, rootExecutor, START_STATE

Fields inherited from class jade.core.behaviours.FSMBehaviour

currentName, lastStates

Fields inherited from class jade.core.behaviours.CompositeBehaviour

currentExecuted

Fields inherited from class jade.core.behaviours.Behaviour

myAgent, myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING

Constructor Summary	Page
EstablishTargetMarkets()	*301*

Method Summary		Page
protected boolean	**checkArrangeMeetingToMeetingRealized**()	*302*
protected boolean	**checkCommunicateListToArrangeMeeting**()	*302*
protected boolean	**checkGenerateCheckListToCommunicateList**()	*302*
protected boolean	**checkMeetingRealizedToEndSucess**()	*302*
protected boolean	**checkNotifyDirectorToCommunicateList**()	*302*

protected boolean	**checkNotifyManagerToArrangeMeeting**()	302
private void	**defineActivities**()	301
private void	**defineTransitions**()	301
protected void	**executeArrangeMeeting**() Arrange meeting between MarketingDirector and ProductManager through the FIPA PROPOSE Protocol.	302
protected void	**executecheckListAware**() Method to wait for the checklist upload.	301
protected void	**executeCommunicateList**() Find the product managers that should be notified based on their "product" property.	301
protected void	**executeEndSucess**()	302
protected void	**executeFailure**()	301
protected void	**executeKickOffMail**(com.tilab.wade.performer.ApplicationList applications)	302
protected void	**executeMeetingRealized**() Waits for a notification that the meeting was indeed realized.	302
protected void	**executeNotifyDirector**()	301
protected void	**executeNotifyManager**()	302
void	**onStart**()	301

Methods inherited from class com.tilab.wade.performer.WorkflowBehaviour
changeActivityOrder, checkModifier, checkTermination, commit, configure, deregisterActivity, deregisterTransition, enterInterruptableSection, exitInterruptableSection, fillFormalParameters, fireEvent, getAgent, getBindingManager, getBuildingBlock, getCurrent, getDefaultPriority, getDescriptor, getExecutionContext, getExecutionId, getFormalParameters, getLastErrorEvent, getLastException, getLimit, getModifier, getModifiers, getOutgoingTransitions, getOwner, getRollbackWorkflow, getTracer, getTransactionManager, handleBeginActivity, handleBeginApplication, handleEndActivity, handleEndApplication, handleException, handleInconsistentFSM, handleStateEntered, handleUngroundedParameters, hasJADEDefaultTransition, initRootExecutor, isError, isFireable, isInterrupted, manageBindings, mark, onEnd, performApplication, performDynamicWebService, performSubflow, performWebService, propagateException, registerActivity, registerActivity, registerTransition, reinit, reset, resume, rollback, setDataStore, setError, setFailureReason, setInterrupted, setUseDataStore, suspend, trace, trace

Methods inherited from class jade.core.behaviours.FSMBehaviour
deregisterDefaultTransition, deregisterState, deregisterTransition, forceTransitionTo, getChildren, getLastExitValue, getName, getPrevious, getState, hasDefaultTransition, registerDefaultTransition, registerDefaultTransition, registerFirstState, registerLastState, registerState, registerTransition, registerTransition, resetStates, scheduleFirst, scheduleNext, stringifyTransitionTable

Methods inherited from class jade.core.behaviours.SerialBehaviour
handle

Methods inherited from class jade.core.behaviours.CompositeBehaviour
action, done, handleBlockEvent, handleRestartEvent, registerAsChild, resetChildren, setAgent

Methods inherited from class jade.core.behaviours.Behaviour
actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, isRunnable, restart, root, setBehaviourName, setExecutionState

Methods inherited from interface com.tilab.wade.performer.HierarchyNode
`getBehaviourName, getDataStore, root`

Field Detail

`public static final String` **`KICKOFFMAIL_ACTIVITY`**

`private static final long` **`serialVersionUID`**

`public static final String` **`ENDSUCESS_ACTIVITY`**

`public static final String` **`MEETINGREALIZED_ACTIVITY`**

`public static final String` **`ARRANGEMEETING_ACTIVITY`**

`public static final String` **`NOTIFYMANAGER_ACTIVITY`**

`public static final String` **`COMMUNICATELIST_ACTIVITY`**

`public static final String` **`FAILURE_ACTIVITY`**

`public static final String` **`NOTIFYDIRECTOR_ACTIVITY`**

`public static final String` **`CHECKLISTAWARE_ACTIVITY`**

`private jade.lang.acl.MessageTemplate` **`template_CheckList`**

`private jade.lang.acl.MessageTemplate` **`template_DirectorReply`**

`private jade.lang.acl.MessageTemplate` **`template_meetingReply`**

`private File` **`checkListFile`**

`private boolean` **`checkListUploaded`**

`private boolean` **`checkListRefined`**

`private boolean` **`meetingRealized`**

`private jade.core.AID` **`productManager`**

`private MarketingDirector` **`MD`**

`private String` **`meetingConversationId`**

`private EstablishTargetMarkets.failureReason` **`reason`**

Constructor Detail

`public` **`EstablishTargetMarkets`**`()`

Method Detail

`private void` **`defineActivities`**`()`

`private void` **`defineTransitions`**`()`

`public void` **`onStart`**`()`

Overrides:

`onStart` in class `com.tilab.wade.performer.WorkflowBehaviour`

```
protected void executecheckListAware()
                              throws Exception
```

Method to wait for the checklist upload. The event is signified by an ACLMessage of the 'INFORM' perfomative.

Throws:

`Exception`

```
protected void executeNotifyDirector()
                              throws Exception
```

Throws:

`Exception`

```
protected void executeFailure()
                       throws Exception
```

Throws:

`Exception`

```
protected void executeCommunicateList()
                               throws Exception
```

Find the product managers that should be notified based on their "product" property. Gets the checkList file through a FileChooser, sets the path as the content of the message and then waits for the managers comments. The comments shold arrive as an ACLMessage of AGREE performative.

Throws:
`Exception`

```
protected void executeNotifyManager()
                             throws Exception
```

Throws:
`Exception`

```
protected void executeArrangeMeeting()
                              throws Exception
```

Arrange meeting between MarketingDirector and ProductManager through the FIPA PROPOSE Protocol.

Throws:
`Exception`

```
protected void executeMeetingRealized()
                               throws Exception
```

Waits for a notification that the meeting was indeed realized. The notification is an ACLMessage of INFORM perfomative.

Throws:
`Exception`

```
protected boolean checkArrangeMeetingToMeetingRealized()
```

```
protected boolean checkGenerateCheckListToCommunicateList()
```

```
protected boolean checkCommunicateListToArrangeMeeting()
```

```
protected boolean checkNotifyDirectorToCommunicateList()
```

```
protected boolean checkNotifyManagerToArrangeMeeting()
```

```
protected void executeEndSucess()
                         throws Exception
```

Throws:
`Exception`

```
protected boolean checkMeetingRealizedToEndSucess()
```

```
protected void executeKickOffMail(com.tilab.wade.performer.ApplicationList applications)
                           throws Exception
```

Throws:
`Exception`

Enum EstablishTargetMarkets.failureReason

[workflows]

```
java.lang.Object
  └ java.lang.Enum<EstablishTargetMarkets.failureReason>
      └ workflows.EstablishTargetMarkets.failureReason
```

All Implemented Interfaces:
Comparable<EstablishTargetMarkets.failureReason>, Serializable

Enclosing class:
EstablishTargetMarkets

```
static enum EstablishTargetMarkets.failureReason
extends Enum<EstablishTargetMarkets.failureReason>
```

Enum Constant Summary	Page
arrangeMeeting	303
checkListDirector	303
checkListManager	303
meetingResult	303

Constructor Summary		Page
private	**EstablishTargetMarkets.failureReason**()	303

Method Summary		Page
static EstablishTargetMarkets.failureReason	**valueOf**(String name)	303
static EstablishTargetMarkets.failureReason[]	**values**()	303

Enum Constant Detail

public static final EstablishTargetMarkets.failureReason **checkListDirector**

public static final EstablishTargetMarkets.failureReason **checkListManager**

public static final EstablishTargetMarkets.failureReason **arrangeMeeting**

public static final EstablishTargetMarkets.failureReason **meetingResult**

Constructor Detail

private **EstablishTargetMarkets.failureReason**()

Method Detail

public static EstablishTargetMarkets.failureReason[] **values**()

public static EstablishTargetMarkets.failureReason **valueOf**(String name)

Class LaunchCampaign

workflows

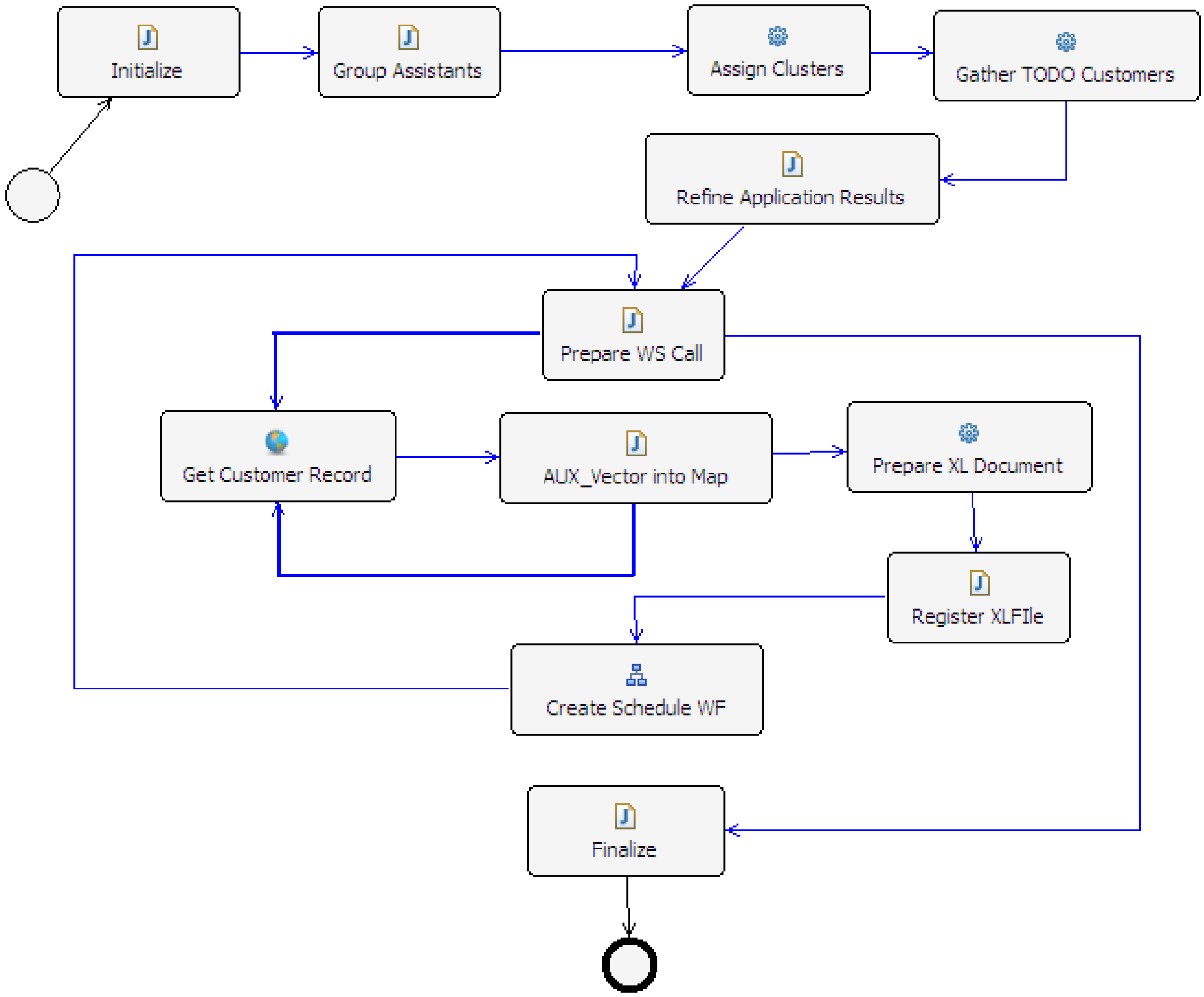
```
java.lang.Object
  └ jade.core.behaviours.Behaviour
      └ jade.core.behaviours.CompositeBehaviour
          └ jade.core.behaviours.SerialBehaviour
              └ jade.core.behaviours.FSMBehaviour
                  └ com.tilab.wade.performer.WorkflowBehaviour
                      └ workflows.LaunchCampaign
```

All Implemented Interfaces:

com.tilab.wade.performer.HierarchyNode, jade.util.leap.Serializable, Serializable

public class **LaunchCampaign**
extends com.tilab.wade.performer.WorkflowBehaviour

The class to model the actual launching of a marketing campaign. It contains customers' clusters assignment to agents, getting customer info through CRM communication, the CreateJobSchedules subflow and database update functions.



Author:

Pavlos Delias

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
`Behaviour.RunnableChangedEvent`

Field Summary		Page
`static String`	**ASSIGNCLUSTERS_ACTIVITY**	307
`private HashMap<String,Vector<String>>`	**assignments**	307
`private Iterator`	**characterIterator**	307
`private boolean`	**charactersLeft**	307
`Connection`	**conn**	307
`static String`	**CREATESCHEDULE_ACTIVITY**	306
`private int`	**customerNum**	307
`private String`	**customerRecord**	307
`private Vector<Integer>`	**customersIDs**	307
`static String`	**FINALIZE_ACTIVITY**	306
`static String`	**GATHERTODOCUSTOMERS_ACTIVITY**	307
`static String`	**GETDATAFROMCRMWS_ACTIVITY**	306

static String	**GROUPASSISTANTS_ACTIVITY**	307
private HashMap<String,Vector<jade.core.AID>>	**groupOfAssistants**	307
static String	**INITIALIZE_ACTIVITY**	306
Statement	**ins**	307
private String	**marketSegmentsFileName**	307
static String	**PREPAREWSCALL_ACTIVITY**	306
static String	**PREPAREXLDOC_ACTIVITY**	306
static String	**PUTVECTORTOMAP_ACTIVITY**	306
private Vector<CustomerRecord>	**records**	307
static String	**REFINEAPPRESULTS_ACTIVITY**	306
ResultSet	**rs**	307
Statement	**stmt**	307
private String	**tempCharacter**	307
private Set<String>	**todoKeySet**	307
private HashMap<String,Vector<Integer>>	**todoLists**	307
static String	**UPDATEDBWITHXLFILE_ACTIVITY**	306
private File	**XLfileName**	307

Fields inherited from class com.tilab.wade.performer.WorkflowBehaviour
COLLECT_ASYNCH_SUBFLOWS_STATE, END_STATE, ERROR_STATE, FINAL, formalParams, INITIAL, INITIAL_AND_FINAL, INTERMEDIATE, lastException, myLogger, rootExecutor, START_STATE

Fields inherited from class jade.core.behaviours.FSMBehaviour
currentName, lastStates

Fields inherited from class jade.core.behaviours.CompositeBehaviour
currentExecuted

Fields inherited from class jade.core.behaviours.Behaviour
myAgent, myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING

Constructor Summary	***Page***
LaunchCampaign()	307

Method Summary		***Page***
protected boolean	**checkprepareWSCallToGetDataFromCRMWS**()	308
protected boolean	**checkPutVectortoMapToGetDataFromCRMWS**()	308
private void	**defineActivities**()	307
private void	**defineTransitions**()	307
protected void	**executeAssignClusters**(com.tilab.wade.performer.ApplicationList applications)	307
protected void	**executeCreateSchedule**(com.tilab.wade.performer.Subflow s)	308
protected void	**executeFinalize**()	308
protected void	**executeGatherTODOCustomers**(com.tilab.wade.performer.ApplicationList applications)	307
protected void	**executeGetDataFromCRMWS**(com.tilab.wade.performer.WebService ws)	307
protected void	**executeGroupAssistants**()	307

protected void	**executeInitialize**() An initial code activity to get the path of the 'MarketSegments' file.	308
protected void	**executeprepareWSCall**()	307
protected void	**executePrepareXLDoc**(com.tilab.wade.performer.ApplicationList applications)	308
protected void	**executePutVectortoMap**()	307
protected void	**executeRefineAppResults**()	307
protected void	**executeUpdateDBwithXLFile**()	308
private CustomerRecord	**parseWSResult**(String s)	308

Methods inherited from class com.tilab.wade.performer.WorkflowBehaviour
changeActivityOrder, checkModifier, checkTermination, commit, configure, deregisterActivity, deregisterTransition, enterInterruptableSection, exitInterruptableSection, fillFormalParameters, fireEvent, getAgent, getBindingManager, getBuildingBlock, getCurrent, getDefaultPriority, getDescriptor, getExecutionContext, getExecutionId, getFormalParameters, getLastErrorEvent, getLastException, getLimit, getModifier, getModifiers, getOutgoingTransitions, getOwner, getRollbackWorkflow, getTracer, getTransactionManager, handleBeginActivity, handleBeginApplication, handleEndActivity, handleEndApplication, handleException, handleInconsistentFSM, handleStateEntered, handleUngroundedParameters, hasJADEDefaultTransition, initRootExecutor, isError, isFireable, isInterrupted, manageBindings, mark, onEnd, onStart, performApplication, performDynamicWebService, performSubflow, performWebService, propagateException, registerActivity, registerActivity, registerTransition, reinit, reset, resume, rollback, setDataStore, setError, setFailureReason, setInterrupted, setUseDataStore, suspend, trace, trace

Methods inherited from class jade.core.behaviours.FSMBehaviour
deregisterDefaultTransition, deregisterState, deregisterTransition, forceTransitionTo, getChildren, getLastExitValue, getName, getPrevious, getState, hasDefaultTransition, registerDefaultTransition, registerDefaultTransition, registerFirstState, registerLastState, registerState, registerTransition, registerTransition, resetStates, scheduleFirst, scheduleNext, stringifyTransitionTable

Methods inherited from class jade.core.behaviours.SerialBehaviour
handle

Methods inherited from class jade.core.behaviours.CompositeBehaviour
action, done, handleBlockEvent, handleRestartEvent, registerAsChild, resetChildren, setAgent

Methods inherited from class jade.core.behaviours.Behaviour
actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, isRunnable, restart, root, setBehaviourName, setExecutionState

Methods inherited from interface com.tilab.wade.performer.HierarchyNode
getBehaviourName, getDataStore, root

Field Detail

public static final String **UPDATEDBWITHXLFILE_ACTIVITY**

public static final String **INITIALIZE_ACTIVITY**

public static final String **FINALIZE_ACTIVITY**

public static final String **CREATESCHEDULE_ACTIVITY**

public static final String **PREPAREXLDOC_ACTIVITY**

public static final String **GETDATAFROMCRMWS_ACTIVITY**

public static final String **REFINEAPPRESULTS_ACTIVITY**

public static final String **PREPAREWSCALL_ACTIVITY**

public static final String **PUTVECTORTOMAP_ACTIVITY**

```
public static final String GATHERTODOCUSTOMERS_ACTIVITY
public static final String ASSIGNCLUSTERS_ACTIVITY
public static final String GROUPASSISTANTS_ACTIVITY
private HashMap<String,Vector<jade.core.AID>> groupOfAssistants
private HashMap<String,Vector<String>> assignments
private HashMap<String,Vector<Integer>> todoLists
private String marketSegmentsFileName
private Set<String> todoKeySet
private Iterator characterIterator
private Vector<Integer> customersIDs
private int customerNum
private String customerRecord
private boolean charactersLeft
private Vector<CustomerRecord> records
private File XLfileName
private String tempCharacter
Connection conn
Statement stmt
Statement ins
ResultSet rs
```

Constructor Detail

```
public LaunchCampaign()
```

Method Detail

```
private void defineActivities()
```

```
protected void executeGroupAssistants()
                               throws Exception
```

Throws:

Exception

```
protected void executeAssignClusters(com.tilab.wade.performer.ApplicationList applications)
                              throws Exception
```

Throws:

Exception

```
private void defineTransitions()
```

```
protected void executeGatherTODOCustomers(com.tilab.wade.performer.ApplicationList application
s)
                                   throws Exception
```

Throws:

Exception

```
protected void executePutVectortoMap()
                              throws Exception
```

Throws:

Exception

```
protected void executeprepareWSCall()
                             throws Exception
```

Throws:

Exception

```
protected void executeRefineAppResults()
                                throws Exception
```

Throws:

Exception

```
protected void executeGetDataFromCRMWS(com.tilab.wade.performer.WebService ws)
                                throws Exception
```

Throws:
`Exception`

```
private CustomerRecord parseWSResult(String s)
```

```
protected void executeFinalize()
                        throws Exception
```

Throws:
`Exception`

```
protected boolean checkPutVectortoMapToGetDataFromCRMWS()
```

```
protected void executePrepareXLDoc(com.tilab.wade.performer.ApplicationList applications)
                            throws Exception
```

Throws:
`Exception`

```
protected void executeCreateSchedule(com.tilab.wade.performer.Subflow s)
                              throws Exception
```

Throws:
`Exception`

```
protected boolean checkprepareWSCallToGetDataFromCRMWS()
```

```
protected void executeInitialize()
                          throws Exception
```

An initial code activity to get the path of the 'MarketSegments' file.

Throws:
`Exception`

```
protected void executeUpdateDBwithXLFile()
                                  throws Exception
```

Throws:
`Exception`

Class MarketResearch

workflows

```
java.lang.Object
  └ jade.core.behaviours.Behaviour
      └ jade.core.behaviours.CompositeBehaviour
          └ jade.core.behaviours.SerialBehaviour
              └ jade.core.behaviours.FSMBehaviour
                  └ com.tilab.wade.performer.WorkflowBehaviour
                      └ workflows.MarketResearch
```

All Implemented Interfaces:

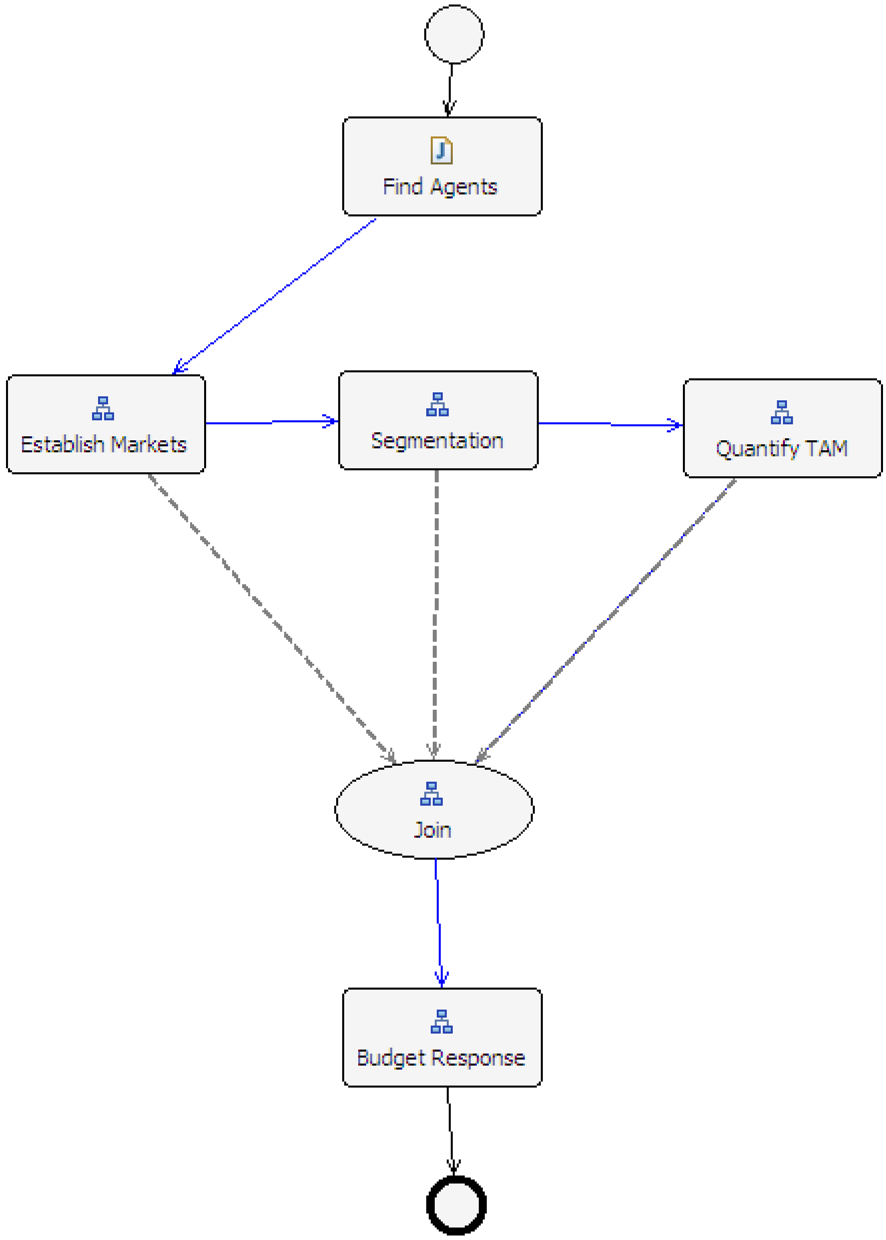
com.tilab.wade.performer.HierarchyNode, jade.util.leap.Serializable, Serializable

---

```
public class MarketResearch
extends com.tilab.wade.performer.WorkflowBehaviour
```

A workflow to represent the activities of the first phase of a marketing campaign. It is used to impose a workflow order to other workflows:`EstablishTargetMarkets`

- `Segmentation`
- `QuantifyTAM`



- BudgetRF

Author:
Pavlos Delias

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
Behaviour.RunnableChangedEvent

Field Summary		Page
private jade.core.AID	**communicator**	311
private jade.core.AID	**director**	311
static String	**ESTABLISHTM_ACTIVITY**	311
static String	**FINDAGENTS_ACTIVITY**	311
private jade.core.AID	**manager**	311
static String	**MARKETRESEARCHSUBFLOWJOINACTIVITY1_ACTIVITY**	311

static String	**QUANTIFY_ACTIVITY**	311
static String	**ROI_ACTIVITY**	311
static String	**SEGMENTATION_ACTIVITY**	311
private static long	**serialVersionUID**	311

Fields inherited from class com.tilab.wade.performer.WorkflowBehaviour

COLLECT_ASYNCH_SUBFLOWS_STATE, END_STATE, ERROR_STATE, FINAL, formalParams, INITIAL, INITIAL_AND_FINAL, INTERMEDIATE, lastException, myLogger, rootExecutor, START_STATE

Fields inherited from class jade.core.behaviours.FSMBehaviour

currentName, lastStates

Fields inherited from class jade.core.behaviours.CompositeBehaviour

currentExecuted

Fields inherited from class jade.core.behaviours.Behaviour

myAgent, myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING

Constructor Summary	Page
MarketResearch()	311

Method Summary		Page
private void	**defineActivities**()	311
private void	**defineTransitions**()	312
protected void	**executeEstablishTM**(com.tilab.wade.performer.Subflow s)	311
protected void	**executeFindAgents**()	312
protected void	**executeMarketResearchSubflowJoinActivity1**(com.tilab.wade.performer.SubflowList ss)	311
protected void	**executeQuantify**(com.tilab.wade.performer.Subflow s)	311
protected void	**executeROI**(com.tilab.wade.performer.Subflow s)	312
protected void	**executeSegmentation**(com.tilab.wade.performer.Subflow s)	311

Methods inherited from class com.tilab.wade.performer.WorkflowBehaviour

changeActivityOrder, checkModifier, checkTermination, commit, configure, deregisterActivity, deregisterTransition, enterInterruptableSection, exitInterruptableSection, fillFormalParameters, fireEvent, getAgent, getBindingManager, getBuildingBlock, getCurrent, getDefaultPriority, getDescriptor, getExecutionContext, getExecutionId, getFormalParameters, getLastErrorEvent, getLastException, getLimit, getModifier, getModifiers, getOutgoingTransitions, getOwner, getRollbackWorkflow, getTracer, getTransactionManager, handleBeginActivity, handleBeginApplication, handleEndActivity, handleEndApplication, handleException, handleInconsistentFSM, handleStateEntered, handleUngroundedParameters, hasJADEDefaultTransition, initRootExecutor, isError, isFireable, isInterrupted, manageBindings, mark, onEnd, onStart, performApplication, performDynamicWebService, performSubflow, performWebService, propagateException, registerActivity, registerActivity, registerTransition, reinit, reset, resume, rollback, setDataStore, setError, setFailureReason, setInterrupted, setUseDataStore, suspend, trace, trace

Methods inherited from class jade.core.behaviours.FSMBehaviour
deregisterDefaultTransition, deregisterState, deregisterTransition, forceTransitionTo, getChildren, getLastExitValue, getName, getPrevious, getState, hasDefaultTransition, registerDefaultTransition, registerDefaultTransition, registerFirstState, registerLastState, registerState, registerTransition, registerTransition, resetStates, scheduleFirst, scheduleNext, stringifyTransitionTable

Methods inherited from class jade.core.behaviours.SerialBehaviour
handle

Methods inherited from class jade.core.behaviours.CompositeBehaviour
action, done, handleBlockEvent, handleRestartEvent, registerAsChild, resetChildren, setAgent

Methods inherited from class jade.core.behaviours.Behaviour
actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, isRunnable, restart, root, setBehaviourName, setExecutionState

Methods inherited from interface com.tilab.wade.performer.HierarchyNode
getBehaviourName, getDataStore, root

Field Detail

private static final long **serialVersionUID**

private jade.core.AID **director**

private jade.core.AID **manager**

private jade.core.AID **communicator**

public static final String **FINDAGENTS_ACTIVITY**

public static final String **ROI_ACTIVITY**

public static final String **MARKETRESEARCHSUBFLOWJOINACTIVITY1_ACTIVITY**

public static final String **QUANTIFY_ACTIVITY**

public static final String **SEGMENTATION_ACTIVITY**

public static final String **ESTABLISHTM_ACTIVITY**

Constructor Detail

public **MarketResearch**()

Method Detail

private void **defineActivities**()

protected void **executeEstablishTM**(com.tilab.wade.performer.Subflow s)
throws Exception

Throws:
Exception

protected void **executeSegmentation**(com.tilab.wade.performer.Subflow s)
throws Exception

Throws:
Exception

protected void **executeQuantify**(com.tilab.wade.performer.Subflow s)
throws Exception

Throws:
Exception

protected void **executeMarketResearchSubflowJoinActivity1**(com.tilab.wade.performer.SubflowList ss)
throws Exception

Throws:
`Exception`

```
protected void executeROI(com.tilab.wade.performer.Subflow s)
                   throws Exception
```

Throws:
`Exception`

```
private void defineTransitions()
```

```
protected void executeFindAgents()
                          throws Exception
```

Throws:
`Exception`

Class PreparePiece

workflows

```
java.lang.Object
  └─jade.core.behaviours.Behaviour
      └─jade.core.behaviours.CompositeBehaviour
          └─jade.core.behaviours.SerialBehaviour
              └─jade.core.behaviours.FSMBehaviour
                  └─com.tilab.wade.performer.WorkflowBehaviour
                      └─workflows.PreparePiece
```

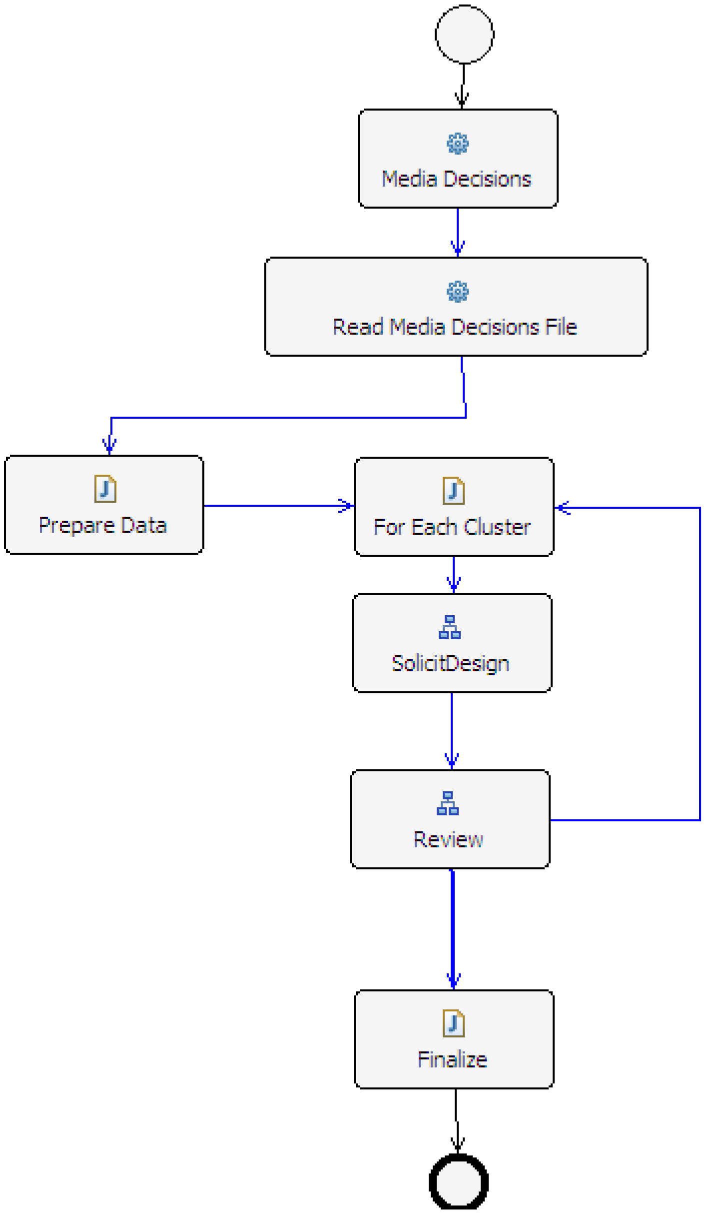
All Implemented Interfaces:

com.tilab.wade.performer.HierarchyNode, jade.util.leap.Serializable, Serializable

---

```
public class PreparePiece
extends com.tilab.wade.performer.WorkflowBehaviour
```

The workflow class to model the "Prepare Marketing Piece" business process. In determining message strategy, management searches for appeals, themes, or ideas that will tie into the brand positioning and help to establish points-of- parity or points-of-difference. A distinct piece will be developped for every market segment (cluster).



Author:
Pavlos Delias

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
`Behaviour.RunnableChangedEvent`

Field Summary		Page
static String	**CLUSTERLOOP_ACTIVITY**	315
private File	**clustersMedia**	315
static String	**FINALIZE_ACTIVITY**	315
private Iterator<String>	**iter**	315
static String	**MEDIADECISIONS_ACTIVITY**	315
jade.core.AID	**MV**	315
private HashMap<String,Offer>	**offers**	315
static String	**PREPARELOOPDATA_ACTIVITY**	315

static String	**PREPAREPIECESUBFLOWACTIVITY1_ACTIVITY**	315
static String	**PREPAREPIECESUBFLOWACTIVITY2_ACTIVITY**	315
static String	**READMEDIAFILE_ACTIVITY**	315
private MediaDecisionsGUI.MediaFormat	**tempOfferFormat**	315
private int	**tempOfferquantity**	315

Fields inherited from class com.tilab.wade.performer.WorkflowBehaviour

COLLECT_ASYNCH_SUBFLOWS_STATE, END_STATE, ERROR_STATE, FINAL, formalParams, INITIAL, INITIAL_AND_FINAL, INTERMEDIATE, lastException, myLogger, rootExecutor, START_STATE

Fields inherited from class jade.core.behaviours.FSMBehaviour

currentName, lastStates

Fields inherited from class jade.core.behaviours.CompositeBehaviour

currentExecuted

Fields inherited from class jade.core.behaviours.Behaviour

myAgent, myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING

Constructor Summary

Constructor	Page
PreparePiece()	315

Method Summary

		Page
protected boolean	**checkPreparePieceSubflowActivity2ToFinalize**()	316
private void	**defineActivities**()	315
private void	**defineTransitions**()	315
protected void	**executeClusterLoop**()	316
protected void	**executeFinalize**()	316
protected void	**executeMediaDecisions**(com.tilab.wade.performer.ApplicationList applications)	315
protected void	**executePrepareLoopData**()	316
protected void	**executePreparePieceSubflowActivity1**(com.tilab.wade.performer.Subflow s)	315
protected void	**executePreparePieceSubflowActivity2**(com.tilab.wade.performer.Subflow s)	315
protected void	**executeReadMediaFile**(com.tilab.wade.performer.ApplicationList applications)	316

Methods inherited from class com.tilab.wade.performer.WorkflowBehaviour

changeActivityOrder, checkModifier, checkTermination, commit, configure, deregisterActivity, deregisterTransition, enterInterruptableSection, exitInterruptableSection, fillFormalParameters, fireEvent, getAgent, getBindingManager, getBuildingBlock, getCurrent, getDefaultPriority, getDescriptor, getExecutionContext, getExecutionId, getFormalParameters, getLastErrorEvent, getLastException, getLimit, getModifier, getModifiers, getOutgoingTransitions, getOwner, getRollbackWorkflow, getTracer, getTransactionManager, handleBeginActivity, handleBeginApplication, handleEndActivity, handleEndApplication, handleException, handleInconsistentFSM, handleStateEntered, handleUngroundedParameters, hasJADEDefaultTransition, initRootExecutor, isError, isFireable, isInterrupted, manageBindings, mark, onEnd, onStart, performApplication, performDynamicWebService, performSubflow, performWebService, propagateException, registerActivity, registerActivity,

registerTransition, reinit, reset, resume, rollback, setDataStore, setError, setFailureReason, setInterrupted, setUseDataStore, suspend, trace, trace

Methods inherited from class jade.core.behaviours.FSMBehaviour
deregisterDefaultTransition, deregisterState, deregisterTransition, forceTransitionTo, getChildren, getLastExitValue, getName, getPrevious, getState, hasDefaultTransition, registerDefaultTransition, registerDefaultTransition, registerFirstState, registerLastState, registerState, registerTransition, registerTransition, resetStates, scheduleFirst, scheduleNext, stringifyTransitionTable

Methods inherited from class jade.core.behaviours.SerialBehaviour
handle

Methods inherited from class jade.core.behaviours.CompositeBehaviour
action, done, handleBlockEvent, handleRestartEvent, registerAsChild, resetChildren, setAgent

Methods inherited from class jade.core.behaviours.Behaviour
actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, isRunnable, restart, root, setBehaviourName, setExecutionState

Methods inherited from interface com.tilab.wade.performer.HierarchyNode
getBehaviourName, getDataStore, root

Field Detail

public static final String **PREPARELOOPDATA_ACTIVITY**

public static final String **FINALIZE_ACTIVITY**

public static final String **CLUSTERLOOP_ACTIVITY**

public static final String **READMEDIAFILE_ACTIVITY**

public static final String **MEDIADECISIONS_ACTIVITY**

public static final String **PREPAREPIECESUBFLOWACTIVITY2_ACTIVITY**

public static final String **PREPAREPIECESUBFLOWACTIVITY1_ACTIVITY**

public jade.core.AID **MV**

private File **clustersMedia**

private HashMap<String,Offer> **offers**

private int **tempOfferquantity**

private MediaDecisionsGUI.MediaFormat **tempOfferFormat**

private Iterator<String> **iter**

Constructor Detail

public **PreparePiece**()

Method Detail

private void **defineActivities**()

protected void **executePreparePieceSubflowActivity1**(com.tilab.wade.performer.Subflow s)
throws Exception

Throws:
Exception

protected void **executePreparePieceSubflowActivity2**(com.tilab.wade.performer.Subflow s)
throws Exception

Throws:
Exception

private void **defineTransitions**()

protected void **executeMediaDecisions**(com.tilab.wade.performer.ApplicationList applications)
throws Exception

Throws:
`Exception`

```
protected void executeReadMediaFile(com.tilab.wade.performer.ApplicationList applications)
                             throws Exception
```

Throws:
`Exception`

```
protected void executeClusterLoop()
                           throws Exception
```

Throws:
`Exception`

```
protected void executeFinalize()
                        throws Exception
```

Throws:
`Exception`

```
protected boolean checkPreparePieceSubflowActivity2ToFinalize()
```

```
protected void executePrepareLoopData()
                               throws Exception
```

Throws:
`Exception`

Class QuantifyTAM

workflows

```
java.lang.Object
  └ jade.core.behaviours.Behaviour
      └ jade.core.behaviours.CompositeBehaviour
          └ jade.core.behaviours.SerialBehaviour
              └ jade.core.behaviours.FSMBehaviour
                  └ com.tilab.wade.performer.WorkflowBehaviour
                      └ workflows.QuantifyTAM
```

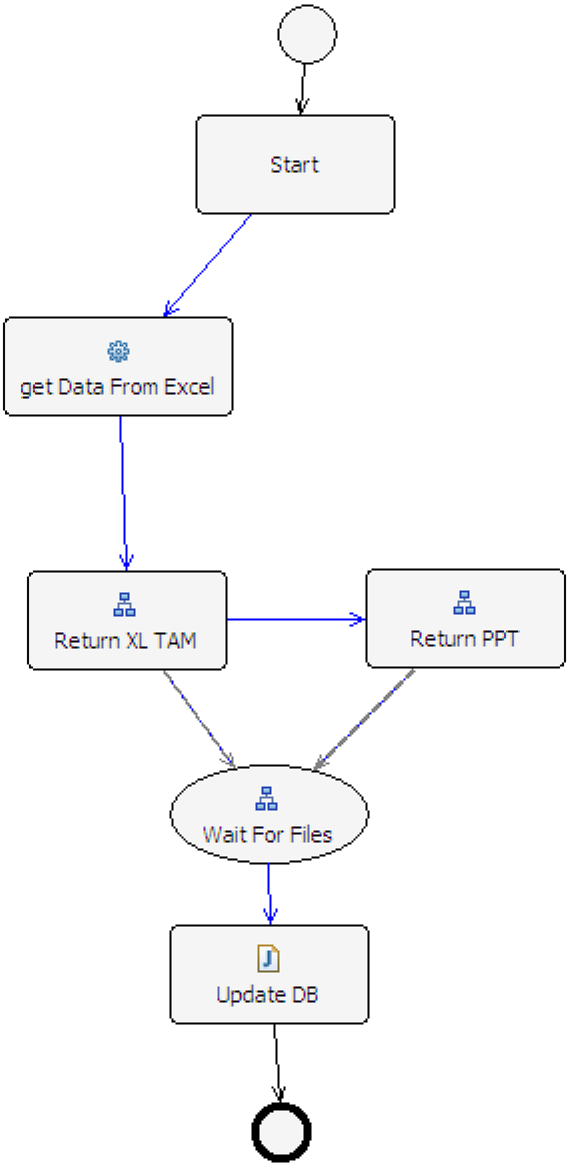
All Implemented Interfaces:

com.tilab.wade.performer.HierarchyNode, jade.util.leap.Serializable, Serializable

---

```
public class QuantifyTAM
extends com.tilab.wade.performer.WorkflowBehaviour
```

A workflow model to represent the "Quantify Total Available Market" business process. Is actually orchestrates marketing reports delivery in order to take the decisions.



Author:
Pavlos Delias

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
`Behaviour.RunnableChangedEvent`

Field Summary

	Field Summary	Page
private List<String>	**excelRanges**	319
static String	**GETDATAFROMXLS_ACTIVITY**	319
private File	**inputExcelFile**	319
private File	**outputExcelFile**	319
private File	**outputPptFile**	319
private List<File>	**PPT**	319
static String	**QUANTIFYTAMROUTEACTIVITY1_ACTIVITY**	319

static String	**RETURNPPT_ACTIVITY**	319
static String	**RETURNXLS_ACTIVITY**	319
static String	**UPDATEDB_ACTIVITY**	319
static String	**WAITFORFILES_ACTIVITY**	319
private List<File>	**XL**	319

Fields inherited from class com.tilab.wade.performer.WorkflowBehaviour

COLLECT_ASYNCH_SUBFLOWS_STATE, END_STATE, ERROR_STATE, FINAL, formalParams, INITIAL, INITIAL_AND_FINAL, INTERMEDIATE, lastException, myLogger, rootExecutor, START_STATE

Fields inherited from class jade.core.behaviours.FSMBehaviour

currentName, lastStates

Fields inherited from class jade.core.behaviours.CompositeBehaviour

currentExecuted

Fields inherited from class jade.core.behaviours.Behaviour

myAgent, myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING

Constructor Summary	Page
QuantifyTAM()	319

Method Summary		Page
private void	**defineActivities**()	319
private void	**defineTransitions**()	319
protected void	**executegetDataFromXLS**(com.tilab.wade.performer.ApplicationList applications)	319
protected void	**executeReturnPPT**(com.tilab.wade.performer.Subflow s)	320
protected void	**executeReturnXLS**(com.tilab.wade.performer.Subflow s)	319
protected void	**executeUpdateDB**()	319
protected void	**executeWaitForFiles**(com.tilab.wade.performer.SubflowList ss)	320

Methods inherited from class com.tilab.wade.performer.WorkflowBehaviour

changeActivityOrder, checkModifier, checkTermination, commit, configure, deregisterActivity, deregisterTransition, enterInterruptableSection, exitInterruptableSection, fillFormalParameters, fireEvent, getAgent, getBindingManager, getBuildingBlock, getCurrent, getDefaultPriority, getDescriptor, getExecutionContext, getExecutionId, getFormalParameters, getLastErrorEvent, getLastException, getLimit, getModifier, getModifiers, getOutgoingTransitions, getOwner, getRollbackWorkflow, getTracer, getTransactionManager, handleBeginActivity, handleBeginApplication, handleEndActivity, handleEndApplication, handleException, handleInconsistentFSM, handleStateEntered, handleUngroundedParameters, hasJADEDefaultTransition, initRootExecutor, isError, isFireable, isInterrupted, manageBindings, mark, onEnd, onStart, performApplication, performDynamicWebService, performSubflow, performWebService, propagateException, registerActivity, registerActivity, registerTransition, reinit, reset, resume, rollback, setDataStore, setError, setFailureReason, setInterrupted, setUseDataStore, suspend, trace, trace

Methods inherited from class jade.core.behaviours.FSMBehaviour
`deregisterDefaultTransition, deregisterState, deregisterTransition, forceTransitionTo, getChildren, getLastExitValue, getName, getPrevious, getState, hasDefaultTransition, registerDefaultTransition, registerDefaultTransition, registerFirstState, registerLastState, registerState, registerTransition, registerTransition, resetStates, scheduleFirst, scheduleNext, stringifyTransitionTable`

Methods inherited from class jade.core.behaviours.SerialBehaviour
`handle`

Methods inherited from class jade.core.behaviours.CompositeBehaviour
`action, done, handleBlockEvent, handleRestartEvent, registerAsChild, resetChildren, setAgent`

Methods inherited from class jade.core.behaviours.Behaviour
`actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, isRunnable, restart, root, setBehaviourName, setExecutionState`

Methods inherited from interface com.tilab.wade.performer.HierarchyNode
`getBehaviourName, getDataStore, root`

Field Detail

`public static final String` **`WAITFORFILES_ACTIVITY`**

`public static final String` **`RETURNPPT_ACTIVITY`**

`public static final String` **`RETURNXLS_ACTIVITY`**

`public static final String` **`UPDATEDB_ACTIVITY`**

`public static final String` **`QUANTIFYTAMROUTEACTIVITY1_ACTIVITY`**

`public static final String` **`GETDATAFROMXLS_ACTIVITY`**

`private List<String>` **`excelRanges`**

`private File` **`inputExcelFile`**

`private File` **`outputExcelFile`**

`private File` **`outputPptFile`**

`private List<File>` **`XL`**

`private List<File>` **`PPT`**

Constructor Detail

`public` **`QuantifyTAM`**`()`

Method Detail

`private void` **`defineActivities`**`()`

```
protected void executegetDataFromXLS(com.tilab.wade.performer.ApplicationList applications)
                              throws Exception
```

Throws:

`Exception`

`private void` **`defineTransitions`**`()`

```
protected void executeUpdateDB()
                        throws Exception
```

Throws:

`Exception`

```
protected void executeReturnXLS(com.tilab.wade.performer.Subflow s)
                         throws Exception
```

Throws:

`Exception`

```
protected void executeReturnPPT(com.tilab.wade.performer.Subflow s)
                         throws Exception
```

Throws:

`Exception`

```
protected void executeWaitForFiles(com.tilab.wade.performer.SubflowList ss)
                            throws Exception
```

Throws:

`Exception`

Class ReviewDrafts

[workflows]

```
java.lang.Object
  └ jade.core.behaviours.Behaviour
      └ jade.core.behaviours.CompositeBehaviour
          └ jade.core.behaviours.SerialBehaviour
              └ jade.core.behaviours.FSMBehaviour
                  └ com.tilab.wade.performer.WorkflowBehaviour
                      └ workflows.ReviewDrafts
```

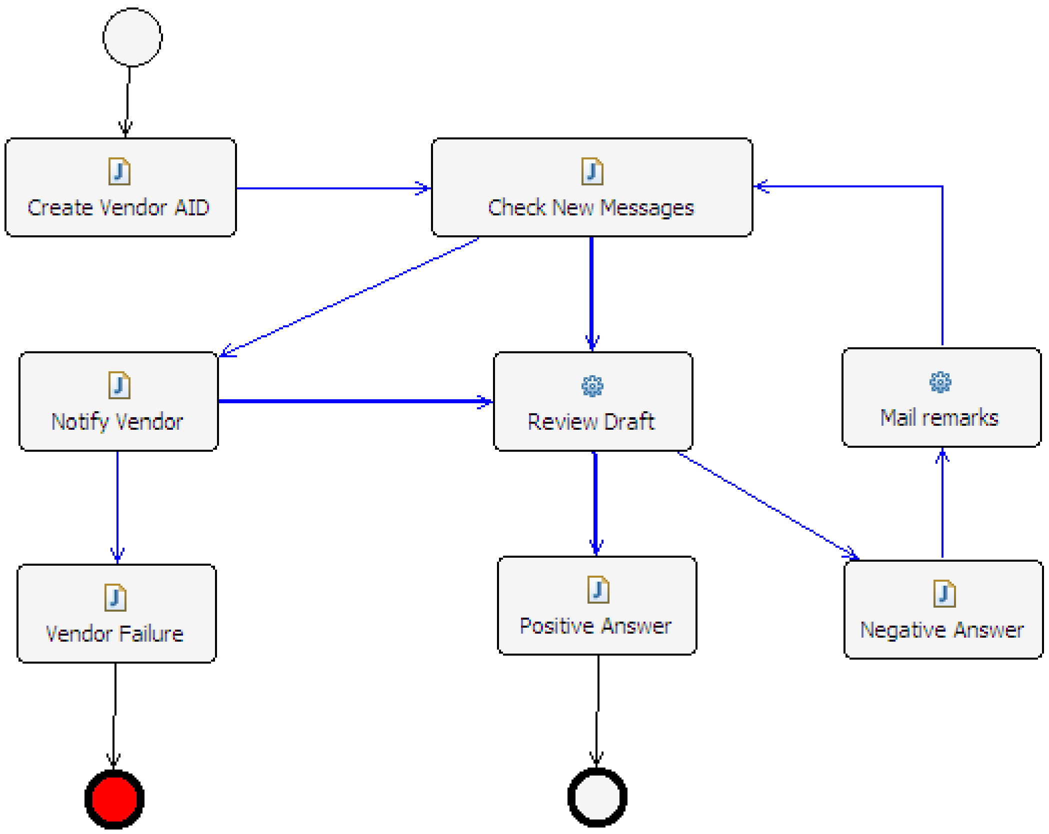
All Implemented Interfaces:

com.tilab.wade.performer.HierarchyNode, jade.util.leap.Serializable, Serializable

---

```
public class ReviewDrafts
extends com.tilab.wade.performer.WorkflowBehaviour
```

The workflow class to model the "Review Drafts of marketing artwork" business process. The draft artwork is sent by the vendor who has subcontracted the job, and it is reviewed. Either a negative answer is sent back with an attached review report, or the artwork is approved.



Author:
Pavlos Delias

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
Behaviour.RunnableChangedEvent

Field Summary

		Page
static String	**CHECKNEWMSG_ACTIVITY**	323
static String	**CREATEMV_ACTIVITY**	322
private jade.lang.acl.ACLMessage	**draftMsg**	323
static String	**FAILURE_ACTIVITY**	323
private boolean	**msgArrived**	323
private jade.core.AID	**MV**	322
static String	**NEGATIVE_ACTIVITY**	323
private jade.lang.acl.MessageTemplate	**newMsg**	323
static String	**NOTIFYVENDOR_ACTIVITY**	323
static String	**POSITIVE_ACTIVITY**	323
String	**reportFileName**	322
static String	**REVIEWDRAFTSTOOL_ACTIVITY**	323
private int	**reviewResult**	323
static String	**SENDMAIL_ACTIVITY**	323

Fields inherited from class com.tilab.wade.performer.WorkflowBehaviour
COLLECT_ASYNCH_SUBFLOWS_STATE, END_STATE, ERROR_STATE, FINAL, formalParams, INITIAL, INITIAL_AND_FINAL, INTERMEDIATE, lastException, myLogger, rootExecutor, START_STATE

Fields inherited from class jade.core.behaviours.FSMBehaviour
currentName, lastStates

Fields inherited from class jade.core.behaviours.CompositeBehaviour
currentExecuted

Fields inherited from class jade.core.behaviours.Behaviour
myAgent, myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING

Constructor Summary

	Page
ReviewDrafts()	323

Method Summary

		Page
protected boolean	**checkCheckNewMsgToReviewDraftsTool**()	323
protected boolean	**checkNotifyVendorToReviewDraftsTool**()	323
protected boolean	**checkReviewDraftsToolToPositive**()	323
private void	**defineActivities**()	323

private void	**defineTransitions**()	323
protected void	**executeCheckNewMsg**()	323
protected void	**executeCreateMV**()	323
protected void	**executeFailure**()	323
protected void	**executeNegative**()	323
protected void	**executeNotifyVendor**()	323
protected void	**executePositive**()	323
protected void	**executeReviewDraftsTool**(com.tilab.wade.performer.ApplicationList applications)	323
protected void	**executeSendMAil**(com.tilab.wade.performer.ApplicationList applications)	323

Methods inherited from class com.tilab.wade.performer.WorkflowBehaviour

changeActivityOrder, checkModifier, checkTermination, commit, configure, deregisterActivity, deregisterTransition, enterInterruptableSection, exitInterruptableSection, fillFormalParameters, fireEvent, getAgent, getBindingManager, getBuildingBlock, getCurrent, getDefaultPriority, getDescriptor, getExecutionContext, getExecutionId, getFormalParameters, getLastErrorEvent, getLastException, getLimit, getModifier, getModifiers, getOutgoingTransitions, getOwner, getRollbackWorkflow, getTracer, getTransactionManager, handleBeginActivity, handleBeginApplication, handleEndActivity, handleEndApplication, handleException, handleInconsistentFSM, handleStateEntered, handleUngroundedParameters, hasJADEDefaultTransition, initRootExecutor, isError, isFireable, isInterrupted, manageBindings, mark, onEnd, onStart, performApplication, performDynamicWebService, performSubflow, performWebService, propagateException, registerActivity, registerActivity, registerTransition, reinit, reset, resume, rollback, setDataStore, setError, setFailureReason, setInterrupted, setUseDataStore, suspend, trace, trace

Methods inherited from class jade.core.behaviours.FSMBehaviour

deregisterDefaultTransition, deregisterState, deregisterTransition, forceTransitionTo, getChildren, getLastExitValue, getName, getPrevious, getState, hasDefaultTransition, registerDefaultTransition, registerDefaultTransition, registerFirstState, registerLastState, registerState, registerTransition, registerTransition, resetStates, scheduleFirst, scheduleNext, stringifyTransitionTable

Methods inherited from class jade.core.behaviours.SerialBehaviour

handle

Methods inherited from class jade.core.behaviours.CompositeBehaviour

action, done, handleBlockEvent, handleRestartEvent, registerAsChild, resetChildren, setAgent

Methods inherited from class jade.core.behaviours.Behaviour

actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, isRunnable, restart, root, setBehaviourName, setExecutionState

Methods inherited from interface com.tilab.wade.performer.HierarchyNode

getBehaviourName, getDataStore, root

Field Detail

public static final String **CREATEMV_ACTIVITY**

private jade.core.AID **MV**

public String **reportFileName**

```
private boolean msgArrived
```

```
private int reviewResult
```

```
private jade.lang.acl.MessageTemplate newMsg
```

```
private jade.lang.acl.ACLMessage draftMsg
```

```
public static final String SENDMAIL_ACTIVITY
```

```
public static final String NEGATIVE_ACTIVITY
```

```
public static final String POSITIVE_ACTIVITY
```

```
public static final String REVIEWDRAFTSTOOL_ACTIVITY
```

```
public static final String FAILURE_ACTIVITY
```

```
public static final String NOTIFYVENDOR_ACTIVITY
```

```
public static final String CHECKNEWMSG_ACTIVITY
```

Constructor Detail

```
public ReviewDrafts()
```

Method Detail

```
private void defineActivities()
```

```
protected void executeCheckNewMsg()
                           throws Exception
```

Throws:

`Exception`

```
protected void executeNotifyVendor()
                            throws Exception
```

Throws:

`Exception`

```
protected void executeFailure()
                       throws Exception
```

Throws:

`Exception`

```
protected void executeReviewDraftsTool(com.tilab.wade.performer.ApplicationList applications)
                                throws Exception
```

Throws:

`Exception`

```
protected void executePositive()
                        throws Exception
```

Throws:

`Exception`

```
protected void executeNegative()
                        throws Exception
```

Throws:

`Exception`

```
protected void executeSendMAil(com.tilab.wade.performer.ApplicationList applications)
                        throws Exception
```

Throws:

`Exception`

```
private void defineTransitions()
```

```
protected boolean checkCheckNewMsgToReviewDraftsTool()
```

```
protected boolean checkNotifyVendorToReviewDraftsTool()
```

```
protected boolean checkReviewDraftsToolToPositive()
```

```
protected void executeCreateMV()
                        throws Exception
```

Throws:

`Exception`

Class Segmentation

[workflows]

```
java.lang.Object
  └ jade.core.behaviours.Behaviour
      └ jade.core.behaviours.CompositeBehaviour
          └ jade.core.behaviours.SerialBehaviour
              └ jade.core.behaviours.FSMBehaviour
                  └ com.tilab.wade.performer.WorkflowBehaviour
                      └ workflows.Segmentation
```

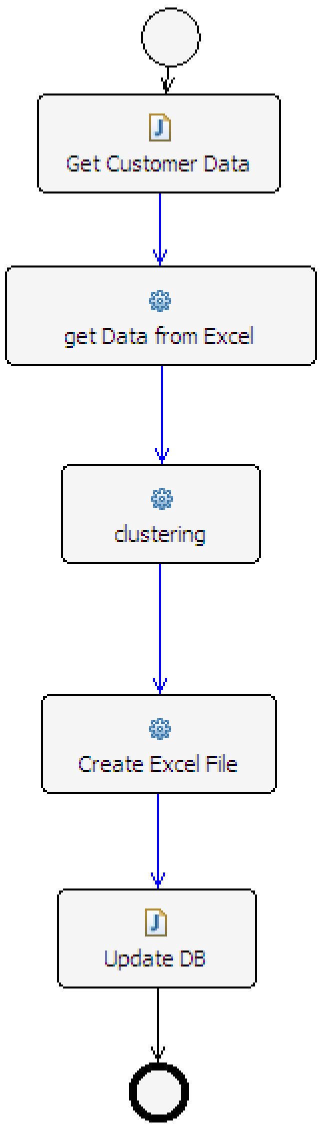
All Implemented Interfaces:

com.tilab.wade.performer.HierarchyNode, jade.util.leap.Serializable, Serializable

---

```
public class Segmentation
extends com.tilab.wade.performer.WorkflowBehaviour
```

The workflow class to model the business process "Find Market Segments". It orchestrates the application of a



clustering algorithm to the customer database.

Author:

Pavlos Delias

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
Behaviour.RunnableChangedEvent

Field Summary

		Page
private double[][]	**centroids**	326
static String	**CLUSTERCUSTOMERS_ACTIVITY**	327
private double[][]	**clusters**	326
Connection	**conn**	326
private String	**customerDataFileName**	326
static String	**EXCELSEGMENTATION_ACTIVITY**	327
static String	**GETCUSTOMERDATA_ACTIVITY**	326
static String	**GETDATAFROMXLS_ACTIVITY**	327
Statement	**ins**	327
private File	**marketSegmentsFile**	326
ResultSet	**rs**	327
private static long	**serialVersionUID**	326
Statement	**stmt**	326
static String	**UPDATEDB_ACTIVITY**	326
private double[][]	**weights**	326

Fields inherited from class com.tilab.wade.performer.WorkflowBehaviour
COLLECT_ASYNCH_SUBFLOWS_STATE, END_STATE, ERROR_STATE, FINAL, formalParams, INITIAL, INITIAL_AND_FINAL, INTERMEDIATE, lastException, myLogger, rootExecutor, START_STATE

Fields inherited from class jade.core.behaviours.FSMBehaviour
currentName, lastStates

Fields inherited from class jade.core.behaviours.CompositeBehaviour
currentExecuted

Fields inherited from class jade.core.behaviours.Behaviour
myAgent, myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING

Constructor Summary

	Page
Segmentation()	327

Method Summary

		Page
private void	**defineActivities**()	327
private void	**defineTransitions**()	327
protected void	**executeclusterCustomers**(com.tilab.wade.performer.ApplicationList applications)	327

protected void	**executeexcelSegmentation**(com.tilab.wade.performer.ApplicationList applications)	327
protected void	**executeGetCustomerData**()	327
protected void	**executegetDataFromXLS**(com.tilab.wade.performer.ApplicationList applications)	327
protected void	**executeSegmenationToolActivity1**(com.tilab.wade.performer.ApplicationList applications)	327
protected void	**executeUpdateDB**()	327

Methods inherited from class com.tilab.wade.performer.WorkflowBehaviour
changeActivityOrder, checkModifier, checkTermination, commit, configure, deregisterActivity, deregisterTransition, enterInterruptableSection, exitInterruptableSection, fillFormalParameters, fireEvent, getAgent, getBindingManager, getBuildingBlock, getCurrent, getDefaultPriority, getDescriptor, getExecutionContext, getExecutionId, getFormalParameters, getLastErrorEvent, getLastException, getLimit, getModifier, getModifiers, getOutgoingTransitions, getOwner, getRollbackWorkflow, getTracer, getTransactionManager, handleBeginActivity, handleBeginApplication, handleEndActivity, handleEndApplication, handleException, handleInconsistentFSM, handleStateEntered, handleUngroundedParameters, hasJADEDefaultTransition, initRootExecutor, isError, isFireable, isInterrupted, manageBindings, mark, onEnd, onStart, performApplication, performDynamicWebService, performSubflow, performWebService, propagateException, registerActivity, registerActivity, registerTransition, reinit, reset, resume, rollback, setDataStore, setError, setFailureReason, setInterrupted, setUseDataStore, suspend, trace, trace

Methods inherited from class jade.core.behaviours.FSMBehaviour
deregisterDefaultTransition, deregisterState, deregisterTransition, forceTransitionTo, getChildren, getLastExitValue, getName, getPrevious, getState, hasDefaultTransition, registerDefaultTransition, registerDefaultTransition, registerFirstState, registerLastState, registerState, registerTransition, registerTransition, resetStates, scheduleFirst, scheduleNext, stringifyTransitionTable

Methods inherited from class jade.core.behaviours.SerialBehaviour
handle

Methods inherited from class jade.core.behaviours.CompositeBehaviour
action, done, handleBlockEvent, handleRestartEvent, registerAsChild, resetChildren, setAgent

Methods inherited from class jade.core.behaviours.Behaviour
actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, isRunnable, restart, root, setBehaviourName, setExecutionState

Methods inherited from interface com.tilab.wade.performer.HierarchyNode
getBehaviourName, getDataStore, root

Field Detail

public static final String **UPDATEDB_ACTIVITY**

public static final String **GETCUSTOMERDATA_ACTIVITY**

private static final long **serialVersionUID**

private double[][] **weights**

private double[][] **clusters**

private double[][] **centroids**

private String **customerDataFileName**

private File **marketSegmentsFile**

Connection **conn**

Statement **stmt**

`Statement` **`ins`**

`ResultSet` **`rs`**

`public static final String` **`EXCELSEGMENTATION_ACTIVITY`**

`public static final String` **`CLUSTERCUSTOMERS_ACTIVITY`**

`public static final String` **`GETDATAFROMXLS_ACTIVITY`**

Constructor Detail

`public` **`Segmentation`**`()`

Method Detail

`private void` **`defineActivities`**`()`

```
protected void executegetDataFromXLS(com.tilab.wade.performer.ApplicationList applications)
                              throws Exception
```

Throws:

`Exception`

```
protected void executeclusterCustomers(com.tilab.wade.performer.ApplicationList applications)
                                throws Exception
```

Throws:

`Exception`

`private void` **`defineTransitions`**`()`

```
protected void executeexcelSegmentation(com.tilab.wade.performer.ApplicationList applications)
                                 throws Exception
```

Throws:

`Exception`

```
protected void executeSegmenationToolActivity1(com.tilab.wade.performer.ApplicationList applic
ations)
                                        throws Exception
```

Throws:

`Exception`

```
protected void executeGetCustomerData()
                               throws Exception
```

Throws:

`Exception`

```
protected void executeUpdateDB()
                        throws Exception
```

Throws:

`Exception`

Class SolicitDesign

workflows

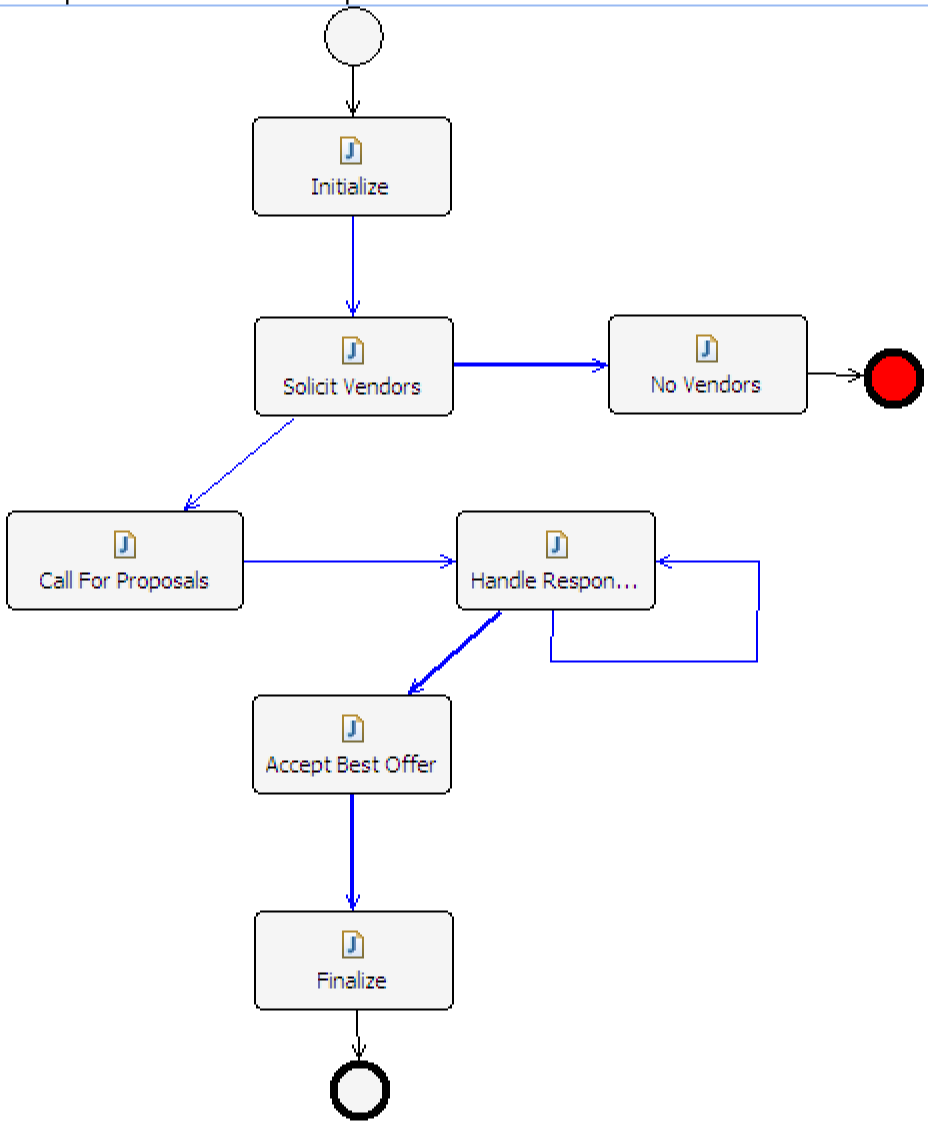
```
java.lang.Object
  └ jade.core.behaviours.Behaviour
      └ jade.core.behaviours.CompositeBehaviour
          └ jade.core.behaviours.SerialBehaviour
              └ jade.core.behaviours.FSMBehaviour
                  └ com.tilab.wade.performer.WorkflowBehaviour
                      └ workflows.SolicitDesign
```

All Implemented Interfaces:

com.tilab.wade.performer.HierarchyNode, jade.util.leap.Serializable, Serializable

```
public class SolicitDesign
extends com.tilab.wade.performer.WorkflowBehaviour
```

A workflow class to model the business process "Solicit Vendor to subcontract the artwork design". The contract net negotiation protocol is used to accept the best offer.



Author:
Pavlos Delias

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
Behaviour.RunnableChangedEvent

Field Summary		Page
static String	ACCEPTBEST_ACTIVITY	330
private double	bestPrice	330
private jade.lang.acl.ACLMessage	bestProposal	330
static String	CFP_ACTIVITY	330
private String	convId	330
static String	FINALIZE_ACTIVITY	330
static String	HANDLERESPONSES_ACTIVITY	330
static String	INIT_ACTIVITY	330
private boolean	isAssigned	330

private jade.lang.acl.MessageTemplate	**myTemplate**	330
static String	**NOVENDORS_ACTIVITY**	330
private boolean	**noVendorsAvailable**	330
private int	**repliesCnt**	330
static String	**SOLICITVENDORS_ACTIVITY**	330
private int	**totExpectedReplies**	330
private Vector<jade.core.AID>	**vendors**	330

Fields inherited from class com.tilab.wade.performer.WorkflowBehaviour

COLLECT_ASYNCH_SUBFLOWS_STATE, END_STATE, ERROR_STATE, FINAL, formalParams, INITIAL, INITIAL_AND_FINAL, INTERMEDIATE, lastException, myLogger, rootExecutor, START_STATE

Fields inherited from class jade.core.behaviours.FSMBehaviour

currentName, lastStates

Fields inherited from class jade.core.behaviours.CompositeBehaviour

currentExecuted

Fields inherited from class jade.core.behaviours.Behaviour

myAgent, myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING

Constructor Summary	Page
SolicitDesign()	330

Method Summary		Page
protected boolean	**checkexistVendors**()	331
protected boolean	**checkHandleResponsesToAcceptBest**()	331
protected boolean	**checkSolicitVendorsToNoVendors**()	331
protected boolean	**checkVendorFound**()	331
private void	**defineActivities**()	330
private void	**defineTransitions**()	331
protected void	**executeAcceptBest**()	331
protected void	**executeCFP**()	331
protected void	**executeFinalize**()	331
protected void	**executeHandleResponses**()	331
protected void	**executeInit**()	330
protected void	**executeNoVendors**()	331
protected void	**executeSolicitVendors**()	331

Methods inherited from class com.tilab.wade.performer.WorkflowBehaviour

changeActivityOrder, checkModifier, checkTermination, commit, configure, deregisterActivity, deregisterTransition, enterInterruptableSection, exitInterruptableSection,

`fillFormalParameters, fireEvent, getAgent, getBindingManager, getBuildingBlock, getCurrent, getDefaultPriority, getDescriptor, getExecutionContext, getExecutionId, getFormalParameters, getLastErrorEvent, getLastException, getLimit, getModifier, getModifiers, getOutgoingTransitions, getOwner, getRollbackWorkflow, getTracer, getTransactionManager, handleBeginActivity, handleBeginApplication, handleEndActivity, handleEndApplication, handleException, handleInconsistentFSM, handleStateEntered, handleUngroundedParameters, hasJADEDefaultTransition, initRootExecutor, isError, isFireable, isInterrupted, manageBindings, mark, onEnd, onStart, performApplication, performDynamicWebService, performSubflow, performWebService, propagateException, registerActivity, registerActivity, registerTransition, reinit, reset, resume, rollback, setDataStore, setError, setFailureReason, setInterrupted, setUseDataStore, suspend, trace, trace`

Methods inherited from class jade.core.behaviours.FSMBehaviour
`deregisterDefaultTransition, deregisterState, deregisterTransition, forceTransitionTo, getChildren, getLastExitValue, getName, getPrevious, getState, hasDefaultTransition, registerDefaultTransition, registerDefaultTransition, registerFirstState, registerLastState, registerState, registerTransition, registerTransition, resetStates, scheduleFirst, scheduleNext, stringifyTransitionTable`

Methods inherited from class jade.core.behaviours.SerialBehaviour
`handle`

Methods inherited from class jade.core.behaviours.CompositeBehaviour
`action, done, handleBlockEvent, handleRestartEvent, registerAsChild, resetChildren, setAgent`

Methods inherited from class jade.core.behaviours.Behaviour
`actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, isRunnable, restart, root, setBehaviourName, setExecutionState`

Methods inherited from interface com.tilab.wade.performer.HierarchyNode
`getBehaviourName, getDataStore, root`

Field Detail

`public static final String` **`NOVENDORS_ACTIVITY`**

`public static final String` **`FINALIZE_ACTIVITY`**

`public static final String` **`ACCEPTBEST_ACTIVITY`**

`public static final String` **`HANDLERESPONSES_ACTIVITY`**

`public static final String` **`CFP_ACTIVITY`**

`public static final String` **`SOLICITVENDORS_ACTIVITY`**

`public static final String` **`INIT_ACTIVITY`**

`private Vector<jade.core.AID>` **`vendors`**

`private boolean` **`noVendorsAvailable`**

`private int` **`totExpectedReplies`**

`private int` **`repliesCnt`**

`private String` **`convId`**

`private jade.lang.acl.MessageTemplate` **`myTemplate`**

`private jade.lang.acl.ACLMessage` **`bestProposal`**

`private boolean` **`isAssigned`**

`private double` **`bestPrice`**

Constructor Detail

`public` **`SolicitDesign`**`()`

Method Detail

`private void` **`defineActivities`**`()`

```
protected void executeInit()
                    throws Exception
```

Throws:
`Exception`

`protected void` **`executeSolicitVendors`**`()`
`throws Exception`

Throws:
`Exception`

`protected void` **`executeCFP`**`()`
`throws Exception`

Throws:
`Exception`

`protected void` **`executeHandleResponses`**`()`
`throws Exception`

Throws:
`Exception`

`protected void` **`executeAcceptBest`**`()`
`throws Exception`

Throws:
`Exception`

`private void` **`defineTransitions`**`()`

`protected boolean` **`checkHandleResponsesToAcceptBest`**`()`

`protected boolean` **`checkexistVendors`**`()`

`protected void` **`executeFinalize`**`()`
`throws Exception`

Throws:
`Exception`

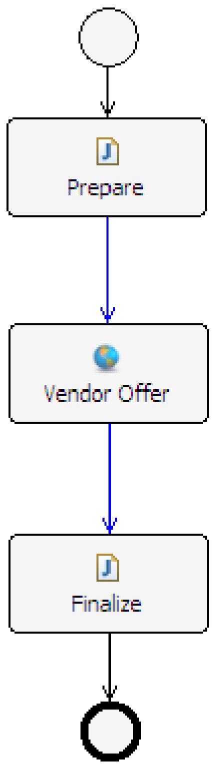
`protected boolean` **`checkVendorFound`**`()`

`protected void` **`executeNoVendors`**`()`
`throws Exception`

Throws:
`Exception`

`protected boolean` **`checkSolicitVendorsToNoVendors`**`()`

Package workflows.auxiliary

Class Summary		Page
AssistantLaunching	The actual performing of the campaign from the assistant-agent view.	332
CreateTAMFile	A workflow class to implement a Subflow.	336
FetchPptFile	A workflow class to implement a Subflow.	338
ProcessBatchMail	A workflow class implemented as an intermediate step of the Contact Center Management process.	340
SpectralScheduling	A workflow class to orchestrate the application of a scheduling algorithm.	343
VendorOffer	A workflow class to model the vendors inner behavior 	346

Class AssistantLaunching

workflows.auxiliary

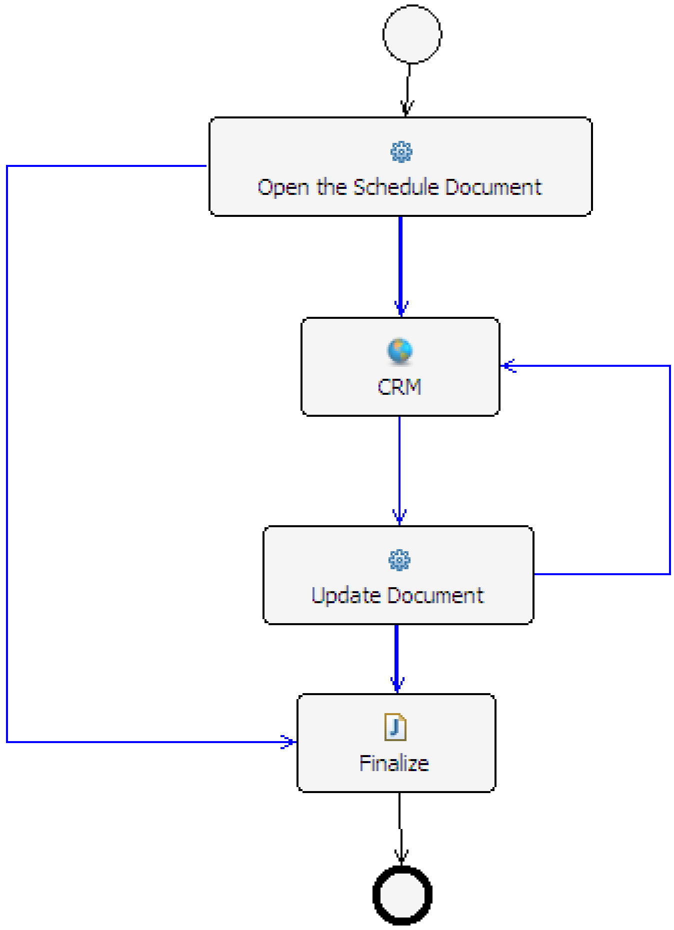
```
java.lang.Object
  └ jade.core.behaviours.Behaviour
      └ jade.core.behaviours.CompositeBehaviour
          └ jade.core.behaviours.SerialBehaviour
              └ jade.core.behaviours.FSMBehaviour
                  └ com.tilab.wade.performer.WorkflowBehaviour
                      └ workflows.auxiliary.AssistantLaunching
```

All Implemented Interfaces:

com.tilab.wade.performer.HierarchyNode, jade.util.leap.Serializable, Serializable

```
public class AssistantLaunching
extends com.tilab.wade.performer.WorkflowBehaviour
```

The actual performing of the campaign from the assistant-agent view. It opens the schedule document, contacts the CRM through a Web Service to find additional customer info. The contact results are saved to an Excel



document.

Author:
Pavlos Delias

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
Behaviour.RunnableChangedEvent

Field Summary		Page
private int	**currentCustomerIndex**	335
private Vector<String>	**customerNames**	335
static String	**FINALIZE_ACTIVITY**	335
private String	**myFile**	335
private int	**numOfCustomers**	335
static String	**OPENSCHEDULE_ACTIVITY**	335
private Vector<Integer>	**processingTimes**	335
private CrmResult	**res**	335
private String	**scheduleImage**	335
private String	**updatedFileName**	335

static String	UPDATEDOCUMENT_ACTIVITY	335
static String	WS2CRM_ACTIVITY	335

Fields inherited from class com.tilab.wade.performer.WorkflowBehaviour
COLLECT_ASYNCH_SUBFLOWS_STATE, END_STATE, ERROR_STATE, FINAL, formalParams, INITIAL, INITIAL_AND_FINAL, INTERMEDIATE, lastException, myLogger, rootExecutor, START_STATE

Fields inherited from class jade.core.behaviours.FSMBehaviour
currentName, lastStates

Fields inherited from class jade.core.behaviours.CompositeBehaviour
currentExecuted

Fields inherited from class jade.core.behaviours.Behaviour
myAgent, myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING

Constructor Summary	Page
AssistantLaunching()	335

Method Summary		Page
protected boolean	checkOpenScheduleToWS2CRM()	336
protected boolean	checkUpdateDocumentToFinalize()	335
private void	defineActivities()	335
private void	defineTransitions()	335
protected void	executeFinalize()	335
protected void	executeOpenSchedule(com.tilab.wade.performer.ApplicationList applications)	335
protected void	executeUpdateDocument(com.tilab.wade.performer.ApplicationList applications)	335
protected void	executeWS2CRM(com.tilab.wade.performer.WebService ws)	335

Methods inherited from class com.tilab.wade.performer.WorkflowBehaviour
changeActivityOrder, checkModifier, checkTermination, commit, configure, deregisterActivity, deregisterTransition, enterInterruptableSection, exitInterruptableSection, fillFormalParameters, fireEvent, getAgent, getBindingManager, getBuildingBlock, getCurrent, getDefaultPriority, getDescriptor, getExecutionContext, getExecutionId, getFormalParameters, getLastErrorEvent, getLastException, getLimit, getModifier, getModifiers, getOutgoingTransitions, getOwner, getRollbackWorkflow, getTracer, getTransactionManager, handleBeginActivity, handleBeginApplication, handleEndActivity, handleEndApplication, handleException, handleInconsistentFSM, handleStateEntered, handleUngroundedParameters, hasJADEDefaultTransition, initRootExecutor, isError, isFireable, isInterrupted, manageBindings, mark, onEnd, onStart, performApplication, performDynamicWebService, performSubflow, performWebService, propagateException, registerActivity, registerActivity, registerTransition, reinit, reset, resume, rollback, setDataStore, setError, setFailureReason, setInterrupted, setUseDataStore, suspend, trace, trace

Methods inherited from class jade.core.behaviours.FSMBehaviour
deregisterDefaultTransition, deregisterState, deregisterTransition, forceTransitionTo, getChildren, getLastExitValue, getName, getPrevious, getState, hasDefaultTransition, registerDefaultTransition, registerDefaultTransition, registerFirstState, registerLastState, registerState, registerTransition, registerTransition, resetStates, scheduleFirst,

scheduleNext, stringifyTransitionTable

Methods inherited from class jade.core.behaviours.SerialBehaviour
handle

Methods inherited from class jade.core.behaviours.CompositeBehaviour
action, done, handleBlockEvent, handleRestartEvent, registerAsChild, resetChildren, setAgent

Methods inherited from class jade.core.behaviours.Behaviour
actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, isRunnable, restart, root, setBehaviourName, setExecutionState

Methods inherited from interface com.tilab.wade.performer.HierarchyNode
getBehaviourName, getDataStore, root

Field Detail

public static final String **FINALIZE_ACTIVITY**

public static final String **UPDATEDOCUMENT_ACTIVITY**

public static final String **WS2CRM_ACTIVITY**

public static final String **OPENSCHEDULE_ACTIVITY**

private String **scheduleImage**

private Vector<String> **customerNames**

private Vector<Integer> **processingTimes**

private CrmResult **res**

private int **numOfCustomers**

private int **currentCustomerIndex**

private String **updatedFileName**

private String **myFile**

Constructor Detail

public **AssistantLaunching**()

Method Detail

private void **defineActivities**()

```
protected void executeOpenSchedule(com.tilab.wade.performer.ApplicationList applications)
                            throws Exception
```

Throws:

Exception

```
protected void executeWS2CRM(com.tilab.wade.performer.WebService ws)
                      throws Exception
```

Throws:

Exception

private void **defineTransitions**()

```
protected void executeUpdateDocument(com.tilab.wade.performer.ApplicationList applications)
                              throws Exception
```

Throws:

Exception

```
protected void executeFinalize()
                        throws Exception
```

Throws:

Exception

protected boolean **checkUpdateDocumentToFinalize**()

```
protected boolean checkOpenScheduleToWS2CRM()
```

Class CreateTAMFile

workflows.auxiliary

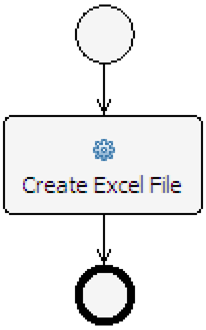
```
java.lang.Object
  └ jade.core.behaviours.Behaviour
      └ jade.core.behaviours.CompositeBehaviour
          └ jade.core.behaviours.SerialBehaviour
              └ jade.core.behaviours.FSMBehaviour
                  └ com.tilab.wade.performer.WorkflowBehaviour
                      └ workflows.auxiliary.CreateTAMFile
```

All Implemented Interfaces:

com.tilab.wade.performer.HierarchyNode, jade.util.leap.Serializable, Serializable

```
public class CreateTAMFile
extends com.tilab.wade.performer.WorkflowBehaviour
```

A workflow class to implement a Subflow. A single-activity workflow, implemented as a subflow and not as an



activity because JOIN gateways are used in the parent process `QuantifyTAM`.

Author:

Pavlos Delias

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
`Behaviour.RunnableChangedEvent`

Field Summary		Page
static String	**CREATETAMFILETOOLACTIVITY1_ACTIVITY**	337

Fields inherited from class com.tilab.wade.performer.WorkflowBehaviour
`COLLECT_ASYNCH_SUBFLOWS_STATE, END_STATE, ERROR_STATE, FINAL, formalParams, INITIAL, INITIAL_AND_FINAL, INTERMEDIATE, lastException, myLogger, rootExecutor, START_STATE`

Fields inherited from class jade.core.behaviours.FSMBehaviour
`currentName, lastStates`

Fields inherited from class jade.core.behaviours.CompositeBehaviour
`currentExecuted`

Fields inherited from class jade.core.behaviours.Behaviour
myAgent, myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING

Constructor Summary	Page
CreateTAMFile()	337

Method Summary		Page
private void	**defineActivities**()	337
protected void	**executeCreateTAMFileToolActivity1**(com.tilab.wade.performer.ApplicationList applications)	338

Methods inherited from class com.tilab.wade.performer.WorkflowBehaviour
changeActivityOrder, checkModifier, checkTermination, commit, configure, deregisterActivity, deregisterTransition, enterInterruptableSection, exitInterruptableSection, fillFormalParameters, fireEvent, getAgent, getBindingManager, getBuildingBlock, getCurrent, getDefaultPriority, getDescriptor, getExecutionContext, getExecutionId, getFormalParameters, getLastErrorEvent, getLastException, getLimit, getModifier, getModifiers, getOutgoingTransitions, getOwner, getRollbackWorkflow, getTracer, getTransactionManager, handleBeginActivity, handleBeginApplication, handleEndActivity, handleEndApplication, handleException, handleInconsistentFSM, handleStateEntered, handleUngroundedParameters, hasJADEDefaultTransition, initRootExecutor, isError, isFireable, isInterrupted, manageBindings, mark, onEnd, onStart, performApplication, performDynamicWebService, performSubflow, performWebService, propagateException, registerActivity, registerActivity, registerTransition, reinit, reset, resume, rollback, setDataStore, setError, setFailureReason, setInterrupted, setUseDataStore, suspend, trace, trace

Methods inherited from class jade.core.behaviours.FSMBehaviour
deregisterDefaultTransition, deregisterState, deregisterTransition, forceTransitionTo, getChildren, getLastExitValue, getName, getPrevious, getState, hasDefaultTransition, registerDefaultTransition, registerDefaultTransition, registerFirstState, registerLastState, registerState, registerTransition, registerTransition, resetStates, scheduleFirst, scheduleNext, stringifyTransitionTable

Methods inherited from class jade.core.behaviours.SerialBehaviour
handle

Methods inherited from class jade.core.behaviours.CompositeBehaviour
action, done, handleBlockEvent, handleRestartEvent, registerAsChild, resetChildren, setAgent

Methods inherited from class jade.core.behaviours.Behaviour
actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, isRunnable, restart, root, setBehaviourName, setExecutionState

Methods inherited from interface com.tilab.wade.performer.HierarchyNode
getBehaviourName, getDataStore, root

Field Detail

public static final String **CREATETAMFILETOOLACTIVITY1_ACTIVITY**

Constructor Detail

public **CreateTAMFile**()

Method Detail

private void **defineActivities**()

```
protected void executeCreateTAMFileToolActivity1(com.tilab.wade.performer.ApplicationList appl
ications)
                                                  throws Exception
```

Throws:
    Exception

Class FetchPptFile

[workflows.auxiliary]

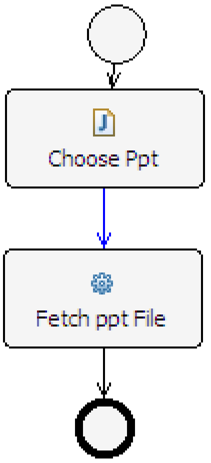
```
java.lang.Object
  └ jade.core.behaviours.Behaviour
      └ jade.core.behaviours.CompositeBehaviour
          └ jade.core.behaviours.SerialBehaviour
              └ jade.core.behaviours.FSMBehaviour
                  └ com.tilab.wade.performer.WorkflowBehaviour
                      └ workflows.auxiliary.FetchPptFile
```

All Implemented Interfaces:
com.tilab.wade.performer.HierarchyNode, jade.util.leap.Serializable, Serializable

---

```
public class FetchPptFile
extends com.tilab.wade.performer.WorkflowBehaviour
```

A workflow class to implement a Subflow. A simple, two-activities workflow, implemented as a subflow and not as an activity because JOIN gateways are used in the parent process `QuantifyTAM`.



Author:
Pavlos Delias

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
Behaviour.RunnableChangedEvent

Field Summary		Page
static String	CHOOSEPPT_ACTIVITY	340
static String	FETCHPPTFILETOOLACTIVITY1_ACTIVITY	340
private File	ppt	340
private static long	serialVersionUID	340

Fields inherited from class com.tilab.wade.performer.WorkflowBehaviour
COLLECT_ASYNCH_SUBFLOWS_STATE, END_STATE, ERROR_STATE, FINAL, formalParams, INITIAL, INITIAL_AND_FINAL, INTERMEDIATE, lastException, myLogger, rootExecutor, START_STATE

Fields inherited from class jade.core.behaviours.FSMBehaviour
currentName, lastStates

Fields inherited from class jade.core.behaviours.CompositeBehaviour
currentExecuted

Fields inherited from class jade.core.behaviours.Behaviour
myAgent, myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING

Constructor Summary	Page
FetchPptFile()	340

Method Summary		Page
private void	**defineActivities**()	340
private void	**defineTransitions**()	340
protected void	**executeChoosePpt**()	340
protected void	**executeFetchPptFileToolActivity1**(com.tilab.wade.performer.ApplicationList applications)	340

Methods inherited from class com.tilab.wade.performer.WorkflowBehaviour
changeActivityOrder, checkModifier, checkTermination, commit, configure, deregisterActivity, deregisterTransition, enterInterruptableSection, exitInterruptableSection, fillFormalParameters, fireEvent, getAgent, getBindingManager, getBuildingBlock, getCurrent, getDefaultPriority, getDescriptor, getExecutionContext, getExecutionId, getFormalParameters, getLastErrorEvent, getLastException, getLimit, getModifier, getModifiers, getOutgoingTransitions, getOwner, getRollbackWorkflow, getTracer, getTransactionManager, handleBeginActivity, handleBeginApplication, handleEndActivity, handleEndApplication, handleException, handleInconsistentFSM, handleStateEntered, handleUngroundedParameters, hasJADEDefaultTransition, initRootExecutor, isError, isFireable, isInterrupted, manageBindings, mark, onEnd, onStart, performApplication, performDynamicWebService, performSubflow, performWebService, propagateException, registerActivity, registerActivity, registerTransition, reinit, reset, resume, rollback, setDataStore, setError, setFailureReason, setInterrupted, setUseDataStore, suspend, trace, trace

Methods inherited from class jade.core.behaviours.FSMBehaviour
deregisterDefaultTransition, deregisterState, deregisterTransition, forceTransitionTo, getChildren, getLastExitValue, getName, getPrevious, getState, hasDefaultTransition, registerDefaultTransition, registerDefaultTransition, registerFirstState, registerLastState, registerState, registerTransition, registerTransition, resetStates, scheduleFirst, scheduleNext, stringifyTransitionTable

Methods inherited from class jade.core.behaviours.SerialBehaviour
handle

Methods inherited from class jade.core.behaviours.CompositeBehaviour
action, done, handleBlockEvent, handleRestartEvent, registerAsChild, resetChildren, setAgent

Methods inherited from class jade.core.behaviours.Behaviour
`actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, isRunnable, restart, root, setBehaviourName, setExecutionState`

Methods inherited from interface com.tilab.wade.performer.HierarchyNode
`getBehaviourName, getDataStore, root`

Field Detail

`public static final String` **`CHOOSEPPT_ACTIVITY`**

`private static final long` **`serialVersionUID`**

`private File` **`ppt`**

`public static final String` **`FETCHPPTFILETOOLACTIVITY1_ACTIVITY`**

Constructor Detail

`public` **`FetchPptFile`**`()`

Method Detail

`private void` **`defineActivities`**`()`

```
protected void executeFetchPptFileToolActivity1(com.tilab.wade.performer.ApplicationList appli
cations)
                                         throws Exception
```

Throws:

`Exception`

```
protected void executeChoosePpt()
                         throws Exception
```

Throws:

`Exception`

`private void` **`defineTransitions`**`()`

Class ProcessBatchMail

workflows.auxiliary

```
java.lang.Object
  └ jade.core.behaviours.Behaviour
      └ jade.core.behaviours.CompositeBehaviour
          └ jade.core.behaviours.SerialBehaviour
              └ jade.core.behaviours.FSMBehaviour
                  └ com.tilab.wade.performer.WorkflowBehaviour
                      └ workflows.auxiliary.ProcessBatchMail
```

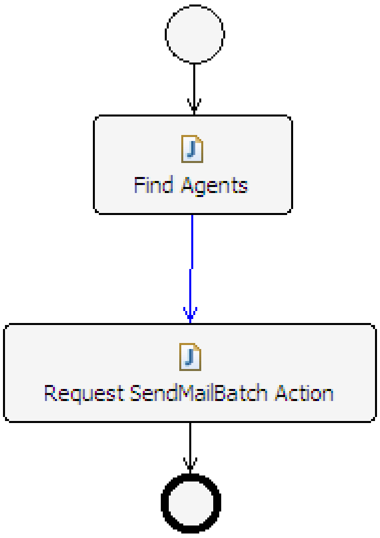
All Implemented Interfaces:

com.tilab.wade.performer.HierarchyNode, jade.util.leap.Serializable, Serializable

---

```
public class ProcessBatchMail
extends com.tilab.wade.performer.WorkflowBehaviour
```

A workflow class implemented as an intermediate step of the Contact Center Management process.



Author:
Pavlos Delias

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
`Behaviour.RunnableChangedEvent`

Field Summary		Page
static String	**FINDAGENTS_ACTIVITY**	342
static String	**SENDMAILBATCHREQUEST_ACTIVITY**	342
private jade.core.AID	**toAgent**	342

Fields inherited from class com.tilab.wade.performer.WorkflowBehaviour
`COLLECT_ASYNCH_SUBFLOWS_STATE, END_STATE, ERROR_STATE, FINAL, formalParams, INITIAL, INITIAL_AND_FINAL, INTERMEDIATE, lastException, myLogger, rootExecutor, START_STATE`

Fields inherited from class jade.core.behaviours.FSMBehaviour
`currentName, lastStates`

Fields inherited from class jade.core.behaviours.CompositeBehaviour
`currentExecuted`

Fields inherited from class jade.core.behaviours.Behaviour
`myAgent, myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING`

Constructor Summary	Page
ProcessBatchMail()	342

Method Summary		Page
private void	**defineActivities**()	342

private void	**defineTransitions**()	343
protected void	**executeFindAgents**() Gets the reference for the Assignment Agent.	342
protected void	**executeSendMailBatchRequest**()	343

Methods inherited from class com.tilab.wade.performer.WorkflowBehaviour
changeActivityOrder, checkModifier, checkTermination, commit, configure, deregisterActivity, deregisterTransition, enterInterruptableSection, exitInterruptableSection, fillFormalParameters, fireEvent, getAgent, getBindingManager, getBuildingBlock, getCurrent, getDefaultPriority, getDescriptor, getExecutionContext, getExecutionId, getFormalParameters, getLastErrorEvent, getLastException, getLimit, getModifier, getModifiers, getOutgoingTransitions, getOwner, getRollbackWorkflow, getTracer, getTransactionManager, handleBeginActivity, handleBeginApplication, handleEndActivity, handleEndApplication, handleException, handleInconsistentFSM, handleStateEntered, handleUngroundedParameters, hasJADEDefaultTransition, initRootExecutor, isError, isFireable, isInterrupted, manageBindings, mark, onEnd, onStart, performApplication, performDynamicWebService, performSubflow, performWebService, propagateException, registerActivity, registerActivity, registerTransition, reinit, reset, resume, rollback, setDataStore, setError, setFailureReason, setInterrupted, setUseDataStore, suspend, trace, trace

Methods inherited from class jade.core.behaviours.FSMBehaviour
deregisterDefaultTransition, deregisterState, deregisterTransition, forceTransitionTo, getChildren, getLastExitValue, getName, getPrevious, getState, hasDefaultTransition, registerDefaultTransition, registerDefaultTransition, registerFirstState, registerLastState, registerState, registerTransition, registerTransition, resetStates, scheduleFirst, scheduleNext, stringifyTransitionTable

Methods inherited from class jade.core.behaviours.SerialBehaviour
handle

Methods inherited from class jade.core.behaviours.CompositeBehaviour
action, done, handleBlockEvent, handleRestartEvent, registerAsChild, resetChildren, setAgent

Methods inherited from class jade.core.behaviours.Behaviour
actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, isRunnable, restart, root, setBehaviourName, setExecutionState

Methods inherited from interface com.tilab.wade.performer.HierarchyNode
getBehaviourName, getDataStore, root

Field Detail

private jade.core.AID **toAgent**

---

public static final String **SENDMAILBATCHREQUEST_ACTIVITY**

---

public static final String **FINDAGENTS_ACTIVITY**

Constructor Detail

public **ProcessBatchMail**()

Method Detail

private void **defineActivities**()

---

protected void **executeFindAgents**()
throws Exception

Gets the reference for the Assignment Agent.

Throws:
`Exception`

```
protected void executeSendMailBatchRequest()
                                    throws Exception
```

Throws:
`Exception`

```
private void defineTransitions()
```

Class SpectralScheduling

[workflows.auxiliary]

```
java.lang.Object
  └ jade.core.behaviours.Behaviour
      └ jade.core.behaviours.CompositeBehaviour
          └ jade.core.behaviours.SerialBehaviour
              └ jade.core.behaviours.FSMBehaviour
                  └ com.tilab.wade.performer.WorkflowBehaviour
                      └ workflows.auxiliary.SpectralScheduling
```

All Implemented Interfaces:
com.tilab.wade.performer.HierarchyNode, jade.util.leap.Serializable, Serializable

---

```
public class SpectralScheduling
extends com.tilab.wade.performer.WorkflowBehaviour
```

A workflow class to orchestrate the application of a scheduling algorithm.

Author:
Pavlos Delias

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
`Behaviour.RunnableChangedEvent`

Field Summary		Page
private int	counter	345
private jade.core.AID	currentAgent	345
static String	FINALIZE_ACTIVITY	345
static String	FINDPERAGENT_ACTIVITY	345
static String	FINDPERAGENTTOFINALIZE_CONDITION	345
static String	LOOP_ACTIVITY	345
static String	SPECTRALSCHEDULINGTOOL_ACTIVITY	345
private double[][]	taskStartTimes	345
private double[][]	taskToAgents	345

Fields inherited from class com.tilab.wade.performer.WorkflowBehaviour
`COLLECT_ASYNCH_SUBFLOWS_STATE, END_STATE, ERROR_STATE, FINAL, formalParams, INITIAL, INITIAL_AND_FINAL, INTERMEDIATE, lastException, myLogger, rootExecutor, START_STATE`

Fields inherited from class jade.core.behaviours.FSMBehaviour
currentName, lastStates

Fields inherited from class jade.core.behaviours.CompositeBehaviour
currentExecuted

Fields inherited from class jade.core.behaviours.Behaviour
myAgent, myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING

Constructor Summary	Page
SpectralScheduling()	345

Method Summary		Page
protected boolean	**checkFindPerAgentToFinalize**()	345
private void	**defineActivities**()	345
private void	**defineTransitions**()	345
protected void	**executeFinalize**()	345
protected void	**executeFindPerAgent**(com.tilab.wade.performer.ApplicationList applications)	345
protected void	**executeLoop**()	345
protected void	**executeSpectralSchedulingTool**(com.tilab.wade.performer.ApplicationList applications) This tool calls MATLAB to solve the scheduling algorithm.	345

Methods inherited from class com.tilab.wade.performer.WorkflowBehaviour
changeActivityOrder, checkModifier, checkTermination, commit, configure, deregisterActivity, deregisterTransition, enterInterruptableSection, exitInterruptableSection, fillFormalParameters, fireEvent, getAgent, getBindingManager, getBuildingBlock, getCurrent, getDefaultPriority, getDescriptor, getExecutionContext, getExecutionId, getFormalParameters, getLastErrorEvent, getLastException, getLimit, getModifier, getModifiers, getOutgoingTransitions, getOwner, getRollbackWorkflow, getTracer, getTransactionManager, handleBeginActivity, handleBeginApplication, handleEndActivity, handleEndApplication, handleException, handleInconsistentFSM, handleStateEntered, handleUngroundedParameters, hasJADEDefaultTransition, initRootExecutor, isError, isFireable, isInterrupted, manageBindings, mark, onEnd, onStart, performApplication, performDynamicWebService, performSubflow, performWebService, propagateException, registerActivity, registerActivity, registerTransition, reinit, reset, resume, rollback, setDataStore, setError, setFailureReason, setInterrupted, setUseDataStore, suspend, trace, trace

Methods inherited from class jade.core.behaviours.FSMBehaviour
deregisterDefaultTransition, deregisterState, deregisterTransition, forceTransitionTo, getChildren, getLastExitValue, getName, getPrevious, getState, hasDefaultTransition, registerDefaultTransition, registerDefaultTransition, registerFirstState, registerLastState, registerState, registerTransition, registerTransition, resetStates, scheduleFirst, scheduleNext, stringifyTransitionTable

Methods inherited from class jade.core.behaviours.SerialBehaviour
handle

Methods inherited from class jade.core.behaviours.CompositeBehaviour
action, done, handleBlockEvent, handleRestartEvent, registerAsChild, resetChildren, setAgent

Methods inherited from class jade.core.behaviours.Behaviour
`actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, isRunnable, restart, root, setBehaviourName, setExecutionState`

Methods inherited from interface com.tilab.wade.performer.HierarchyNode
`getBehaviourName, getDataStore, root`

Field Detail

`public static final String` **`FINDPERAGENTTOFINALIZE_CONDITION`**

`public static final String` **`FINALIZE_ACTIVITY`**

`public static final String` **`FINDPERAGENT_ACTIVITY`**

`public static final String` **`LOOP_ACTIVITY`**

`public static final String` **`SPECTRALSCHEDULINGTOOL_ACTIVITY`**

`private double[][]` **`taskStartTimes`**

`private double[][]` **`taskToAgents`**

`private int` **`counter`**

`private jade.core.AID` **`currentAgent`**

Constructor Detail

`public` **`SpectralScheduling`**`()`

Method Detail

`private void` **`defineActivities`**`()`

```
protected void executeSpectralSchedulingTool(com.tilab.wade.performer.ApplicationList applicat
ions)
                                      throws Exception
```

This tool calls MATLAB to solve the scheduling algorithm.

Throws:
`Exception`

```
protected void executeLoop()
                    throws Exception
```

Throws:
`Exception`

```
protected void executeFindPerAgent(com.tilab.wade.performer.ApplicationList applications)
                            throws Exception
```

Throws:
`Exception`

```
protected void executeFinalize()
                        throws Exception
```

Throws:
`Exception`

`private void` **`defineTransitions`**`()`

```
protected boolean checkFindPerAgentToFinalize()
                                       throws Exception
```

Throws:
`Exception`

Class VendorOffer

workflows.auxiliary

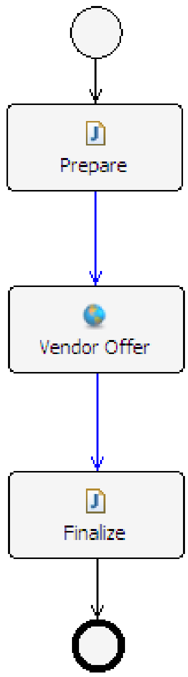
```
java.lang.Object
  └ jade.core.behaviours.Behaviour
      └ jade.core.behaviours.CompositeBehaviour
          └ jade.core.behaviours.SerialBehaviour
              └ jade.core.behaviours.FSMBehaviour
                  └ com.tilab.wade.performer.WorkflowBehaviour
                      └ workflows.auxiliary.VendorOffer
```

All Implemented Interfaces:

com.tilab.wade.performer.HierarchyNode, jade.util.leap.Serializable, Serializable

---

```
public class VendorOffer
extends com.tilab.wade.performer.WorkflowBehaviour
```



A workflow class to model the vendors inner behavior

Author:

Pavlos Delias

Nested classes/interfaces inherited from class jade.core.behaviours.Behaviour
Behaviour.RunnableChangedEvent

Field Summary		Page
static String	ACTUALWS_ACTIVITY	348
static String	FINALIZEWSCALL_ACTIVITY	348

private MediaFormat	**format**	***Error! Bookmark not defined.***
static String	**PREPAREWSCALL_ACTIVITY**	*348*

Fields inherited from class com.tilab.wade.performer.WorkflowBehaviour
COLLECT_ASYNCH_SUBFLOWS_STATE, END_STATE, ERROR_STATE, FINAL, formalParams, INITIAL, INITIAL_AND_FINAL, INTERMEDIATE, lastException, myLogger, rootExecutor, START_STATE

Fields inherited from class jade.core.behaviours.FSMBehaviour
currentName, lastStates

Fields inherited from class jade.core.behaviours.CompositeBehaviour
currentExecuted

Fields inherited from class jade.core.behaviours.Behaviour
myAgent, myEvent, NOTIFY_DOWN, NOTIFY_UP, parent, STATE_BLOCKED, STATE_READY, STATE_RUNNING

Constructor Summary	*Page*
VendorOffer()	*348*

Method Summary		*Page*
private void	**defineActivities**()	*348*
private void	**defineTransitions**()	*348*
protected void	**executeActualWS**(com.tilab.wade.performer.WebService ws)	*348*
protected void	**executeFinalizeWSCall**()	*348*
protected void	**executePrepareWSCall**()	*348*

Methods inherited from class com.tilab.wade.performer.WorkflowBehaviour
changeActivityOrder, checkModifier, checkTermination, commit, configure, deregisterActivity, deregisterTransition, enterInterruptableSection, exitInterruptableSection, fillFormalParameters, fireEvent, getAgent, getBindingManager, getBuildingBlock, getCurrent, getDefaultPriority, getDescriptor, getExecutionContext, getExecutionId, getFormalParameters, getLastErrorEvent, getLastException, getLimit, getModifier, getModifiers, getOutgoingTransitions, getOwner, getRollbackWorkflow, getTracer, getTransactionManager, handleBeginActivity, handleBeginApplication, handleEndActivity, handleEndApplication, handleException, handleInconsistentFSM, handleStateEntered, handleUngroundedParameters, hasJADEDefaultTransition, initRootExecutor, isError, isFireable, isInterrupted, manageBindings, mark, onEnd, onStart, performApplication, performDynamicWebService, performSubflow, performWebService, propagateException, registerActivity, registerActivity, registerTransition, reinit, reset, resume, rollback, setDataStore, setError, setFailureReason, setInterrupted, setUseDataStore, suspend, trace, trace

Methods inherited from class jade.core.behaviours.FSMBehaviour
`deregisterDefaultTransition, deregisterState, deregisterTransition, forceTransitionTo, getChildren, getLastExitValue, getName, getPrevious, getState, hasDefaultTransition, registerDefaultTransition, registerDefaultTransition, registerFirstState, registerLastState, registerState, registerTransition, registerTransition, resetStates, scheduleFirst, scheduleNext, stringifyTransitionTable`

Methods inherited from class jade.core.behaviours.SerialBehaviour
`handle`

Methods inherited from class jade.core.behaviours.CompositeBehaviour
`action, done, handleBlockEvent, handleRestartEvent, registerAsChild, resetChildren, setAgent`

Methods inherited from class jade.core.behaviours.Behaviour
`actionWrapper, block, block, getBehaviourName, getDataStore, getExecutionState, getParent, getRestartCounter, isRunnable, restart, root, setBehaviourName, setExecutionState`

Methods inherited from interface com.tilab.wade.performer.HierarchyNode
`getBehaviourName, getDataStore, root`

Field Detail

private MediaFormat **format**

---

public static final String **ACTUALWS_ACTIVITY**

---

public static final String **FINALIZEWSCALL_ACTIVITY**

---

public static final String **PREPAREWSCALL_ACTIVITY**

Constructor Detail

public **VendorOffer**()

Method Detail

private void **defineActivities**()

---

protected void **executePrepareWSCall**()
throws Exception

Throws:
Exception

---

protected void **executeFinalizeWSCall**()
throws Exception

Throws:
Exception

---

private void **defineTransitions**()

---

protected void **executeActualWS**(com.tilab.wade.performer.WebService ws)
throws Exception

Throws:
Exception