applied sciences

MDPI

Article

Artificial Intelligence-Driven Composition and Security Validation of an Internet of Things Ecosystem

George Hatzivasilis [1,2,*], Nikos Papadakis [3], Ilias Hatzakis [3], Sotiris Ioannidis [1,2] and George Vardakis [3]

1 Foundation for Research and Technology–Hellas, Institute of Computer Science, 70013 Vassilika Vouton, Greece; sotiris@ics.forth.gr or sotiris@ece.tuc.gr
2 Electrical and Computer Engineering, Technical University of Crete, 73100 Akrotiri Campus, Greece
3 Electrical and Computer Engineering, Hellenic Mediterranean University (HMU), 71410 Estavromenos, Greece; npapadak@cs.hmu.gr (N.P.); hatzakis@cs.teicrete.gr (I.H.); gvardakis@cs.hmu.gr (G.V.)
* Correspondence: hatzivas@ics.forth.gr; Tel.: +30-2810-391600

Received: 29 May 2020; Accepted: 11 July 2020; Published: 15 July 2020

Abstract: Key challenges in Internet-of-Things (IoT) system design and management include the secure system composition and the calculation of the security and dependability level of the final system. This paper presents an event-based model-checking framework for IoT systems' design and management, called CompoSecReasoner. It invokes two main functionalities: (i) system composition verification, and (ii) derivation and validation of security, privacy, and dependability (SPD) metrics. To measure the SPD values of a system, we disassemble two well-known types of security metrics—the attack surface methodologies and the medieval castle approach. The first method determines the attackable points of the system, while the second one defines the protection level that is provided by the currently composed system-of-systems. We extend these techniques and apply the Event Calculus method for modelling the dynamic behavior of a system with progress in time. At first, the protection level of the currently composed system is calculated. When composition events occur, the current system status is derived. Thereafter, we can deploy reactive strategies and administrate the system automatically at runtime, implementing a novel setting for Moving Target Defenses. We demonstrate the overall solution on a real ambient intelligence application for managing the embedded devices of two emulated smart buildings.

Keywords: dependability; dynamic system composition; event calculus; Internet-of-Things; IoT; JADE; JESS; metrics; OSGi; Moving Target Defenses

1. Introduction

In this era of Internet-of-Things (IoT) and the 4th Industrial Revolution, intelligence is integrated into ordinary things. These smart devices are composed in complex systems, forming ambient intelligence and pervasive computing applications (e.g., [1–3]). Case studies include among other assisting living, smart transportation, and e-health. Thus, design technologies are becoming imperative in order to guarantee the desired requirements of a heterogeneous system-of-systems, like security, privacy, and dependability (SPD).

During the lifecycle of a system, quantitating and measuring its features plays a significant part of the applied risk analysis. Industrial and scientific enterprises, such as the Ford Motor Company [4] and NASA [5], apply formal techniques to verify that the implemented systems and their counterparts guarantee the mission critical properties and achieve the intended design goals.

Successful cyber-attacks in actual systems derive security in a core design property [6,7]. Nowadays, privacy is also coming in the foreground [8–10], due to the processing of vast amounts of personal data, for example in IoT and cloud computing settings. Nevertheless, the simple deployment of security and privacy protection is not enough. Dependability of the underlying mechanisms has to be also taken into account [11–13].

However, it is still not easy to estimate in a practical and systematic manner, the overall security, privacy, and dependability aspects. With the adoption of ubiquitous and pervasive computing, computational resources are composed in a dynamic fashion, enabling the interaction between a high number of heterogeneous embedded and mobile devices. Therefore, the problem is becoming even more difficult to handle when we have to face such a dynamically composed ecosystem.

In this paper, we implement CompoSecReasoner—a methodology for the description of the SPD aspects of composed systems, and the effects of changes in the state-architecture. CompoSecReasoner is appropriate for SPD verification, composition validation, comparison between different system configurations, impact assessment of changes in the system, and materialization of automated reactive strategies. Furthermore, the developed CompoSecReasoner framework is utilized as middleware of IoT applications, providing real-time monitoring and administration.

CompoSecReasoner addresses four core elements in system development:

1. **Extension with metrics of the system composition and management procedures.** This enables the calculation of the SPD for the currently composed system. Metrics are evolving as an integral characteristic of systems' development as they are utilized for the comparison of the various possible system configurations, as well as, the impact quantification of the ongoing changes in the main setting.
2. **Verification that the various components can be composed into the examined system**. This becomes imperative for heterogeneous environments (settings with different device types) and system-of-systems (a system comprised of many subsystems).
3. **Validation of properties that hold after the composition of two systems**. This capability is useful in order to assure that the finally composed system fulfills the designed specifications and works correctly.
4. **Automated reactive strategies and moving-target defenses (MTDs)**. The IoT ecosystem is dominant by machine-to-machine (M2M) communications. Therefore, we need means to make the overall system self-adaptable to changes, failures, and/or other ongoing events. MTD is also such an approach in the cyber-security field [14]. MTDs include proactive altering of the system configuration, networking, and/or topology to increase the malicious effort for analyzing the system in normal operation, as well as, reactive operations to respond to specific types of ongoing attacks.

Technically, CompoSecReasoner is developed as the reasoning behavior of Java Agent DEvelopment framework (JADE) [15] agents and deployed in the middleware platform Open Services Gateway initiative (OSGi) [16]. The IoT devices implement OSGi bundles, which communicate information with the JADE bundles, and describe their provided services in DPWS [17]. The implementation is installed in a smart home application to assess and administrate the underlying smart devices with ambient intelligence capabilities for assisting living.

The rest of this study is structured as follows: Section 2 presents related works, Section 3 describes the fundamental metrics and measurement methodologies that are embodied in CompoSecReasoner, Section 4 analyses the basic concepts of the framework, Section 5 details the systematic validation definitions, Section 6 presents the implementation aspects, Section 7 discusses the validity of the proposed SPD assessment and refers future work and possible extensions, and the conclusions follow in Section 8. Also, Appendix A details the protection features that are evaluated by the SPD metrics.

2. Related Work

2.1. System Composition

Nowadays more than ever, secure system design becomes imperative as well as the proof that an integrated system is working properly. Reviews of composition verification methodologies for IoT are presented in [18,19]. In general, such procedures are modelled either with Unified Modeling Language (UML) [20] or logic-based techniques. Moreover, two main validation types are used. Pre-composition validation checks policies or rules during a composition event (e.g., interface compliance and runtime enforcement). If the relevant policies and/or rules are satisfied, the composition is completed. Therefore, we expect that the composed setting should exhibit some desirable properties. On the other hand, post-composition validation examines the system after the composition event and tries to disclose which properties (e.g., security) are satisfied by the current setting. More advanced techniques utilize both validation types.

2.1.1. UML-Based Proposals

UML-based solutions provide a user-friendly representation of the system and are mainly covering the initial stages of the design phase. The Interaction Flow Modeling Language (IFML) [21], standardized by the Object Management Group [22], utilizes the main data types of the UML meta-model and meta-classes, and can model the user interfaces (UI) of IoT-based applications. The work in [23] utilizes IFML to further define design patterns for a set of common user interactions for incorporated applications. Exploiting the UML features and available tools, the abstract code for these interfaces can be also generated by the graphical designs. However, such solutions represent only static instances for a system and their ability to assess a dynamically composed settings is limited.

2.1.2. Logic-Based Proposals

On the contrary, the logic-based proposals are more suitable for dynamic verification, as they can evaluate various system states and their change over time. The theoretic foundations are surveyed in [24]. From the various approaches, the model-based solutions meet the design challenges and are the most promising. A key advantage is the abstraction level that is provided for an examined application domain. Therefore, they offer a set of concepts which are customized to specific application fields.

Historically, the Model Integrated Computing (MIC) [25] was designed for formal analysis, validation, verification, and development of embedded systems. It utilizes the Domain Specific Modeling Language (DSML) [26], which can be also expressed by UML meta-models, to provide the abstraction in an application domain. Such general composition proposals are presented in [27–30]. The Society of Automotive Engineers (SAE) [27] standardized the Architecture Analysis and Design Language (AADL) [28] for modeling and systems' analysis. It can express the hardware/software layers of embedded systems while assuring real-time requirements, such as fault-tolerance, safety, power consumption, schedulability, etc. With AADL the underlying setting is disassembled in hardware, software, or system components. *Interface structures* determine the externally disclosed features, such as utilized ports, while the *implementation structures* determine the inner elements, such as the sub-components for a main component and their connections. Thereupon, several modern AADL-based solutions or other model-driven settings (e.g., [29–34]) have been proposed.

SMoLES [30] uses DSML to express in an abstract way the system's configurations and provide a concise syntax for embedded systems' implementation. System components are defined as objects which are synchronized with each other and exchange information. Components constitute the building blocks for a system, containing input/output ports to receive/send data, accordingly. An assembly is comprised of components, presenting how they are connected.

Dracena [31] is an IoT data processing platform. It models the components interactive modules as plug-ins of data-processing logic and the composition as distributed stream processing in the level of

these components and their involved plug-ins. Dracena supports pre-composition verification and has been successfully applied in the IoT applications for smart vehicles.

The proposal in [32] models the interaction of IoT systems by analyzing Message Sequence Charts (MSCs) for the utilized messaging protocol. Its current version verifies the composition of systems that use the protocol Constrained Application Protocol (CoAP) [33]. Post-composition validation is performed afterwards. A packet sniffer records the communication of the network for a specified period. Then, based on the context knowledge of the messaging protocol the system creates the MSCs and based on their analysis it derives if the compositions comply with the design specifications or not. However, as the overall solution has many drawbacks when it comes on the post-composition of actual systems as it is not generic and it requires significant resources and effort.

BLESS [34] constitutes a behavioral interface specification language and proof environment for AADL. The components create AADL contracts, capturing timing and functional aspects. The pre-composition verification is performed by checking the compatibility of two interfaces while post-composition validation is achieved via the related composition contracts that the incorporated components must fulfill.

The IoT Composer [35] models the behavioral aspects of IoT objects and their compositions, and performs pre-composition verification that the integration can be achieved as well as post-composition validation that the integrated components are co-working properly. Compositions are defined as interface bindings. Post-composition validation is determined in the form of deployment plans that are executed by the tool to configure and run the composed application, i.e., check if the composition is free of deadlocks.

2.2. Security Validation

However, the aforementioned general composition methods do not take into consideration the special intrinsic of security. Theoretical works for secure system composition are presented in [36–39]. In [36], the authors document a set of composition theorems which can prove protocols' security under standard stand-alone and compositional definitions of security. The study in [37] documents a formal validation of automated policy refinement. The analysis evaluates the management procedures in network security systems. It formally validates policy hierarchies for model-based management and propagates from an abstract level to its immediate lower neighbor. The outcomes are two theorems that ensure the compliance among the actual system behavior and the abstract design policies. In [38], methods of concurrent general composition and universal composability are examined on public-key models and scenarios with no trust infrastructure. Newer studies [39] update the security composition theory with properties like mitigation of eavesdropping and identity faking. Therefore, a modeling framework is proposed [39] for security verification and is applied in distributed and industrial IoT infrastructures. The framework performs model-checking techniques and the overall analysis is supported by the Alloy Analyzer [40]. Yet, theoretical works cannot be immediately applied in real and dynamic systems. Nevertheless, the deduced security and composition outcomes are considered by practical solutions and developed tools for more robust formal documentation.

Also, in the field of Knowledge Representation and Reasoning event-driven model-based approaches are suggested to better express the dynamic behavior of the IoT ecosystem in a formal manner. Thus, security verification techniques are additionally proposed to tackle this issue (e.g., [41–44]).

SMoLES-SEC [41] integrates the Security Model Analysis Language (SMAL) [41] to the pure SMoLES. SMAL embodies security extensions to the DSML's meta-model composition and defines access control policies for IoT settings. However, the reasoning capabilities are bounded due to this constrained expressiveness. Also, SMoLES-SEC fails to define the security features which are valid after the composition as well as the overall security status of the composed setting.

The study in [42] proposes a framework for runtime enforcement mechanisms. An analyzed system is modeled as an automaton. Then, the framework can track a sequence of states and events to derive

wherever the security policy constraints are satisfied at runtime. The enforcement mechanism develops two main operations: (i) allows legal functionality without changes and (ii) blocks illegal functionality.

A modeling and visualization tool for security evaluation and secure system management is presented in [43]. Threat modeling is applied in the composed setting for the evaluation of metrics at the architectural level. Then, hierarchical presentation of security metrics is performed based on decomposition methods for the measured security objectives, while deriving the overall security status.

The work in [44] proposes IoT service secure composition verification based on Service Dependency Trees (SDTs). Each IoT device node builds its own SDT, denoting the external service nodes that the provided services are depending on. Also, each node may be aware of all recursive SDTs for its composed services. Henceforth, secure service composition can be assessed by permitting only integration with SDTs were all the paths and involved entities are considered trusted. However, constructing a SDT for a real IoT system is not ease, as well as, their consistency and trustworthiness in an actual complex and dynamically composed setting become challenging.

2.3. Quantifying Security

To further quantify the security of an integrated setting, we have to measure the security of the underlying components and afterwards calculate the total protection. One main methodology to measure the provided security for a system is the medieval castle model [45–47]. The system is considered as a castle with doors that the attacker tries to break through. The infiltration is successful, when the attacker passes the doors and reaches the protected treasure rooms (e.g., assets which the deploy defense mechanisms try to safeguard) in the inner layers of the castle/system. The difficulty of bypassing these doors and reach the inner treasure rooms designates the provided security level for this castle/system. A door represents a security mechanism with the resilience to assaults being assessed by related metrics. The overall method can evaluate the defense of static system instances. However, it is not directly applicable for dynamic system analysis.

Other approaches include the attack surface metrics [48–53] and multi-metric approaches [4,54]. A literature review for attack surface studies is conducted in [48]. Such methodologies (e.g., [49–53]) take into account that the attacker will utilize the system's channels, methods, and data elements to infiltrate the system. The set of these elements at the entry/exit ports which could be exploited by attackers, define the system's attack surface. Quantitative methods assess the damage potential-effort, determining the surface's size. In [4,54], multiply quantitative metrics are incorporated and capture the system's protection aspects. The various metrics take the form of vectors with each factor evaluates a specific property, such as Security, Privacy, Dependability, and/or Usability. The measured values are in the range of 0–100 (modelling no to optimal satisfaction) and easy the composition of several metrics as well their comparison. Thereby, meta-heuristic methods are applied to assess and visualize the offered parameters of the current system setting.

Moving Target Defenses (MTDs) [55] constitute an active and offensive protection strategy where the system configuration and setting is changed, either periodically or as a response to an occurred event, in order to harden the analysis of the system by an attacker or to mitigate ongoing attacks. Such solutions can incorporate security metrics in order to better evaluate the protection level of each potential system state. The study in [56] implements MTDs by exploiting the attack surface metrics. The proposed framework measures the distance of the security surface between different system states. The goal is to administrate responses to attackers' probes, so as to induce an external view of the system that complies with specific desirable properties. Minimizing the cost for the defender is also considered the study in [57] presents a multi-metric-driven management framework for e-health digital environments. The method is applied in e-health settings for chronic diseases and utilizes three metric categories. Risk-driven security engineering and assurance metrics are established at deployment-time, providing an early evaluation on the effectiveness of the deployed security. Continuous security monitoring metrics get assigned at operational-time and support the security correctness assessment, improved systematization as well as traceability among the various metric outcomes and product

requirements. Then, automated adaptive decision-making metrics can be estimated at operational-time and achieve higher quality security effectiveness understanding in operational security monitoring and future versioning of the examined setting. The framework performs continuous security monitoring along with metric-driven fully- or semi- automated security decisions.

A first theoretical work for the measurement of security is conducted in [58]. The authors document a formal model for the representation and evaluation of security metrics and define a set of relevant metrics. Therefore, they examine associations to derive more secure metrics. In contrast to the above-mentioned practical solutions (e.g., [45–57]), which take into consideration only the structural aspects of the system, the authors conclude that a potential metric can be assumed good or bad based also on the attacker model.

2.4. Comparison

In this work, we present the CompoSecReasoner framework that monitors system and the ongoing changes, and evaluates the security, privacy, and dependability status. CompoSecReasoner utilizes an event and model-based approach and it implements dynamic system composition verification, properties validation, and automated administration based on metrics. As we know, no such method has been developed so far, combining the four design operations which are described in the introductory sections.

Related composition and management techniques are either general methodologies [28–35] or cover specific security issues [36–58], none of which fulfils the targeted design requirements. For most composition techniques (e.g., [28,30,52]), the composition verification is modelled as interfaces, inputs/outputs, or plug-ins of the composing components. Then, validation methodologies can verify security properties based on theorems, polices, or contracts (e.g., [44–58]). Thereupon, methods that quantify security can evaluate the finally composed system (e.g., [45,52]). Metric-driven administration and MTDs assess the possible system states and defend the system in real-time [56,57].

For the composition verification procedure, CompoSecReasoner presents a novel feature, defined as *operation* (see Section 4.3). An operation is a series of actions (such as protocols and technologies) that two components have to execute in order to be incorporated and work together. The abstraction level that is offered under this formalism is sufficient for modelling an actual application domain, in contrast to the interface and input/output proposals which are becoming quite modelling-intensive in the field of heterogeneous embedded systems.

For system properties validation, related methodologies apply either pre- or post-validation techniques, modelling a relevant sub-set of composition situations. CompoSecReasoner uses both validation types by enforcing rules during the composition procedure (pre-validation) and model-checking afterwards (post-validation). The overall validation procedure is implemented by a set of functions which are performed for each evaluated property. The procedure is further enhanced with the metrics characteristic.

For the core metrics formation, we adopt the well-studied techniques for attack surface calculation and medieval castle composition. The discussion of the underlying metrics is detailed in Section 2. The old surface metrics [49–53] consider only security as a system property and do not estimate the protection level. Moreover, the castle composition methodology evaluates static instances of a system. The SPD surface aggregates an analysis regarding security, privacy, and dependability while the SPD multi-metric reveals the provided protection level of the current setting. Finally, the composed SPD multi-metric tackles the measurement aspects of dynamic system composition and the added protection of subsequent layers of defense.

All these features contribute to a better understanding of the composition and SPD aspects. This enables the dynamic evaluation of the various system states and development of more effective MTDs. The implementation of CompoSecReasoner supports distributed reasoning as well as conflict resolution among the involved SPD agents [59].

The overall comparison results are summarized in Table 1.

Table 1. System composition and security validation features: (A) CompoSecReasoner [**this paper**], (B) IFML [21], (C) SMoLES [30], (D) Dracena [31], (E) MSC [32], (F) BLESS [34], (G) IoT Composer [35], (H) SMoLES-SEC [36], (I) SDT [37], (J) Attack Surface MTD [56], (K) Multi-metric-driven MTD [57]. The following notations are used: Y(es), N(o), P(artial)

Feature	A	B	C	D	E	F	G	H	I	J	K
System composition											
Dynamicity	**Y**	N	Y	Y	Y	Y	Y	Y	Y	N	N
Expressiveness generality	**Y**	Y	Y	Y	N	Y	Y	N	Y	N	N
Validation											
Pre-composition	**Y**	Y	Y	Y	N	Y	Y	Y	Y	N	N
Post-composition	**Y**	N	N	N	Y	Y	Y	N	N	N	N
Evaluated Properties											
Security	**Y**	N	N	N	N	N	N	Y	Y	Y	Y
Privacy	**Y**	N	N	N	N	N	N	N	N	N	N
Dependability	**Y**	P	P	P	Y	Y	Y	P	P	N	P
Artificial Intelligence											
Distributed reasoning/processing	**Y**	N	N	Y	Y	Y	Y	N	Y	N	Y
Conflict resolution	**Y**	N	N	N	N	N	N	N	N	N	N
MTD	**Y**	N	N	N	N	N	Y	N	N	Y	Y

3. Materials and Methods

This section details the measurement methodologies that are utilized by CompoSecReasoner. Initially, we introduce the core measurement method for estimating the SPD level of a single system component, which is based on our previous work, detailed in [60]. Here, we present the main concept. Then, we describe the extended version that is proposed in this paper for evaluating the composition of different components and systems (Section 3.3).

3.1. SPD Multi-Metric

3.1.1. Attack Surface

Consider a man taking refuge inside a mountain to avoid lightning. The physical barrier provides full protection. Every hole in the mountain or any other mean to cause harm to the man reduces his defense.

The attack surface metrics [52] consider that attackers would utilize the system's communication channels, methods, and data to infiltrate the system. The attack surface is designated by those elements at the entry and exit points which may be exploited by attacks. A threat is successful if it enables direct or indirect interaction with a protected asset. Microsoft developed the Relative Attack Surface Quotient (RASQ) [53] and formally quantified the relative attackability of the Windows server operating system platforms. The metric's outcome is further assessed against real vulnerabilities (based on related reports in the Computer Emergency Response Team (CERT) and Common Vulnerabilities and Exposures (CVE) databases).

These dated surface metrics take into consideration only the security perspective. While the conventional viewpoint that a hacker will choose a high privilege process for a potential attack seems correct in several occasions, he/she can also target a method with lower privileges which accesses sensitive personal data. The aforementioned metrics would falsely result that the second method is the least presumable to be exploited, with low influence in the attack surface. Same as with dependability, components with several dependencies or methods that are mission critical may be more valued for the hacker's perspective than a method with administrator access rights.

Based on these studies and the attack surface formal basis, we proposed the SPD surface to evaluate the properties of a system [60]. As the individual analysis of the three SPD properties produces erroneous conclusions regarding attackability, the SPD surface evaluates the aggregated effect, resulting

a more precise analysis and better design. Figure 1 illustrates the main SPD multi-metric concepts, which are detailed below.

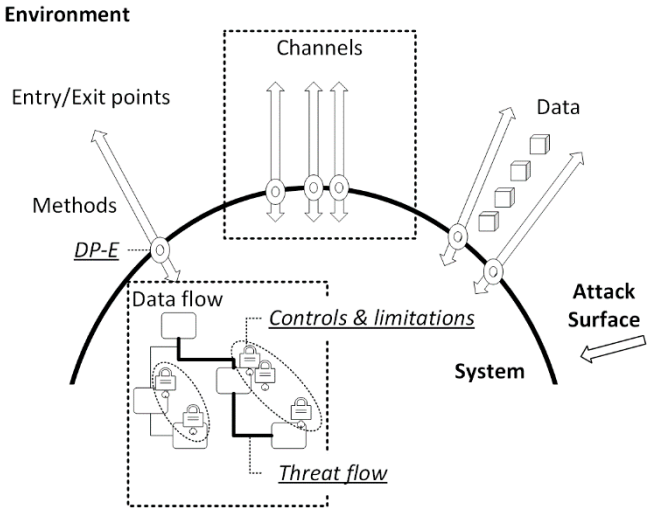

Figure 1. The main SPD multi-metric concepts of a system. The interaction points are located at the boundaries of the system's surface. Controls are placed along the dataflow to protect the system. The damage potential-effort ratio (*DP-E*) represents the opportunity to exploit a dataflow.

Not all threats contribute the same. Some resources are more likely to be attacked or an attacker may gain higher benefit from exploiting specific resources. An involving system resource introduces possible threat flows (TF). We analyze the system and determine the TFs where interaction between threats and assets is possible. For each of these flows, the damage potential-effort ratio (*DP-E*) is calculated, representing the flow's significance in the SPD surface. The higher the potential damage (or the lower the effort), the higher the flow's confluence in surface. As a result, the smaller the surface (lower assets' exposure) the higher the protection. The analysis is further enhanced with the contextual notions of standards and well-known evaluation methodologies for security, privacy, and dependability (Open Source Security Testing Methodology Manual (OSSTMM) [61] and Common Criteria Evaluation Methodology (CEM) [62] for security, ISO/IEC standards 27018 [63] 29100 [64], and the European General Data Protection Regulation (GDPR) for privacy, and IEC standard 60300 [65] for dependability).

The *DP-E* ratio of a TF integrates three primary ratios for security, privacy, and dependability, respectively. In security, the damage potential (DP) ratio reflects the method's privilege. The effort for the attacker is determined by the access rights of the examined method. In privacy-related interactions, DP is defined by the personal identifiable information (PII) type. The actuator's type, who processes the data, prescribes the effort. In dependability, DP is derived from the component's criticality. A component's dependency on other counterparts determines the effort. Table 2 details the *DP-E* parameters and the values that are applied in CompoSecReasoner. The values' assignment relies on the respective formal analysis of the primary attack surface metric [49,52]. The damage potential in case of a successful exploit on the legitimate system ranges from 1 (low damage) to 5 (high damage). The effort that the attacker has to devote to perform an attack ranges from 1 (low effort) to 4 (high effort).

Table 2. *DP-E* ratios.

SPD Property	Damage Potential	Effort
Security	Privilege [62] *-Root: 5* *-Debugger: 4* *-Authenticated user: 3*	Access rights [62] *-Administrator: 4* *-Authenticated user: 3* *-Anonymous user: 1* *-Unauthenticated user: 1*
Privacy	PII (personal identifiable information) type [64] *-Sensitive personal data: 5* *-Personal data: 4* *-Statistical data:1*	PII actuator [64] *-PII principal: 4* *-Contracted PII processor: 3* *-Third party: 2*
Dependability	Criticality [65] *-Mission critical: 4* *-Business critical: 3* *-Business operational: 2* *-Business supporting: 1*	Dependency level [65] *-Coupling: 4* *-Outgoing dependency: 3* *-Incoming dependency: 2* *-Independent operation: 1*

For each TF, the *DP-E* ratio is calculated by summing the related DP rates and dividing with *E*:

$$TF_{DP-E} = (S_{DP} + P_{DP} + D_{DP}) / (S_E + P_E + D_E) \quad (1)$$

where $TF_{DP\text{-}E}$ is the total *DP-E* ratio for a TF, $[*]_{DP}$ resembles the damage potential for each SPD property, while $[*]_E$ represents the related effort for each SPD property.

The surface for a system is the *DP-E* summation for all TFs:

$$Surface_{sys} = \sum TF_{DP-E} \quad (2)$$

where $Surface_{sys}$ is the system's surface and $\sum TF_{DP-E}$ represents the summation of all underlying *DP-Es*.

3.1.2. Protection Level

At this point, the surface metrics reveal the attackability of a system, but not the protection level. The SPD multi-metric methodology further extends the system's surface with the *porosity* feature. The TFs in the surface constitute the system pores where interaction among threats and assets is possible. Every TF is disassembled in security, privacy, and dependability pores, called *Access*, *Trust*, and *Complexity* pores respectively, representing its effect in the three SPD properties respectively. The system designer places controls (defence mechanisms) to protect the pore from attacks. Limitations form known conditions under which a control does not work properly. If all types of controls are placed for a pore and work correctly, full coverage of the interaction point is achieved.

Controls are the means that are applied by the system to restrict a threat's impact and increase the separation with the assets (e.g., protection mechanisms imposing authentication, confidentiality, or integrity). Two control types are described: (i) interactive controls which affect directly the operation when the interaction happens; and process controls which influence indirectly the operation by safeguarding the assets after the interaction's occurrence. Totally, *12 interactive* and *14 passive controls* are defined based on the aforementioned relevant security, privacy, and dependability standards. Table A1 in Appendix A summarizes these controls.

The applied controls of each pore are identified. At least one instance of each control type must be acquired in order to accomplish the perfect coverage for the pore. All of them contribute equally to a pore's coverage. The porosity value is defined as the TFs' summation, called *PorosityBase*. Perfect coverage for the two control types, defined as *PerfectCoverageInteractive* and *PerfectCoveragePassive* respectively, is denoted as:

$$PerfectCoverageInteractive = PorosityBase \times 12 \quad (3)$$

$$PerfectCoveragePassive = PorosityBase \times 14 \quad (4)$$

Limitations represent the inability for the defense mechanisms to work properly. They indicate the SPD state in regards to known restrictions and flaws. Six limitation types are modelled. Vulnerability, disclosure, and exposure limitations harm security, privacy, and dependability pores respectively. Weakness and concern limitations decrease the positive contribution of interactive and passive controls respectively. Anomaly limitations (faulty situation that cannot be controlled) state unidentifiable elements that are nor observed during the normal operation. A proper audit should take into account any anomalies as such situations cannot be controlled. They obstruct all three SPD factors and their effect is aggregated to each factor's evaluation.

For each pore, the deployed controls must be denoted. Then, the contribution of the relevant limitations on the deployed controls is measured based on the *attack potential* risk analysis that is described in CEM [15]. It is a function that assembles the hacker's preference, expertise, motive to presume upon specific limitations and attack particular assets, as well as, the resources that he/she is ready to dedicate in achieving his/her goal.

The function is further extended to cover not only successful wily attacks but the generic notion of a threat as well. Therefore, five factors are considered for the analysis of a potential threat:

- *Required time:* The period that it takes (e.g., days or weeks) to detect and exploit a limitation.
- *Expertise:* The technical knowledge and skills required (e.g., copy-cat, advanced, or expert).
- *Knowledge of the target:* Familiarity with the victim's system and operation (e.g., public, sensitive, or critical knowledge regarding some subsystems, etc.).
- *Window of opportunity:* Hackers may need appreciable access to the system to successfully presume upon a threat while avoiding detection.
- *Resources:* The required software, hardware, or other equipment (e.g., common or specialized resources).

Henceforth, the exploitation possibility of a threat is categorized as basic, enhanced basic, moderate, high, or beyond high. The method does not examine every likely scenario but derives a good indication concerning the defense status in accordance with standard ratings.

The distinct values of each parameter could denote high protection. Nevertheless, a limitation can ease the exploitation of other ones, ending up with a lower defense state after all. Thus, in the SPD measurement, the assigned rate of each of the three individual limitations (L_V for vulnerability, L_D for disclosure, and L_E for exposure, respectively) is calculated as the weighted summation of the related distinct ratings: V_V, V_D, and V_E, respectively. Then, these results are divided by the attackability of the specific pore type (*Access* pores for vulnerability limitations, *Trust* pores for disclosure limitations, and *Complexity* pores for exposure limitations, respectively):

$$L_V = V_V / \sum Access \times TF_{DP-E} \tag{5}$$

$$L_D = V_D / \sum Trust \times TF_{DP-E} \tag{6}$$

$$L_E = V_E / \sum Complexity \times TF_{DP-E} \tag{7}$$

The impact of the weaknesses and concerns is determined by measuring the absent interactive and process controls, divided by the *PerfectCoverageInteractive* and *PerfectCoverageProcess*, called L_W and L_C, respectively. Similarly with the relevant controls, the two limitations affect all three SPD parameters.

Anomalies cannot influence SPD by their own. The side-effects are considered in the presence of other limitations and estimated as their quantity divided by the sum of *PerfectCoverageInteractive* and *PerfectCoverageProcess*, called L_A.

3.1.3. Measurement

The surface and the protection level analysis are integrated. The overall method calculates the actual protection level by integrating the risk analysis for the attackable points with the efficacy of the deployed defense mechanisms.

The outcome is a triple vector of ⟨*Security, Privacy, Dependability*⟩ representing the total SPD of the system. The final SPD vector reflects the perfect separation (100) subtracting the weighted sum of the limitations' impact:

$$\begin{aligned} S &= 100 - [W_V \times L_V + (W_W \times L_W + W_C \times L_C + W_A \times L_A), \\ P &= 100 - [W_D \times L_D + (W_W \times L_W + W_C \times L_C + W_A \times L_A), \\ D &= 100 - [W_E \times L_E + (W_W \times L_W + W_C \times L_C + W_A \times L_A) \end{aligned} \tag{8}$$

where W_* represents the threat bias of each limitation type (as in Microsoft RASQ [53]).

The SPD evaluation is a function of separation among the possible threats and the defended assets. Each property's value falls in the range of 0–100, with 0 represents no defense and 100 full coverage and separation among the assets and threats. It has to be noted that the absolute SPD of ⟨100,100,100⟩ does not guarantee absolute protection. This designates that all required defense mechanisms have been deployed for all possible interaction points and set correctly, based on the current set of known limitations (e.g., CVE bulletins or CERT reports). Higher rating reveals redundancy of defense mechanisms, as well as, higher operational/implementation expenditures.

More information on the SPD multi-metric and the performed analysis can be obtained in the initiatory article [60].

3.2. Medieval Castle Approach

The medieval castle approach [20] is the main method to assess the security of composed systems. A system is considered as a medieval castle with security doors that constitute the target for assaults. Assaults are succeeded if they disrupt the castle's doors and reach the treasure room inside–the assets that the security mechanisms are safeguarding. The difficulty of breaking through the doors and reaching the treasure room reflects the security level for this castle/system. Each door represents a deployed security mechanism in the system and its resistibility in assaults is estimated by related metrics.

To quantify security, we have to measure the security of the system's sub-components and then calculate the overall protection based on composition rules. Figure 2 portraits the main settings of system composition under the medieval castle analysis. Circles designate protection mechanisms while the dashes represent the related doors (attack points). The assets are modelled as the black center of a circle.

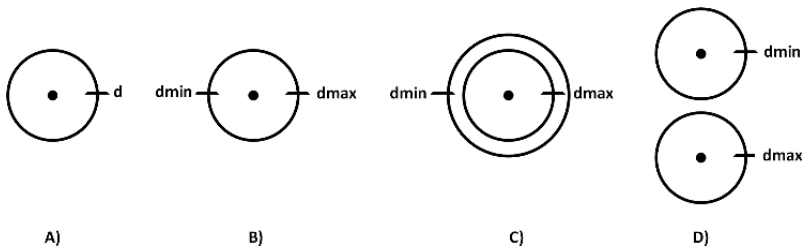

Figure 2. The core composition synthesis of the medieval castle approach. (**A**) is a simple system with one door. (**B**) features two doors at the same layer. (**C**) has two doors at sequential layers. (**D**) illustrates two castles where the attacker can target only one of them for a specific attack.

Assume that d constitutes a castle's overall protection level. Castle Figure 2A is the simplest composition set, with the assets been safeguarded by a single door. The castle's security is equal

with the defense level of this door. The other castles have two doors (d_{max} and d_{min}). Consider that d_{max} is stronger than d_{min}. Castle Figure 2B deploys two doors at the same layer, allowing concurrent assaults on both doors. The castle's security is equal to or weaker than the security of the weakest door (we presume that offending d_{max} could further weaken the protection at d_{min}). Castle Figure 2C deploys the two doors in a sequential manner. To reach its assets, attackers have to pass through both doors. The security here is stronger than Figure 2B and is at least as good as the defense of d_{max}. In the composition Figure 2D, there exist two castles that the wily attackers could mark. The assault is successful if it reaches at least one protected resource. However, it is assumed that the distance between the two castles is too long to enable concurrent assaults (e.g., we assume that an attacker will launch either a DoS attack on a web service or an on-line password guessing attack on the authentication service, but not both simultaneously). Thus, the interval of d_{min} and d_{max} determines the overall security in this setting.

Totally, three core composition operations are specified:

- AND-operation: $0 \leq d \leq d_{min}$
- OR-operation: $d_{max} \leq d \leq 1$
- MEAN-operation: $d_{min} \leq d \leq d_{max}$

The mediaeval castle analysis can be performed for more complex compositions as well, disassembling complex castles of any doors' topology. Based on the core composition operations, attack trees can represent the strain to infiltrate the system from the external protection layers. More information on the overall medieval castle methodology can be obtained in the primary article [45].

3.3. Composed SPD Multi-Metric

In this subsection, we describe the composed SPD multi-metric method which it is proposed in this paper. The SPD evaluation is applied as the basic metric to assess the defense level of the distinct system components. Then, the core security concept of the medieval castle analysis is extended with the SPD perspective. SPD multi-metrics are incorporated in the medieval castle approach, acting as a systematic methodology to estimate the protection of the castle's *"SPD doors"*. The result is the integral SPD of the currently composed system, which progresses the medieval castle composition analysis and covers the requirements of two premier real-time SPD modelling features, as they are detailed below.

However, only static system instances can be examined under the medieval castle approach. On the other hand, the proposed method assesses dynamic systems with de/composition events changing the overall composition and SPD aspects at runtime. For instance, two sub-systems could be secure in some aspect, but their composition might not be necessarily safe in the same aspect. The composed SPD multi-metric enforces pre- and post-composition validation. Initially, it derives if the composition is feasible. Afterwards, it concludes the composition's result and its side-effects. Decomposition events are handled in a similar manner.

In an actual system, the deployed defenses at the various layers could interact with each other. This concept is not covered sufficiently by the castle analysis. In the proposed method, the missing controls of the SPD multi-metric are covered by the related controls which have been deployed in the adjacent outer layers, capturing the increasing defense of the interacting subsequent protection layers. Such as, consider an insecure data exchange service which can be defended by a secure communication service performed at the network layer, offering integrity, authentication, and confidentiality. The total SPD reflects the overall control coverage which is accomplished by the current composition.

The method is mounted in the ambient intelligence filed of a smart home. The idea is that every business that provides a smart home component (e.g., embedded equipment or software applications) will calculate the main SPD multi-metric features in advance. Then, the user buys on-the-self products. The in-house composed SPD multi-metric evaluates the final SPD, once these products are getting installed.

In a common installation, the deployed smart devices observe ubiquitous factors and interchange data. The user manages to the overall setting with computer or smart phone. We implement a smart home system with smart devices for assisting living conditions. The house includes a laptop and several smart devices (i.e., surveillance cameras, TV, air-condition, printer, fridge, etc.). A wireless router creates a LAN, interconnecting all equipment, and permits external connection with Internet. This smart home composition is depicted in Figure 3.

Initially, the SPD multi-metric of each distinct element (i.e., laptop, smart devices, and router) is measured. Afterwards, these ingredients can be integrated based on the medieval castle composition operations (the composed SPD assessment is detailed in Sections 3 and 4). These components are composed to a LAN with MEAN-operations, the LAN is composed with the router with an OR-operation, while the router is composed to Internet with an OR-operation. Figure 3A illustrates the overall composition. Attackers could penetrate the home router via Internet and then attack the rest components in the LAN. In Figure 3B, the legitimate user connects the LAN with a smart phone (MEAN-operation). The SPD is measured as in Figure 3A. However, in Figure 3C, the user connects the LAN similarly as Figure 3B (MEAN-operation), but here the phone also deploys its own mobile Internet connection (AND-operation with Internet). Hackers are given now an alternative port to infiltrate the system via the mobile Internet connection.

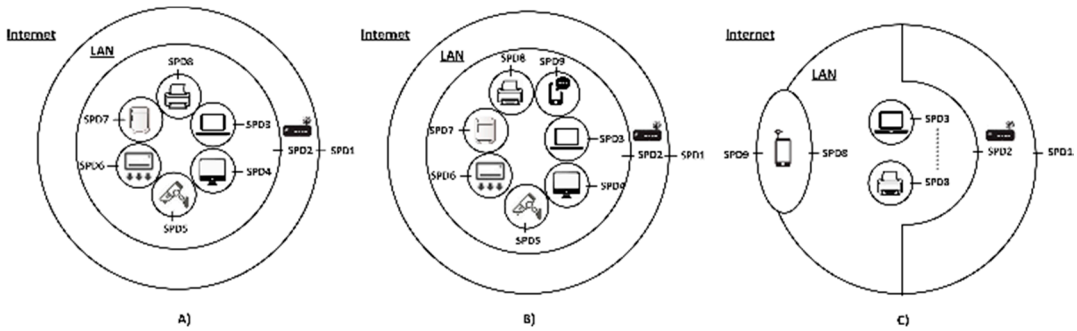

Figure 3. The core composition operations of the medieval castle method. (**A**) represents the current setting of the smart home with 6 devices connecting to a LAN with Internet connection through a router. (**B**) figures the composition of a 7th device to the LAN. (**C**) depicts the direct connect of this device to the Internet.

A user-driven scanning tool is implemented in Java to assist the SPD evaluation. Most evaluation steps that are mentioned above are automated, leaving a specific set of actions to the user. For the SPD multi-metric, this tool identifies the resources and the high-level entry and exit ports that constitute the surface. The user specifies which of them confluence the porosity and designates the appropriate controls. The tool performs the potential-threat analysis for the limitations and the estimation of the relevant values. Finally, the user matcher the limitations to controls and pores, with the software estimating the SPD level. System designers could easily test different configurations and deduce the best SPD for an examined setting. For the composed SPD multi-metric, the user must also define the composition operation types.

4. CompoSecReasoner

CompoSecReasoner acts a practical framework that captures the dynamic nature of a system with new sub-components (for which some SPD aspects are known) are incorporated into the main pre-existing one. In our work, we examine the secure composition procedure for systems-of-systems. As a case study, we study heterogeneous embedded ecosystems in the IoT domain which are composed and materialize a new system. We apply a systematic method to evince that the deployed technologies are actually composable under a metric-driven composition strategy and the composed result accomplishes the desired SPD features.

CompoSecReasoner can be utilized for secure system design and offers systematic verification of the underlying systems. The metric-driven strategy can be used for the comparison of different ongoing designs and extract the best ones based on their SPD properties.

4.1. Attributes

The attributes are services/technologies which are offered by the system's components, such as a specific cryptographic protocol, and are defined when a component is instantiated. Their defense status is evaluated by the SPD multi-metric method and they are mostly instantiated at the device layer. Each different device type provides a specific collection of attributes, based on its computational capabilities, i.e., of nano, personal/micro, and power nodes.

4.2. Sources

Sources constitute critical assets and data which are processed by components and constitute the main theme of the SPD assessment (modelling the treasure rooms of the medieval castle method and the subject of the malicious actions). Sources can be created/deleted, processed, or exchanged by system components. A component owns a specific set of them which can manipulate accordingly by performing operations that can alter their SPD properties. The SPD value of sources is defined by the SPD value of the latest operation that processed them (see the following subsection). When components exchange a source, like transmitting encrypted data via Internet, they really convey a new instance of that source. The sender preserves the original one while the receiver a copy of it, that is now deemed a different source (with indirect association).

4.3. Operations

4.3.1. Process Operation

Operations form a primary characteristic of the proposed method. They constitute series of actions which are executed by the components when creating and processing sources. An operation includes of a set of attributes and their execution sequence. The involved attributes are performed either in parallel or subsequently. At any given time-point, the minimum SPD of the concurrently performed attributes is disclosed, as it constitutes the weakest security link. The SPD for an operation is defined by these attributes. The absent controls of the SPD multi-metric are wrapped by affiliated controls in the adjacent previous time-point, modelling the added defense of the subsequent execution. The overall SPD reflects the operation's pore coverage which is provided at the final time-point. The exact operation's SPD calculation is also detailed in Algorithm 1. As a result, performing an operation could de/increase the SPD state of a source.

Algorithm 1: OperationEvaluation

Input: The operation identifier op.
Output: The operation's <SPD>.
for *each timepoint* t_i *in op* **do**
    SequentialAttrs.Add(AttrWithMinSPDAtTi(op));
end
for *each attribute* $attr_i$ *in SequentialAttrs* **do**
    $attr_i$*.CoverMissingControlsFromAttr(*$attri_{i-1}$*)*;
end
operationSPD = SequentialAttrs.getFinalAttr().EvaluateSPD();
Return(*operationSPD*);

Let's take as an example an attribute that defines a plain network transmission-service. The security property of the SPD vector is low, as the affiliated controls are absent (i.e., confidentiality, integrity,

and authenticity). Thereupon, an attribute describing an encryption-service is defined. This service performs authenticated-encryption on the processed information (sources), offering the confidentiality, integrity, and authenticity features. Thus, the security is enhanced by performing the two services in a subsequent execution—information is encrypted by the encryption-service and then sent by the network transmission-service. Security controls are cascaded from the first service and cover the absent mechanisms of the second one. The total SPD describes the protection of the network transmission-service enhanced in security by the controls of the encryption-service.

4.3.2. Composition Operation

Under our methodology, an operation can also designate the composition steps which have to be performed to incorporate components. Components of the same system layer are composed to form a component of the adjacent higher layer, e.g., local nodes which are interconnected into a network. A set of operations determine the operation that these nodes have to execute to connect the network, i.e., the communication protocol steps.

Component execute composition operations as long as they can satisfy the functional and non-functional pre-conditions. Functional pre-conditions are determined by the attribute's set that is required for this operation. Components have to be capable to perform all the attributes of the set. The applicability check is also detailed in Algorithm 2.

Algorithm 2: OperationApplicabilityCheck

Input: The component *C*, the set of operations set_{op}.
Output: TRUE/FALSE: checks if the operation can be performed or not.
for *each operation* op_i *in op* **do**
    opiAttrs = GetOperationAttrs(op_i*)*;
    if *(!ContainsAllAttrs(C, opiAttrs))* **then**
        Return(*FALSE*);
    end
end
Return(*TRUE*);

Non-functional pre-conditions are quality features which have to be also fulfilled. They are represented as minimum levels of the SPD multi-metric factors, such as the minimum coverage value of a confidentiality control. To execute a composition operation, components have to grant these SPD constraints as well.

The proof that a component can execute an operation is implemented in Event Calculus (EC) [21]. We define our context features and the validity checks of the functional/non-functional pre-conditions as fluents, events, and rules in EC. Thereafter, we can prove that if the pre-conditions are valid, then the functionality and the derived SPD values are valid.

For composition operations, apart from the relevant attributes, the composition definition denotes also the composition type, based on the corresponding composition operations of the medieval castle approach (i.e., AND, OR, MEAN). This concept is used by the process that estimated the SPD level of the composed component (refer to the following subsection).

4.4. System Components

Based on our method, a system is consisted of SPD-sensitive components. Component types include nodes, networks, middleware, and security agents (the overlay), and they can:

- possess sources–processed data;
- deploy attributes–technologies and protocols;
- execute operations–series of attributes;
- contain sub-components–components of lower layers;

- offer specific SPD levels–based on the assessed metrics and their underlying sub-components.

A component's SPD value is strongly affected by the SPDs of its sub-components'–as the weaker inner defense links; and the component of higher layer with the minimum SPD which contains the examined component–as the weaker outer link. For single components of the lowest layer, we their sources act as the equivalent element for the sub-components.

The assessment of a component's SPD is formally described in Algorithm 3. The structural SPD for a component is estimated based on the medieval castle methodology on its sub-components. The sub-components are modelled as the castle's doors and their SPD reflects the difficulty of penetrating via those doors. The architectural significance for each sub-component/door is defined by the relevant composition operation type.

Algorithm 3: *ComponentSPD*

Input: The component *C*.
Output: The component's <SPD>.
structuralSPD = MedievalCastleSPD(C);
constraintSPD = MinRelativeComponentSPD(C);
componentSPD = structuralSPD;
if (*structuralSPD > constraintSPD*) **then**
    componentSPD = constraintSPD;
end
Return(*componentSPD*);

The SPD for a component cannot overdraw the minimum SPD of the associated component at the higher layer. For instance, the SPD level for a node is determined by the networks that it is currently connected in (i.e., a device's security could be reduced when it connects to Internet, as more attack vectors are now open/possible due to the device being networked).

4.5. Composition and Decomposition

As an example, let us examine a network composition consisting of two nodes (i.e., LAN of two smart home devices) exchanging data with a secure communication protocol (e.g., IPsec via WiFi). Initially, the SPDs for individual node components are calculated. The communication protocol is defined as a series of subsequent operations. This series reflects the functional requirements for this composition. The non-functional pre-conditions are determined by relevant metrics along with their SPD parameters, which must be supported by all associated components. A related network component is modelled and all nodes have to fulfil those pre-conditions to connect (e.g., Figure 3A). A composition event occurs by each node, designating the connection attempt. If a node enters successfully the network, the network SPD will be re-calculated (e.g., Figure 3B). Them we also reason if other properties hold under this composition (i.e., the network component–two nodes under the secure communication protocol). For instance, when the placed controls of network confidentiality are deployed, it is derived that the network is now offering also the confidentiality property. The evaluation for a composition event on a component *C* to compose a sub-component *subC* is further detailed in Algorithm 4.

Algorithm 4: Composition

Input: The component *C*, a subcomponent *subC*.
Output: *TRUE/FALSE*: checks if the composition is performed or not.
currentSPD = C.SPD;
if *(!C.ComplyWithFunctionalPre(subC) or !C.ComplyWithNonFunctionalPre(subC))* **then**
    Return(*FALSE*); //Composition is blocked
end
C.compose(subC); //Composition is performed
if (*ComponentSPD(C)* != *currentSPD*) **then**
    Return(*FALSE*); //Composition is blocked
    for *each subcomponent of C* **do**
        ComponentSPD(subcomponent);
    end
end
ComponentSPD(subC);
Return(*TRUE*);

Components which are affected by a composition will re-calculate their SPD and composition associations. Composition events might cascade sequential changes to the status of components at different system layers and even affect the whole setting (e.g., Figure 3C). The system balances after a few time-points (depending on the amount of components and layers) and the total SPD is evaluated. Decomposition events are encountered in the same manner. If a sub-component is decomposed from a component, their SPD aspects are re-calculated as aforementioned.

5. Compositions of SPD Metrics

The properties which are offered by a system can be attested with formal methods. In general, it is commonly accepted that no computer system is theoretically secure. Thus, validation methodologies confirm if a system meets some security properties against predefined threats, based on our specifications and best practices. For instance, when a system deploys the defenses against Denial of Service (DoS) attacks which are mentioned in the NIST's standards, we could deduce that the system is DoS resilient for the currently known DoS threats.

In this section, we frame the systematic representation of the system's security aspects and composition. The methodology is based on EC and the theoretical research in [58].

5.1. Composition Verification Definitions

Definition 1 (Situations)**.** *A situation S is defined as a system snapshot in a specific time-point.*

Definition 2 (Events)**.** *A set of events E includes all events that can alter the current system situation. If we consider that S is the current situation and* $e \in E$*, then after* $Happens(e, S, t) \rightarrow HoldsAt(S\prime, t+1)$*, where S' is the new state.*

Definition 3 (Component Composition)**.** *Assume Comp as the set of distinct components. For each composed component* $qcomp \in Comp$*:*

(A) Assume QCompOp as the set of composition operations that all qcomp's sub-components have to be capable to execute.
(B) Assume QCompCons as a set of constraints along with the minimum SPD values that all qcomp's sub-components have to satisfy.
(C) A component $qcomp_2 \in QComp$ *at the adjacent lower layers of qcomp, could be incorporated to qcomp if it can perform an operation* $qcompop \in QCompOp$ *and grantify all the constraints in QCompCons.*

The composition event is modelled as Happens(Compose(qcomp,qcomp$_2$,qcompop), S,t))→(HoldsAt(S′,t + 1) ∧ Happens(EvaluateComposedSPD(qcomp),S′,t + 2)). After a successful composition at S′, it holds that $qcomp_2 \in (QCompSub\ of\ qcomp)$. *A component* $qcomp_2$ *could be disintegrate from qcomp at any time-point. The decomposition event is modelled as* $Happens(Decompose(qcomp, qcomp_2, qcompop),\ S, t)) \rightarrow HoldsAt(S\prime, t+1)$. *After a successful decomposition at S′, it holds that* $qcomp_2 \notin (QCompSub\ of\ qcomp)$.

Definition 4 (System Composition). *Assume Sys as an evaluated system. Sys is a set that includes the composed components of the system along with their composition associations, properties, and metrics. Composition of the component comp* $\in QComp$ *to Sys by executing the operation sysop* $\in QCompOp$ *of Sys is modelled as* $Happens(Compose(Sys, qcomp, sysop\ ), S, t)) \rightarrow HoldsAt(S\prime, t+1)$, *where it holds* $Sys' = Sys \cap qcomp$. *Composition of two distinct systems* Sys_1 *and* Sys_2 *is modelled as* $Happens(Compose(Sys_1, Sys_2, sys_1op), S, t)) \rightarrow HoldsAt(S\prime, t+1)$, *where it holds* $Sys' = Sys_1 \cap Sys_2$.

5.2. SPD Validation Definitions

SPD validation is modelled by the features of metrics and properties. Metrics describe measurable aspects of a system. Section 2 presents the main characteristics and the assessment procedure for metrics and the SPD formation. The SPD multi-metric measurement is presented in [60]. The Medieval Castle compositions are detailed in [45]. The composed SPD multi-metric calculation of an operation is based on these two approaches and their systematic validation. The operation assessment procedure is detailed in the Section 4.3. The component's assessment procedure is based on similar assumptions and is detailed in the Section 4.4.

The Section 4.4 defines the association among components and metrics. Definition 3 and Definition 4 determine the prior-composition validation procedure for enforcing the pre-conditions which have to be fulfilled to materialize the composition. The definitions Definition 6 and Definition 7 describe the post-composition validation procedure.

Definition 5 (Constraint Validation). *For a constraint cons which is enforced by the component qcomp* $\in QComp$ *(cons* $\in QCompCons$:

(A) Assume cons_comp_pre as the pre-conditions of this constraint at qcomp's layer.

(B) For a sub-layer i, assume cons_subcomp_pre_i as pre-conditions of this the constraint at sub-layer i.

(C) the constraint cons holds iff qcomp fullfils cons_comp_pre and each sub-component $qcomp_i \in (QCompSub\ of\ qcomp)$ *of sub-layer i fulfils the related cons_subcomp_pre_i.*

(D) if a sub-component$qcomp_i \in (QCompSub\ of\ qcomp)$ *of sub-layer i violates the related cons_subcomp_pre_i, then disintegrate the sub-component:* $HoldAt\Big(ViolateSubCompCons\big(qcomp_i, cons\big), S, t\Big)) \rightarrow$
$Happens\Big(Decompose\big(qcomp, qcomp_i, decompop\big), S, t\Big) \wedge$
$Happens(EvaluateComposedSPD(qcomp, cons), S, t+1)) \wedge$
$Happens\Big(EvaluateComposedSPD\big(qcomp_i, cons\big), S, t+1\Big))$.

Definition 6 (Metric SPD). *For a metric* $m \in Q$ *of the component qcomp* $\in QComp$ *(m* $\in QCompM$*):*

(A) Assume mParam as the parameters of the system which influence m.

(B) If the state of mParam changes, m's SPD level is re-calculated: $HoldAt(CompMetric(qcomp, m), S, t)) \wedge Happens(ChangeParamStatus(mParam), S, t) \rightarrow Happens(EvaluateMetricSPD(qcomp,\ m), S, t))$.

Definition 7 (Component SPD). *For a sub-component qcompsub* $\in QComp$ *of the component qcomp* $\in QComp$ *(qcompsub* $\in QCompSub$*):*

(A) *If the state of qcompsub changes, qcomp's SPD level is re-calculated:* $HoldAt(ComponentSub(qcomp, qcompsub), S, t)) \wedge Happens(ChangeCompStatus(qcomp), S, t) \rightarrow Happens(EvaluateComposedSPD(qcompsub), S, t))$.

(B) *If the state of qcomp changes, qcompsub's SPD level is re-calculated:* $HoldAt(ComponentSub(qcomp, qcompsub), S, t)) \wedge Happens(ChangeCompStatus(qcomp), S, t) \rightarrow Happens(EvaluateComposedSPD(qcompsub), S, t))$.

6. Implementation

6.1. System Layers

The EU research project nSHIELD [66,67] investigated new multi-layer architectures for heterogeneous embedded systems. It proposed four generic layers for embedded system designs. From bottom-up, the node layer is comprised of all the distinct components/devices. The network layer includes clusters of nodes which are composed into networks. The middleware layer forms the software layer for the networks' administration. Finally, at the top the overlay layer consists of security agents, which interchange knowledge and manage the individual sub-systems.

6.2. Motivating Example

We develop the CompoSecReasoner logic in Event Calculus (EC) [68], and deploy it for the evaluation of an actual IoT ecosystem. The application implements an ambient intelligence setting for the administration of a smart home system.

In the current smart home setting, most of the utilized devices reply to any access request and do not offer any secure access control techniques. Thus, cyber-attacks are becoming feasible as these devices are connected to Internet either directly or indirectly through the user's smart phone [69]. To improve security, we deploy the secure Policy-Based Access Control (PBAC) mechanism which is proposed in [70]. At the device/node layer, this framework enforces access control based on pre-defined security configurations. The policies are securely maintained at the framework's back-end framework, which is installed on a laptop.

Furthermore, we use the lightweight cryptographic library (ULCL) [71] to develop a service for secure storage. This service encrypts information at the device-end to safeguard data from disclosure in case where the equipment gets compromised. The standardized cipher AES is utilized for the main cryptographic operations.

The embedded system is comprised by a BeagleBone and a BeagleBoard devices [72] (nodes with constraint computational/communicational features), and a laptop (power node administrates the whole setting). The BeagleBone/BeagleBoard nodes capture environmental factors (like humidity and temperature) and manage electrical devices, such as fridge, air-conditioning, room-lights, etc. The BeagleBones/BeagleBoard audit the associated electrical equipment based on the enforced access strategies of PBAC. The laptop runs the administration framework of smart home management, gathers data from the underlying resources, and displays them to the user. The offered functionality of each device is developed as a service. The nodes interact wirelessly with WiFi and IPsec. All data are encrypted and several security levels are provided, ranging from low-energy consumption settings for "green" operation to high security ones.

CompoSecReasoner runs on the laptop with the PBAC mechanism. CompoSecReasoner is implemented as the reasoning behavior of the home's SPD agent, which can also interact with other smart building agents laying in the overlay layer. The management framework forms the middleware layer which administrates the local ambient ecosystem. The devices are composed into networks. The BeagleBone/BeagleBoard and the laptop are the nodes. As an application scenario, we define a smart-home for assisting living conditions, while for the attacker's model we consider individual hackers with moderate cyber-security knowledge and low attack/computational capabilities (i.e., 1 to 5 PCs).

6.3. Technical Details

6.3.1. Reasoning System and Agent Technologies

EC [68] is implemented and extended by the Discrete Event Calculus Knowledge Theory (DECKT) [73], which provides reasoning for automated epistemic, casual, and temporal settings with dynamic, partially-known, and/or uncertain domains. DECKT is developed in Java and JESS. JESS is a rule engine for knowledge-based reasoning in the form of declarative rules, written in Java.

DECKT was further extended by [74] where preferences, rule priorities, and real-time events' evaluation were introduced. The overall solution was deployed as the reasoning behavior of agents implemented in the Java Agent DEvelopment framework (JADE). JADE constitutes one of the most popular agent platforms and it is open source. It simplifies the development of multi-agent systems and supports the associated specifications, made by the Foundation for Intelligent Physical Agents (FIPA) [75]—an IEEE Computer Society standards body. Moreover, JADE materializes the Agent Communication Language (ACL) [76]—the standard language for agent interaction specified by FIPA. Finally, a real-time multi-agent epistemic reasoner is implemented that also provides a conflict resolution mechanism, which resolves inconsistences among the agents' local knowledge and viewpoints, resulting in a coherent global view of the overall system.

We utilized this reasoner from [74] to develop the composition framework, which is presented in this study. Thus, the main reasoning behavior is extended to model our proposal, and henceforth, all "SPD agents" deploy the CompoSecReasoner behavior. Thereafter, we further proceed with the implementation of the technical aspects and offer the overall solution as a service.

6.3.2. Middleware and Service Platform

OSGi constitutes a standardized service platform and module system in Java, developing a holistic and dynamic component model. The components' services are materialized as bundles for deployment and are remotely installed/uninstalled and started/stopped without the need for reboot. The leading open source distribution, called Knopflerfish [77], is adopted by the CompoSecReasoner framework.

JADE, apart from its pure distribution as a multi-agent platform, is also offered in the form of a JADE-OSGi bundle. Thus, the above mentioned CompoSecReasoner agent is deployed as a JADE-OSGi bundle, facilitating the real-time system administration functionality.

The components port their services as Knopflerfish-OSGi bundles and interact with the associated JADE-OSGi bundle that runs the CompoSecReasoner agent. The software modules of the proposed framework are portrayed in Figure 4.

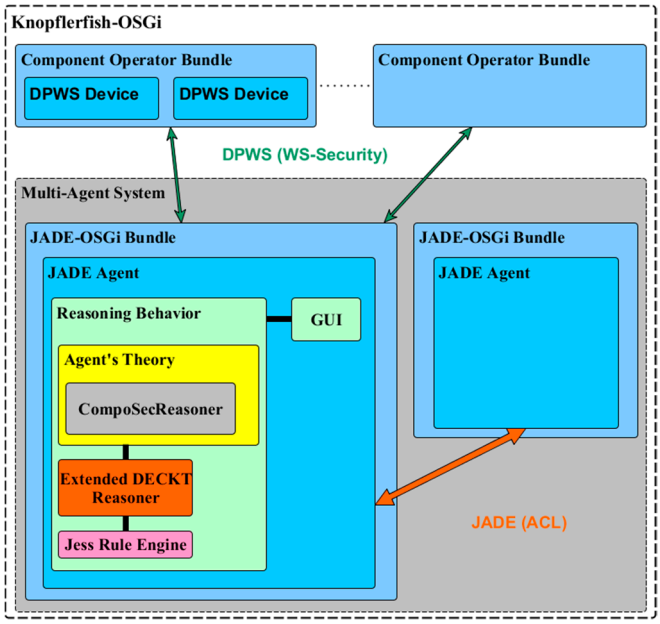

Figure 4. Software components of the CompoSecReasoner framework. CompoSecReasoner is formed as the reasoning process for the agents. All agents communicate upon the JADE-OSGi platform via ACL, forming a multi-agent system. Each agent controls the underlying embedded devices through DPWS.

The overall setting adopts a Service oriented Architecture (SoA). Each entity specifies semantic information such as the supported services and their type, in the standardized Device Profile for Web Services (DPWS), which is specified by the Organization for the Advancement of Structured Information Standards (OASIS). This enables device/service description, discovery, eventing as well as messaging functionality.

6.3.3. Platform Security Features

The build-in security mechanisms of the OSGi platform support inner-platform protection, both for devices and agents, by restricting the bundle operation to predefined capabilities, such as which bundle could be started or stopped, by whom, and when. The JADE-S add-on safeguards the ACL messages by offering the Java Cryptography Extension (JCE), Java Authentication and Authorization Service (JAAS), and Java Secure Socket Extension (JSSE) extensions. JADE-S enables among others message encryption and signature, agent actions authorization against agent permissions, and user authentication [78].

6.4. Demonstration

We implement the smart home system that is described in the Sections 3.3 and 6.2. We mainly model the aforementioned PBAC framework with IPsec [70], which is comprised of the following modules:

- *Policy Enforcement Point* (PEP): lays in the node layer imposing the access control at the device-end
- *Policy administration Point* (PAP): runs in middleware layer maintaining the access policies
- *Policy Decision Point* (PDP): also runs in the middleware, assessing the applicable policies which make the authorization decisions
- *Policy Information Point* (PIP): deployed in the middleware as the framework's repository for the offered features

PEP interplays with the PDP (between nodes and the middleware), supporting four configurations for different SPDs: plaintext-only, authentication-only, encryption and authentication, as well as, WS-Security. PEP interacts with user of a device, offering three configurations: plaintext-only,

authentication-only, encryption-and-authentication. Further information can be found in the primary article [70].

In the node layer, the two BeagleBone/BeagleBoard devices and a laptop act as the nodes n_1, n_2, and n_3, respectively. The nodes n_1 and n_2 are similar and deploy the attributes IPsec, ULCL, PEP with the three SPD configurations, and WiFi. As sources, they store the local information by PEP. The node n_3 offers the attributes IPsec, ULCL, PBAC with the four SPD configurations of the PDP module, CompoSecReasoner, and WiFi. As sources, it maintains the information from PDP, PIP, and PAP accordingly.

The network *net* is defined in the network layer. This network supports the attributes IPsec and WiFi. To connect the network *net*, a node has to execute a composition operation (MEAN-operation) that demands the subsequent performance of PEP, IPsec, and WiFi–op_5. Nodes n_1 and n_2 execute the operation and connect *net*.

The middleware *midl* is defined in the relevant layer. It deploys the attribute PBAC and the aforementioned protections of OSGi. To connect the middleware, two composition operations are supported to incorporate node and network components, respectively. A node has to execute the attributes PBAC and IPsec–op_2; while a network has to execute only IPsec–op_6. The middleware lays on the laptop node. Therefore, a composition event is executed by n_3 to *midl*. The network *net* executes the second operation and is integrated with *midl*.

In the demonstrated scenario, we model a single SPD agent, called *SA*. It offers the attributes CompoSecReasoner and the aforementioned protections of JADE-S. Two composition operations are supported to integrate components from the node and middleware layers, respectively. Nodes have to execute the attribute CompoSecReasoner–op_1; while the middleware component has to execute the attribute PBAC–op_3. *SA* is installed on the laptop node, as well as, the PBAC middleware. Hench, n_3 executes the relevant composition event to *SA*. Thereafter, *midl* executes the second composition operation to be incorporated as well. Finally, the overlay *over* is defined in the top layer and includes two alike *SAs*. To be integrated in the overlay, the agents have to deploy the CompoSecReasoner–op_4.

6.5. Levels of SPD

The SPD multi-metric evaluates the SPD value for all the demonstrated attributes. ULCL offers a single SPD level for the secure storage of local data of $ULCL = \langle 80, 50, 30 \rangle$, which affects the confidentiality aspects of a node. WiFi enables the wireless communication of devices (nodes) and also provides a single SPD level of $WiFi = \langle 20, 20, 50 \rangle$, which affects data integrity property of a network component. IPsec offers authentication and encryption and supports three SPD levels for relevant cryptographic key sizes ($IPsec_{128} = \langle 68, 20, 70 \rangle$, $IPsec_{191} = \langle 75, 20, 70 \rangle$, $IPsec_{256} = \langle 85, 20, 70 \rangle$). if all node components in a network intercommunicate via IPsec, the SPD factor for data confidentiality is enhanced. The node attribute PEP supports three levels of SPD based on the underlying security configurations ($PEP_{Plain} = \langle 10, 30, 10 \rangle$, $PEP_{Auth} = \langle 50, 50, 50 \rangle$, $PEP_{Enc\_Auth} = \langle 80, 70, 80 \rangle$). In the same manner, PBAC incorporates four SPD states between the communication of the PEP and PDP modules ($PBAC_{Plain} = \langle 10, 30, 10 \rangle$, $PBAC_{Auth} = \langle 50, 50, 50 \rangle$, $PBAC_{Enc\_Auth} = \langle 80, 70, 80 \rangle$, $PBAC_{WS\_Sec} = \langle 90, 80, 90 \rangle$). CompoSecReasoner's SPD is estimated at $CompoSecReasoner = \langle 70, 50, 70 \rangle$. All metrics are assessed dynamically at runtime. The security parameter is supported by several protection features, such as IPsec and JADE-S. PBAC safeguards the overall privacy. Dependability is enhanced by all the deployed mechanisms.

6.6. Composition and SPD Assessment

Initially, we instantiate the system's components (n_1, n_2, n_3, *router*, *net*, *over*) for moderate SPD with low power consumption. PBAC, PEP, and IPsec are set at the SPD's of $PBAC_{Auth}$, PEP_{Auth}, and $IPsec_{128}$, accordingly. The SPD of each distinct component is assessed by the CompoSecReasoner–no composition operations are performed at this time-point and no component includes sub-components.

Node n_3 (laptop) executes three operations with ULCL to encrypt the underlying data for PIP, PDP, and PAP, accordingly: $Happens(PerformOp(n_3, ULCL, Source_i),\ SourceSPD(Source_i,\ S, P, D), t)) \rightarrow HoldsAt(SourceSPD(Source_i,\ S\prime, P\prime, D\prime), t+1)$.

The SPD value for each of these sources is the SPD level of *ULCL*–$\langle 80, 50, 30\rangle$. The changes trigger the SPD assessment for the node n_3 ($\langle 80, 50, 30\rangle$—Definition 7).

In the same manner, the nodes n_1 and n_2 encrypt their local versions of PEP. The nodes' and sources' SPDs are calculated at $\langle 80, 50, 30\rangle$.

Nodes have to execute the network and middleware attributes to enter the LAN. Operation op_1 enforces the relevant composition requirements, designating the subsequent execution of WiFi, IPsec, and PEP. The type of such compositions is the MEAN-operation (as discussed in Section 3.3).

Local networking is supported by the router. The LAN is a sub-component of the router. The composition requirements for this network are modelled by operation op_2, denoting the subsequent performances WiFi and IPsec under an OR-operation composition.

Apart from the internal networking, the router offers external connection with Internet, modelled under op_3. An OR-operation defines the composition between the router and the Internet.

Thereafter, n_3 is integrated to *net* (op_1): $Happens(Component(net, n_3, op_1), ComponentSub(net), t)) \rightarrow (HoldsAt(ComponentSub(net), n_3, t+1) \wedge (EvaluateCompSPD(net), CompSPD(net, S, P, D), t+2))$. The SPD for *net* is revisited due to this composition ($\langle 70, 50, 70\rangle$—Definition 5), and then, the n_3's SPD of is also re-assessed ($\langle 70, 50, 30\rangle$—Definition 7). Nodes n_1 and n_2 are also integrated to *net* (op_1). The SPD level reaches $\langle 68, 30, 70\rangle$ for *net*.

The network *net* is incorporated to the *router* (op_2). The *router*'s SPD restricts the SPDs of its sub-components (*net* and subsequently n_1, n_2, and n_3). Thereupon, *router* connects to Internet (op_2). After the system's composition, we can reason about the metrics that are eventually achieved. The total SPD of this system is estimated at $\langle 70, 50, 30\rangle$ and is strongly influenced by n_3's SPD.

6.7. Reactive Strategies & MTDs

We can utilize the metric-driven runtime administration features of the tool and dynamically adjust the 36 configurations of the system (combinations of 3 IPsec, 4 PDP, and 3 PEP configurations) based on our high-level SPD goals and the AI reactive plans that can be triggered the monitored events in real-time. The relevant SPD levels are illustrated in Figure 5 below (the individual S, P, and D, values indicate the percentage (%) of pore coverage, ranging from 0–100% for no to optimal protection, respectively). The operational status ranges from configurations for low energy consumption to settings for high SPD.

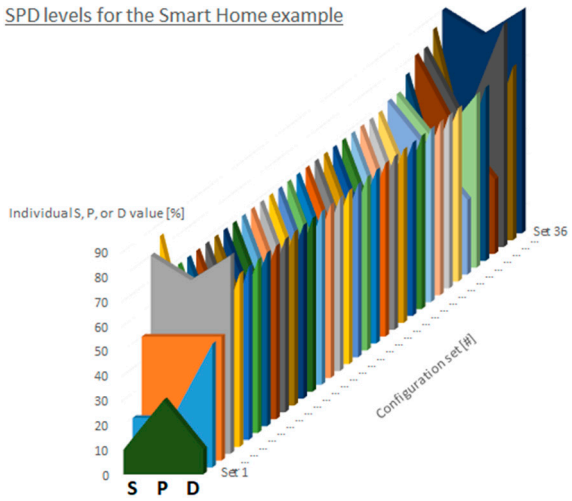

Figure 5. The SPD level of the 36 possible system configurations.

The smart home can connect with other smart entities via Internet (op_3), as it is depicted in Figure 6. Assume the scenario where the SPD agent of a neighboring smart home monitors a cyber-attack in its region. The agent alerts the overlay to warn the rest agents. The SPD agent of the house takes the decisions to adjust the system's configuration from low power consumption with moderate security, to high security and protection. The reactive strategy enforces the underlying components to deploy $PBAC_{WS\_Sec}$, PEP_{Enc_Auth}, and $IPsec_{256}$ instead. The plan triggers various operations to set the new system state. The overall protection is enhanced and the new SPD level for the house is increased to $\langle 90, 80, 90 \rangle$.

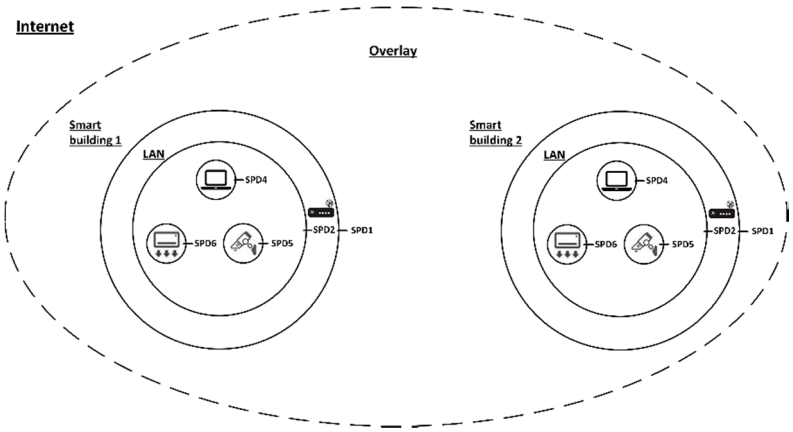

Figure 6. Composition of two smart buildings.

6.8. Performance Assessment

CompoSecReasoner is evaluated under a testbed. The user administrates the system's SPD via an Android application in a smart phone (Android 5.0 KNOX OS, 16GB RAM, 2.5GHz quad-core CPU, 4G, Wi-Fi) which connects the home's LAN wirelessly with WiFi. PEP and the controlling software of the electric device are installed in the BeagleBones (Ubuntu Linux OS, 256MB RAM, 720MHz ARM Cortex-A8 processor, USD-WiFi). We use two BeagleBone devices that connect the LAN wirelessly with the USB-WiFi. The BeagleBones are also supplied with a weather cape–for the measurement of environmental parameters, like temperature, ambient light, and humidity; and a battery cape–for power supply. The BeagleBoard manages a surveillance USB-camera and is plugged to the home's power supply. CompoSecReasoner and PBAC run on a laptop (Windows 8.1 Pro OS, 8GB RAM, 2.1GHz Inter Core i-7 CPU, WiFi). The testbed and demo settings are depicted in Figure 7.

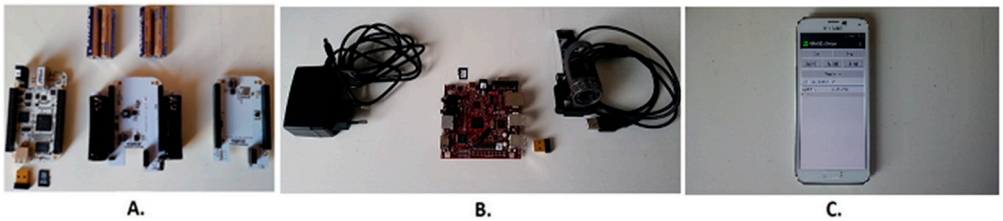

Figure 7. Smart home equipment. (**A**) BeagleBone with weather and battery capes, USB-WiFi, and SD memory card. (**B**) BeagleBoard with USB-Camera, power supply, USB-WiFi, and SD memory card. (**C**) Android smart phone device with the user's smart home application.

We utilize the smart phone application to run the benchmark where the user triggers SPD changes on the home equipment, such as the SPD configurations of PBAC and IPsec. We assess the changes' impact on the computational complexity of CompoSecReasoner as well as the overall system's responsiveness.

The RETE algorithm (one of the main and widely-utilized pattern matching algorithms for implementing rule-based systems) [79] is utilized internally by JESS, achieving optimized speed and forming a quite efficient pattern-matching engine. The CompoSecReasoner's reasoning theory for the smart home application requires around 400 facts and 30 rules. The whole reasoning framework need on average 1.6 s, 45 MB RAM, and 1.87 MB for the code. Nevertheless, this is expected to be done once, when an agent is started. After that, when the rule engine is up and running, it takes around 0.002 s to process a theory with a few hundreds of facts [80]. Therefore, this is the actual real-time delay for applications. The code size is not affected while the additional RAM is minimal.

The most notable performance factor is the response time that the user experiences when he/she retrieves information or when SPD changes are happening. The overall delay of the concurrent benchmark requests is depicted in Figure 8. Spikes reflect the SPD changes (e.g., incoming SA requests, system configuration, and notification of the SA when a change is completed). Such changes produce considerable delay, however, in normal operation the system is set in a specific configuration for most of the time and changes are expected to be rare. Nonetheless, the framework exhibits acceptable delay, even for a real-time environment.

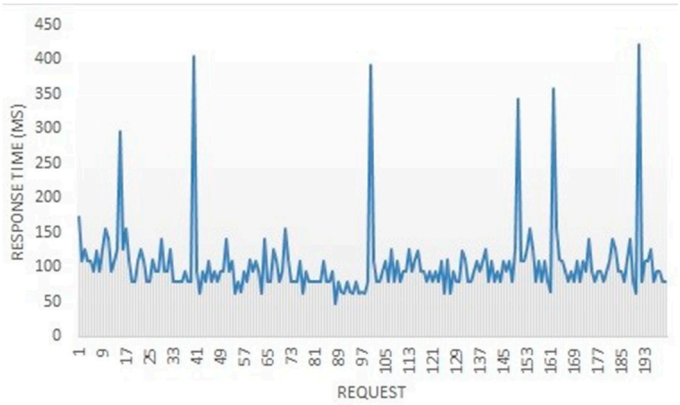

Figure 8. Performance assessment–Response time.

Regarding scalability, the reasoning complexity is affected by: (i) the number of rules (R), (ii) the number of facts (F) on the working memory, as well as, (iii) the average patterns per rule left-hand-side (P). The total complexity is linearly influenced by the size of the working memory and is of the order of $O(RFP)$. In CompoSecReasoner, the number of the theory rules (R) is very low, i.e., around *30* rules the smart home scenario. Also, unique identifiers are assigned to every entity and the upcoming events influence specifically denoted parameters, thus, the pattern-matching ratio (P) is also low and in the magnitude of one to three. Actually, scalability is influenced mostly from the facts' amount (F). In this demo, each system component takes around 5–10 facts to be defined, producing a low computational overhead (in the nanosecond range) and less than 40 extra bytes in RAM for the SPD agent.

7. Discussion

7.1. Validation of the Derived SPD Values

Apart from the functional and operational aspects of the proposed framework, it is important to examine wherever the deduced information regarding the SPD aspects of an evaluated system are meaningful and truly reflect the actual protection level. Therefore, we also compare the results of the SPD assessment procedure with similar standardized or widely-used methodologies proposed by the National Institute of Standards and Technology (NIST) [81].

As we presented in the Section 3.1, our method starts with the analysis of the individual components and the estimation of their SPD status. For the evaluation of distinct system modules, software, hardware, or other assets, NIST has established a methodology called Common Vulnerability

Scoring System (CVSS) [82]. The method takes into account known vulnerabilities for the examined component as well as the existence of mitigation mechanisms and tries to figure out their severity and likelihood to be exploited. The result is a score from 0 (low risk) to 10 (high attackability). NIST performs CVSS analyses for known vulnerabilities (e.g., in MySQL, Internet Explorer, Chrome, etc.) and maintains the results in a repository, which is publicly available [82].

Our SPD multi-metric follows a similar approach by assessing the attacker's effort and preference to exploit an existing vulnerability as well as its impact. We also take into account the potential attack vectors (called Threat Flows (TFs) in our methodology) and assess a list of known vulnerabilities (defined as limitations) that the deployed defense mechanisms (controls) may fail to mitigate. Moreover, the SPD levels are meant to be calculated in advance, as the CVSS scores, and be processed by CompoSecReasoner during the system's composition and operation.

We utilize CVSS v3.1 Calculator [83] to assess the security for the secure storage service at the device-end with the ULCL component of the smart home example. The derived CVSS base score is 2.7, reflecting low attackability and high security. This is in accordance with the SPD multi-metric assessment and the SPD level of $\langle 80, 50, 30 \rangle$, where the security ($S = 80$) also reflects high protection and low exploitation possibility. Similar results were derived for the other individual evaluated components that are referred in the Section 6.5. This fact designates that our proposal successfully discloses the main security notion as it is acceptable by the cyber-security community. Moreover, we have extended the analysis to also cover privacy and dependability in a similar fashion.

Furthermore, the proposed methodology goes one step further from the distinct analysis of assets and estimates the overall protection status of the composed system.

NIST and the Health Insurance Portability and Accountability Act (HIPAA) [84] have developed the Security Risk Assessment (SRA) tool [85] to evaluate the overall security posture of healthcare organizations. Except from technical features, the tool takes also into account procedural aspects, threat detection and response operations, as well as the human factor. We utilize this tool to evaluate the demonstrated smart home composition and correlate the results with the outcomes of our methodology. We consider the same setting that is described in the Section 6.4, with the same devices and protected assets (e.g., PEP, PDP, etc.), the contingency and response functionality of the SPD agents, and a home owner with low to moderate security knowledge as the main actuator. Figure 9 depicts the fulfillment of the risk assessment questionnaires as well as the final report of the SRA tool for the smart home setting.

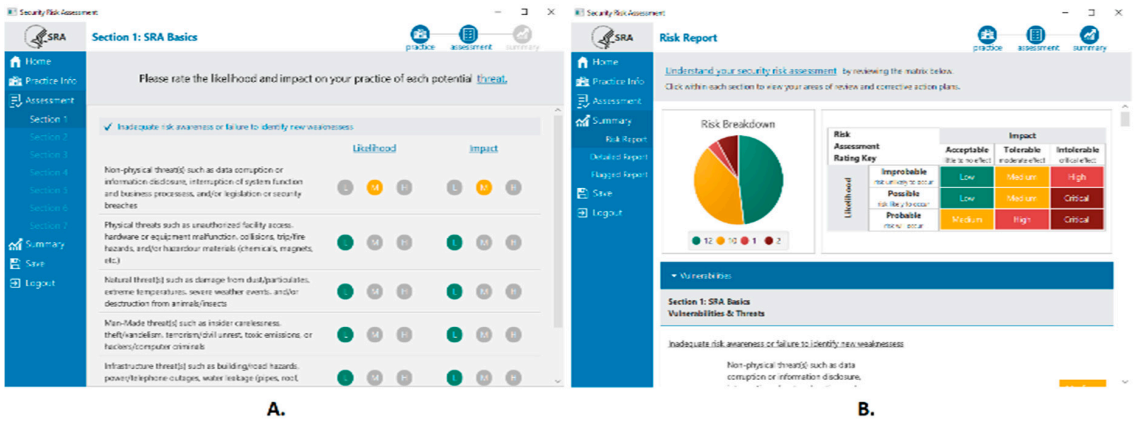

Figure 9. Smart home system evaluation with the SRA tool: (**A**). intermediate steps, (**B**). final report.

The analysis reveals that there is high risk in three of the 25 main security categories that are investigated. The technical safeguards and contingency policies are in place and the reported problems that exhibit high exploitation risk are related with the lack of security knowledge by the user (homeowner). This is also in accordance with the SPD evaluation of $\langle 90, 80, 90 \rangle$ that designates a high level of system protection. Our method is mainly assessing the technical and procedural aspects of a

system, while the incorporation of metrics that capture the security awareness of its operator (e.g., [86]) could be considered as a future work.

7.2. Future Work

The assessment of the SPD aspects for a system is usable for figuring out if it meets the desired SPD requirements and mitigates attacks under a specific threat model, and comparing various configurations and reasoning which is the best.

Nevertheless, from the engineering and business points of view, it also becomes fruitful to deduce which of the settings that achieve the targeted SPD level is the most cost-effective/profitable. Therefore, CompoSecReasoner can be extended to assess the implementation cost of the examined settings. The SPD and cost measurements could be incorporated in business frameworks for cost-benefit analyzes and investment evaluation for IoT or other information systems.

In the long-term, technology ageing also raises as a significant parameter [87,88]. In the smart home environment, the deployed equipment (e.g., surveillance cameras, fridge, etc.) is in use for several years. The long lifetime influences protection, and after decades of technology evolution, the currently deployed and secure mechanisms could have become inadequate. In the cryptography domain for instance, the outdated cipher DES was replaced by the newer cipher AES some years ago, while now crypto-systems which are based on elliptic curves are considering as the substitution of the RSA ones. Thus, an adequate assessment methodology has to consider the ageing problem of the system as well. The proposed method can tackle several of these issues. The analysis which evaluates the provided defense of the deployed controls based on the known limitations could be periodically revisited to reflect the current situation. Thereupon, vendors could keep up-to-date the SPD of their products with the system operator (or the automated management framework) re-calculating a system's SPD, when changes are reported.

8. Conclusions

In this study, we demonstrate CompoSecReasoner–an event-based model checking framework for metric-driven management of dynamic embedded systems. We use Event Calculus (EC) to describe the changes of a system with the progress of time and decide if they result in a more or less secure system. We implement the CompoSecReasoner as a reasoning behavior for JADE agents and apply it in the multi-agent filed. Then, we embody the whole framework as an OSGi bundle in Knopflerfish. The agents verify dynamically the composition of their underlying systems based on their metric-driven policies. Moreover, they communicate security, privacy, and dependability (SPD) related information to build more secure systems and reach global SPD levels. We demonstrate the CompoSecReasoner in a real ambient intelligence scenario, where embedded devices control ambient parameters of two smart buildings. We use CompoSecReasoner to reason if the composition is feasible and figure out the total outcome for SPD metrics.

Author Contributions: Conceptualization, G.H. and N.P.; methodology, G.H. and N.P.; software, G.H.; validation, S.I., I.H. and G.V.; formal analysis, N.P. and G.V.; writing—original draft preparation, G.H.; writing—review and editing, S.I., N.P. and I.H. All authors have read and agreed to the published version of the manuscript.

Funding: This work has received funding from the European Union Horizon's 2020 research and innovation programme under the grant agreements No. 786890 (THREAT-ARREST) and No. 830927 (CONCORDIA).

Conflicts of Interest: The authors declare no conflict of interest. "The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results".

Appendix A

The appendix lists the interactive and passive controls that are evaluated under the SPD multi-metric approach.

Table A1. The SPD controls that are evaluated by the SPD methodology.

Control	Description
Interactive controls	
Security	
Authentication	Challenge of credentials based on authorization and identification
Resilience	Preserve protection in cases of failure or corruption
Subjugation	Assure that interactions happen based on a defined plan, removing liability and freedom of choice of the interacting entity in case of disclosure
Continuity	Preserve activity when failure or corruption occurs
Indemnification	Formal agreement between the asset owner and other interacting entities. May involve warnings as a precursor of public legislative protection and legal action
Privacy	
Consent	Freely given, informed, and specific agreement of the personal identifiable information (PII) principal for processing the PII. PII should not be shared or disclosed to third parties without consent from the PII principal
Opt-in	Policy or procedures under which the PII principal consents for the explicit processing actions of the PII, prior to relevant consent
Indentifiability	The ability to infer the identity of the PII principal, directly or indirectly, by a given set of PII. It may subsume complete indentifiability, pseudonymization or anonymity.
Notification	Inform the PII principals whose data is being gathered regarding such collection
Dependability	
Survivability	Available degraded operations that are useful and acceptable to users when a failure occurs for a specified period
Performability	Operations that assures how well the system will perform in the presence of faults for a specified period
Removal during use	Mechanisms that record and remove faults via the maintenance cycle after the production stage
Passive controls	
Security	
Confidentiality	Assurance that a processed asset is not known outside the interacting entities
Integrity	Assurance that the interacting entities know when an asset has been modified
Non-repudiation	Obstructs the interacting entities for denying their role in an occurred interaction
Alarm	Notification that an interaction is happening or has happened
Privacy	
Fairness	PII should be collected, used or disclosed for appropriate and limited purposes
Challenge compliance (accountability)	PII principals should be able to hold PII processors accountable for complying with all privacy controls
Retention	Insurance that PII which is no longer required, is not retained in order to limit unauthorized collection, usage and disclosure
Disposal	Mechanisms for the disposal or destruction of PII
Report	Report that an interaction regarding PII is occurring or has occurred
Break/incident response	Procedure for managing a breach concerning PII
Dependability	
Tolerance	Insurance that the needed functionality will be delivered in the presence of faults
Forecasting	Prediction of possible faults so that they can be removed or their effects can be circumvented
Prevention	Obstructs faults from being integrated into a system. It is achieved by using good implementation techniques and development techniques
Removal during development	Verification of a system in order to detect and remove faults before the system is put into production

References

1. Wang, C.; Zhao, Z.; Gong, L.; Zhu, L.; Liu, Z.; Cheng, X. A Distributed Anomaly Detection System for In-Vehicle Network Using HTM. *IEEE Access* **2018**, *6*, 9091–9098. [CrossRef]
2. Hu, L.; Cheng, X.; Che, X. Survey of Grid Resource Monitoring and Prediction Strategies. *Int. J. Intell. Inf. Process.* **2010**, *1*, 78–85.

3. Olszewski, R.; Pałka, P.; Turek, A.; Kietlinska, B.; Płatkowski, T.; Borkowski, M. Spatiotemporal Modeling of the Smart City Residents' Activity with Multi-Agent Systems. *Appl. Sci.* **2019**, *9*, 2059. [CrossRef]
4. Prasad, K.V.; Giuli, T.J.; Watson, D. The Case for Modeling Security, Privacy, Usability and Reliability (SPUR) in Automotive Software. *Aswsd Springer Lncs* **2008**, *4922*, 1–14.
5. Shull, F.; Feldmann, R.L.; Seaman, C.; Regardie, M.; Godfrey, S. Fully Employing Software Inspections Data. *Innov. Syst. Softw. Eng. A Nasa J. Springer* **2012**, *8*, 243–254. [CrossRef]
6. Tabrizi, F.M.; Pattabiraman, K. Formal security analysis of smart embedded systems. In Proceedings of the Annual Computer Security Applications Conference (ACSAC), Los Angeles, CA, USA, 5–9 December 2016.
7. Tabrizi, F.M.; Pattabiraman, K. A model for security analysis of smart meters. In Proceedings of the IEEE/IFIP 42nd International Conference on Dependable Systems and Networks Workshops (DSN-W), Boston, MA, USA, 25–28 June 2012.
8. Rupp, A.; Baldimtsi, F.; Hinterwalder, G.; Paar, C. Cryptographic Theory Meets Practice: Efficient and Privacy-Preserving Payments for Public Transport. *ACM Trans. Inf. Syst. Secur. (TISSEC)* **2015**, *17*, 1–31. [CrossRef]
9. Li, J.; Li, J.; Chen, X.; Liu, Z.; Jia, C. Privacy-preserving data utilization in hybrid clouds. *Future Gener. Comput. Syst.* **2014**, *30*, 98–106. [CrossRef]
10. Jia, B.; Hao, L.; Zhang, C.; Huang, B. A Privacy-sensitive Service Selection Method Based on Artificial Fish Swarm Algorithm in the Internet of Things. *Mob. Netw. Appl.* **2020**. [CrossRef]
11. Avizienis, A.; Laprie, J.-C.; Randell, B.; Landwehr, C. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput. (Tdsc)* **2004**, *1*, 11–33. [CrossRef]
12. Natella, R.; Cotroneo, D.; Madeira, H.S. Assessing dependability with software fault injection. *Acm Comput. Surv.* **2016**, *48*, 1–55. [CrossRef]
13. Cinque, M.; Cotroneo, D.; Pecchia, A. Enabling effective dependability evaluation of complex systems via a rule-based logging framework. *Int. J. Adv. Softw.* **2009**, *2*, 323–336.
14. Lei, C.; Zhang, H.-Q.; Tan, J.-L.; Zhang, Y.-C.; Liu, X.-H. Moving Target Defense Techniques: A Survey. *Secur. Commun. Netw.* **2018**, *2018*, 1–25. [CrossRef]
15. JADE Framework. Available online: http://jade.tilab.com/ (accessed on 14 July 2020).
16. OSGi Alliance. Available online: http://www.osgi.org/ (accessed on 14 July 2020).
17. OASIS, DPWS. Available online: http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.pdf (accessed on 14 July 2020).
18. Asghari, P.; Rahmani, A.M.; Javadi, H.H.S. Service composition approaches in IoT: A systematic review. *J. Netw. Comput. Appl.* **2018**, *120*, 61–77. [CrossRef]
19. Aoudia, I.; Benharzallah, S.; Kahloul, L.; Kazar, O. Service composition approaches for internet of things: A review. *Int. J. Commun. Netw. Distrib. Syst.* **2019**, *23*, 194–230.
20. UML. Available online: https://www.uml.org/ (accessed on 14 July 2020).
21. Brambilla, M.; Fraternali, P. *Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML*; The MK/OMG Press: Burlington, MA, USA, 2014; pp. 1–422.
22. Object Management Group (OMG), 1989–2020. Available online: https://www.omg.org/ (accessed on 14 July 2020).
23. Brambilla1, M.; Umuhoza1, E.; Acerbis, R. Model-driven development of user interfaces for IoT systems via domain-specific components and patterns. *J. Internet Serv. Appl.* **2017**, *8*, 1–21.
24. Souri, A.; Norouzi, M. A State-of-the-Art Survey on Formal Verification of the Internet of Things Applications. *J. Serv. Sci. Res.* **2019**, *11*, 47–67. [CrossRef]
25. Sztipanovits, J.; Karsai, G. Model-integrated computing. *IEEE Comput.* **1997**, *30*, 110–112. [CrossRef]
26. Kelly, S.; Tolvanen, J.-P. *Domain-Specific Modeling: Enabling Full Code Generation*; Wiley-IEEE Computer Society Pr.: Hoboken, NJ, USA, 2008; pp. 1–444.
27. Society of Automotive Engineers (SAE), 1905–2020. Available online: https://www.sae.org/ (accessed on 14 July 2020).
28. SAE. Architecture Analysis & Design Language. SAE Standard AS-5506. 2004. Available online: www.sae.org/standards/content/as5506c/ (accessed on 14 July 2020).
29. Murugesan, A.; Whalen, M.W.; Rayadurgam, S.; Heimdahl, M.P.E. Compositional Verification of a Medical Device System. In Proceedings of the ACM SIGAda annual conference on High integrity language technology (HILT), Pittsburgh, PA, USA, 10–14 November 2013; pp. 51–64.

30. Szemethy, T.; Karsai, G. Platform modeling and model transformation for analysis. *J. Univers. Comput. Sci.* **2004**, *10*, 1383–1406.
31. Yamaoka, H.; Itakura, K.; Takahashi, E.; Nakagawa, G.; Michaelis, J.; Kanemasa, Y.; Ueki, M.; Matsumoto, T.; Take, R.; Tanie, S.; et al. Dracena: A Real-Time IoT Service Platform Based on Flexible Composition of Data Streams. In Proceedings of the IEEE/SICE International Symposium on System Integration, Paris, France, 14–16 January 2019; pp. 596–601.
32. Incki, K.; Ari, I. A Novel Runtime Verification Solution for IoT Systems. *IEEE Access* **2018**, *6*, 13501–13512. [CrossRef]
33. Shelby, Z.; Hartke, K.; Bormann, C. The Constrained Application Protocol (CoAP). *Internet Eng. Task Force (IETF) RFC7252* **2014**. [CrossRef]
34. Larson, B.R.; Chalin, P.; Hatcliff, J. BLESS: Formal Specification and Verification of Behaviors for Embedded Systems with Software. In Proceedings of the NASA Formal Methods Symposium, Moffett Field, CA, USA, 14–16 May 2013; pp. 276–290.
35. Krishna, A.; Pallec, M.L.; Mateescu, R.; Noirie, L.; Salaun, G. IoT Composer: Composition and Deployment of IoT Applications. In Proceedings of the IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), Montreal, QC, Canada, 25–31 May 2019; pp. 19–22.
36. Kushilevitz, E.; Lindell, Y.; Rabin, T. Information-theoretical secure protocols and security under composition. *Slam J. Comput.* **2010**, *39*, 2090–2112.
37. de Albuquerque, J.P.; Krumm, H.; de Geus, P.L. Formal validation of automated policy refinement in the management of network security systems. *Int. J. Inf. Secur.* **2010**, *9*, 99–125. [CrossRef]
38. Kidron, D.; Lindell, Y. Impossibility results for universal composability in public-key models and with fixed inputs. *J. Cryptol.* **2011**, *24*, 517–544. [CrossRef]
39. Kulik, T.; Tran-Jørgensen, P.W.V.; Boudjadar, J.; Schultz, C. A Framework for Threat-driven Cyber Security Verification of IoT Systems. In Proceedings of the IEEE International Conference on Software Testing, Verification and Validation Workshops, Västerås, Sweden, 9–13 April 2018; pp. 89–97.
40. Alloy Analyzer. Available online: https://alloytools.org/ (accessed on 14 July 2020).
41. Eby, M. Integrating Security Modeling into Embedded System Design. Master's Thesis, Vanderbilt University, Nashville, TN, USA, 2007.
42. Bielova, N. A Theory of Constructive and Predictable Runtime Enforcement Mechanisms. Ph.D. Dissertation, University of Trento, Trento, Italy, November 2011.
43. Savola, R.M.; Heinonen, P. A visualization and modeling tool for security metrics and measurements management. In Proceedings of the 2011 Information Security for South Africa, Johannesburg, South Africa, 15–17 August 2011; pp. 1–8.
44. Ko, H.; Jin, J.; Keoh, S.L. Secure Service Virtualization in IoT by Dynamic Service Dependency Verification. *IEEE Internet Things J.* **2016**, *3*, 1006–1014. [CrossRef]
45. Walter, M.; Trinitis, C. Quantifying the security of composed systems. In Proceedings of the International Conference on Parallel Processing and Applied Mathematics (PPAM'05), Poznan, Poland, 11–14 September 2005; Springer: Berlin/Heidelberg, Germany, 2006; pp. 1026–1033.
46. Leuprecht, C.; Skillicorn, D.B.; Tait, V.E. Beyond the Castle Model of cyber-risk and cyber-security. *Gov. Inf. Q.* **2016**, *33*, 250–257. [CrossRef]
47. Schannep, J.H.; Doukas, J.C.; Song, S.C. Advancing cybersecurity from Medieval Castles to STrategic Deterrence: A Systems Approach to cybersecurity. In Proceedings of the International Annual Conference of the American Society for Engineering Management, Coeur d'Alene, ID, USA, 17–20 October 2018; pp. 1–10.
48. Theisen, C.; Munaiah, N.; Al-Zyoud, M.; Carver, J.C.; Meneely, A.; Williams, L. Attack surface definitions: A systematic literature review. *Inf. Softw. Technol.* **2018**, *104*, 94–103. [CrossRef]
49. Younis, A.A.; Malaiya, Y.K.; Ray, I. Using Attack Surface Entry Points and Reachability Analysis to Assess the Risk of Software Vulnerability Exploitability. In Proceedings of the IEEE 15th International Symposium on High-Assurance Systems Engineering, Miami Beach, FL, USA, 9–11 January 2014; pp. 1–8.
50. Theisen, C.; Murphy, B.; Herzig, K.; Williams, L. Risk-Based Attack Surface Approximation: How Much Data is Enough? In Proceedings of the IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track, Buenos Aires, Argentina, 20–28 May 2017; pp. 271–280.

51. Yoon, C.; Lee, S.; Kang, H.; Park, T.; Shin, S.; Yegneswaran, V.; Porras, P.; Gu, G. Flow Wars: Systemizing the Attack Surface and Defenses in Software-Defined Networks. *IEEE/Acm Trans. Netw.* **2017**, *25*, 3514–3530. [CrossRef]
52. Manadhata, P.K.; Wing, J.M. An attack surface metric. *IEEE Trans. Softw. Eng. (Tse)* **2010**, *37*, 371–386. [CrossRef]
53. Howard, M.; Corporation, M. Determining Relative Attack Surface. U.S. Patent US 7299497 B2, 20 November 2007.
54. Eguia, I.; Ser, J.D. A Meta-Heuristically Optimized Fuzzy Approach towards Multi-Metric Security Risk Assessment in Heterogeneous System of Systems. In Proceedings of the MeSeCC, Lisbon, Portugal, 7–9 January 2014.
55. Chen, J. Dynamic Cyber Defence Framework. *J. Inf. Warf.* **2016**, *15*, 46–55.
56. Albanese, M.; Battista, E.; Jajodia, S.; Casola, V. Manipulating the Attacker's View of a System's Attack Surface. In Proceedings of the IEEE Conference on Communications and Network Security, San Francisco, CA, USA, 29–31 October 2014; pp. 472–480.
57. Savola, R.M.; Sihvonen, M. Metrics Driven Security Management Framework for E-Health Ecosystem Focusing on Chronic Diseases. In Proceedings of the International Conference on Management of Emergent Digital EcoSystems, Addis Ababa, Ethiopia, 28–31 October 2012; pp. 75–79.
58. Krautsevich, L.; Martinelli, F.; Yautsiukhin, A. Formal approach to security metrics. What does "more secure" mean to you? *ECSA* **2010**, *10*, 162–169.
59. Hatzivasilis, G. Multi-agent distributed epistemic reasoning in ambient intelligence environments. Master's Thesis, University of Crete, Greece—FORTH-ICS, Heraklion, Greece, November 2011.
60. Madl, G.; Abdelwahed, S. Model-based analysis of distributed real-time embedded system composition. In Proceedings of the 5th ACM international conference on Embedded software, EMSOFT'05, Jersey City, NJ, USA, 19–22 September 2005; pp. 371–374.
61. Hatzivasilis, G.; Papaefstathiou, I.; Manifavas, C. Software Security, Privacy and Dependability: Metrics and Measurement. *IEEE Softw.* **2016**, *33*, 46–54. [CrossRef]
62. ISECOM. Open Source Security Testing Methodology Manual. 1988–2015. Available online: https://www.isecom.org/OSSTMM.3.pdf (accessed on 14 July 2020).
63. ISECOM. Common Criteria for Information Security Evaluation. 1996–2015. Available online: http://www.commoncriteriaportal.org (accessed on 14 July 2020).
64. ISO/IEC. Code of Practice for Protection of Personally Identifiable Information (PII) in Public Clouds Acting as PII Processors. ISO/IEC 27018. 2014. Available online: http://www.iso.org/iso/catalogue_detail?csnumber=61498 (accessed on 14 July 2020).
65. ISO/IEC. Privacy framework. ISO/IEC 29100. 2011. Available online: https://www.iso.org/obp/ui/#iso:std:iso-iec:29100:ed-1:v1:en (accessed on 14 July 2020).
66. IEC. International Standard on Dependability. IEC 60300. 2017. Available online: https://tc56.iec.ch/dependability-standards/ (accessed on 14 July 2020).
67. EU Funded Project—nSHIELD: New Embedded Systems arcHItecturE for Multi-Layer Dependable Solutions. Available online: https://artemis-ia.eu/project/34-nshield.html/ (accessed on 14 July 2020).
68. Cesena, M. SHIELD Technology Demonstrators. In *Measurable and Composable Security, Privacy, and Dependability for Cyberphysical Systems*; CRC Press: Boca Raton, FL, USA, 2017; pp. 381–434.
69. Muller, E.T. *Commonsense Reasoning*; Morgan Kaufmann Publishers: Burlington, MA, USA, 2010.
70. Mantas, G.; Lymberopoulos, D.; Komninos, N. Security in smart home environment. In *Wireless Technologies for Ambient Assisting Living and Healthcare: Systems and Applications*; IGI Global: Hershey, PA, USA, 2010; pp. 170–191.
71. Rantos, K.; Fysarakis, K.; Manifavas, C.; Askoxylakis, I. Policy-controlled authenticated access to LLN-connected healthcare resources. *IEEE Syst. J.* **2015**, *12*, 92–102. [CrossRef]
72. Hatzivasilis, G.; Gasparis, E.; Theodoridis, A.; Manifavas, C. ULCL: An Ultra-Lightweight Cryptographic Library for embedded systems. In Proceedings of the MeSeCCS, Lisbon, Portugal, 7–9 January 2014; pp. 11–18.
73. BeagleBone.org. BeagleBone Device Manual. Available online: http://beagleboard.org/static/beaglebone/a3/Docs/Hardware/BONE_SRM.pdf (accessed on 14 July 2020).

74. Patkos, T.; Plexousakis, D. DECKT: Epistemic reasoning for ambient intelligence. In *ERCIM News Magazine*; ERCIM: Valbonne, France, 11 January 2011.
75. FIPA. Available online: http://www.fipa.org/ (accessed on 14 July 2020).
76. FIPA, ACL. Available online: http://en.wikipedia.org/wiki/Agent_Communication_Language (accessed on 14 July 2020).
77. Makewave, Knopflerfish. Available online: http://www.knopflerfish.org/ (accessed on 14 July 2020).
78. Strzałek, M.; Pałka, P. The issue of confidentiality, authentication, integrity and data non-repudiation in the multiagent systems. *Studia Inform.* **2012**, *33*, 217–227.
79. Berstel, B. Extending the RETE algorithm for event management. In Proceedings of the IEEE 9th International Symposium on Temporal Representation and Reasoning, Manchester, UK, 7–9 July 2002; pp. 49–51.
80. Malcolm, Y. A Federated Agent-Based Crowd Simulation Architecture. In Proceedings of the 21st European Conference on Modelling and Simulation (ECMS), Prague, Czech Republic, 4–6 June 2007; pp. 1–7.
81. NIST. National Institute of Standards and Technology. USA, 1901–2020. Available online: https://www.nist.gov/ (accessed on 14 July 2020).
82. TNIS. Common Vulnerability Scoring System (CVSS). NIST, 2019. Available online: https://www.first.org/cvss/ (accessed on 14 July 2020).
83. NIST. CVSS Calculator. NIST, 2019. Available online: https://www.first.org/cvss/calculator/3.1 (accessed on 14 July 2020).
84. Brian, A.; Fox, K.; Daniel, M. The Politics of the Health Insurance Portability and Accountability Act. *Health Aff.* **1997**, *16*, 146–150.
85. TNIS; AHIPA. Security Risk Assessment Tool (SRA). NIST, 2019. Available online: https://www.healthit.gov/topic/privacy-security-and-hipaa/security-risk-assessment-tool (accessed on 14 July 2020).
86. Manifavas, C.; Fysarakis, K.; Rantos, K.; Hatzivasilis, G. DSAPE—Dynamic Security Awareness Program Evaluation. In Proceedings of the International Conference on Human Aspects of Information Security, Privacy, and Trust, Heraklion, Crete, Greece, 22–27 June 2014; Volume 8533, pp. 258–269.
87. Cotroneo, D.; Natella, R.; Rietrantuono, R.; Russo, S. A survey of software aging and rejuvenation studies. *Acm J. Emerg. Technol. Comput. Syst.* **2014**, *10*, 1–34. [CrossRef]
88. Kapica, J.U.S. Exploits Iraq's Aging Cryptography. Globe and Mail Update. 2013. Available online: http://www.theglobeandmail.com/technology/us-exploits-iraqs-aging-cryptography/article20448490/ (accessed on 14 July 2020).