Architectural Patterns for Secure IoT Orchestrations

Konstantinos Fysarakis*, George Spanoudakis*, Nikolaos Petroulakis†, Othonas Soultatos†,
Arne Bröring‡ and Tobias Marktscheffel§
*Sphynx Technology Solutions AG, Zug, Switzerland, †Institute of Computer Science,
Foundation for Research and Technology Hellas, Iraklio, Greece, ‡Siemens AG, Corporate Technology
, Munich, Germany, §University of Passau, Passau, Germany

***Abstract*—The vast amount of connected devices on the Internet of Things (IoT) creates an enormous potential for new applications, by leveraging synergies arising through the convergence of consumer, business and industrial Internet, and creating open, global networks connecting people, data, and "things". In this context, the SEMIoTICS project aims to develop a pattern-driven framework, built upon existing IoT platforms, to enable and guarantee secure and dependable actuation and semi-autonomic behaviour in IoT/Industrial IoT applications. To achieve this, patterns are used to encode proven dependencies between the security, privacy, dependability and interoperability (SPDI) properties of individual smart objects and corresponding properties of orchestrations (composition) involving them. This paper sketches this approach followed by SEMIoTICS, whereby the SPDI patterns are used to generate IoT orchestrations with proven SPDI properties at design time, while at runtime these properties are monitored in real-time, across system layers, triggering adaptations to return the deployed orchestration to the desired SPDI state, when needed.**

***Index Terms*—security modeling; pattern-based engineering; internet of things;**

I. Introduction

The introduction of massive numbers of interconnected smart devices in the IoT era creates significant potential in various domains, such as industrial and healthcare environments [1] [2]. Nevertheless, early adopters are faced with numerous challenges [3] stemming from the intricacies of the IoT environment, such as their dynamicity, scalability, and heterogeneity, as well as the end-to-end security, privacy and Quality of Service (QoS) requirements of each of these applications domains.

Motivated by the above, project SEMIoTICS (https://www.semiotics-project.eu/) aims to enable and guarantee secure and dependable actuation and semi-autonomic behaviour in IoT/IIoT applications, through a pattern-driven approach. Patterns are re-usable solutions to common problems and building blocks to architectures. The work presented herein focuses on the development of patterns for orchestration of smart objects and IoT platform enablers in IoT applications with guaranteed security, privacy, dependability and interoperability (SPDI) properties. The achievement of this objective is based on developing patterns defining generic ways for integrating and orchestrating different types of smart objects and components that can guarantee specific SPDI properties, henceforth referred to as SPDI patterns. This guarantee is based on verification of the individual properties, via testing or monitoring evidence, a certificate, and/or formal verification, as appropriate for the type of properties and the components orchestrated by the pattern.

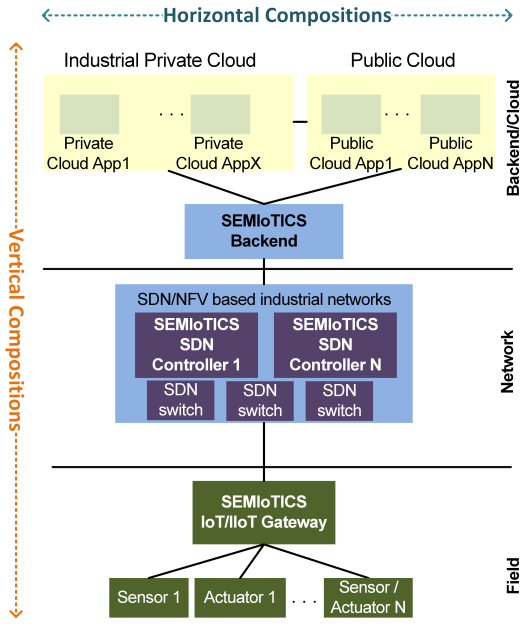

Figure 1. High-level view of typical SEMIoTICS deployment

This paper is organised as follows: Section II presents the concept and the key building blocks of the proposed pattern-driven IoT service orchestration solution; Section III provides an example of the use of the solution to guarantee integrity in the context of a wind park IIoT use case; Section IV presents related works in the key expertise domains involved, and; Section V features the concluding remarks and pointers to future work.

II. Concept & Approach

Considering the dynamicity, scalability and heterogeneity of IoT and IIoT ecosystems , as well as the ever-present need for end-to-end security guarantees, the SEMIoTICS pattern-driven approach offers a holistic approach covering the Security, Privacy, Dependability and Interoperability properties of IoT/IIoT systems. The pattern framework is designed to cover said properties from both the service orchestration and actual cyber-physical deployment perspectives. The latter spans the backend/cloud systems as well as the services and

applications running there, the field layer of a typical IoT/IIoT deployment where sensors and actuators reside, as well as the Software Defined Networks (SDN) interconnecting the two (see Figure 1). Moreover, SPDI patterns are leveraged to cover both vertical composition of smart objects at different layers in the implementation stack of IoT applications – including sensors/actuators, network, infrastructure, IoT platform and IoT application components – and horizontal composition of smart objects that appear at any of these layers, as necessary.

The SPDI guarantees are achieved through the verification of the corresponding properties, which takes place both at design time, to verify global and/or local level orchestrations for desired properties based on a model, as well as on run time, to verify based on registries and the monitoring conditions (e.g., that monitoring conditions required by patterns are satisfied). The verification process can also trigger adaptations, that should by definition guarantee the pursued properties.

To provide a usable solution tailored to the intricacies of IoT/IIoT environments, the pattern components are augmented with service orchestration definition capabilities, as well as real-time monitoring and adaptation mechanisms. The key building blocks of the provided solution are detailed in the subsections below.

A. SPDI Property Verification

Pattern-based encoding enables the verification that smart object orchestrations satisfy certain SPDI properties, and the generation (and adaptation) of orchestrations in ways that are guaranteed to satisfy required SPDI properties.

In more detail, machine interpretable SPDI patterns support: (i) the composition structure of the IoT applications and platform components; (ii) the end-to-end SPDI properties guaranteed by the pattern; (iii) the smart object/component level SPDI properties required for the end-to-end SPDI properties to hold; (iv) conditions about pattern components that need to be monitored at runtime to ensure; (v) end-to-end SPDI properties; and (vi) ways of adapting and/or replacing individual IoT application smart objects/components that instantiate the pattern if it becomes necessary at runtime (e.g., when some components become unavailable). e Patterns cover all SPDI properties, including the six core property types, namely Security (S), broken down to Confidentiality, Integrity and Availability, as well as Privacy (P), Dependability (D) and Interoperability (I). Moreover, all three data states are considered, i.e. Data-in-transit, Data-at-rest, and Data-in-processing, as well as two cases of IoT platform connectivity, namely within the SEMIoTICS platform and across IoT platforms; the latter to support SEMIoTICS interactions with other IoT platforms.

To enable the above, the SEMIoTICS Pattern Language is under development, which: (i) provides constructs for expressing/encoding dependencies between SPDI properties at the component and at the composition/orchestration level; (ii) is structural and does not prescribe exactly how the functions should be executed nor, e.g., how the ports ensure communication; (iii) it supports the static verification of SPDI properties; (iv) it is automatically processable by the SEMIoTICS framework so that IoT applications can be adapted at runtime.

Patterns expressed in this language will enable the pattern-based management process followed in SEMIoTICS, whereby patterns will be leveraged to encode proven dependencies between security, privacy, dependability and interoperability (SPDI) properties of components and corresponding properties of compositions involving them. The conjunction of the latter properties is known/proven to entail end-to-end properties. The encoding of such dependencies between orchestration and component level SPDI properties enables the following during the design time of an IoT orchestration and at runtime during the execution of an IoT orchestration: (i) The verification at design time that an orchestration satisfies certain SPDI properties; (ii) The runtime adaptation of orchestrations in ways that are guaranteed to satisfy required SPDI properties at the orchestration level. The latter adaptations may take three forms: (1) to replace particular components of an orchestration, (2) to change the structure of an orchestration, and (3) a combination of (1) and (2).

In the work presented herein SPDI patterns cover the different and heterogeneous orchestration models required for IoT and IIoT applications, including message-driven, event-driven and data-driven models [4].

B. Recipes for IoT Orchestrations

When new devices are added to an IoT/IIoT environment, they need to be connected physically, and the software on the centralized controller needs to be re-parameterized and reconfigured. Manually designing composite services is time-consuming and error-prone, but also unfeasible when considering the scale of IoT deployments.

Therefore, when designing the composition of services and devices, the *Recipes* approach is followed to separate the design of an IoT service composition from its implementation [5]. Recipes are used to define an abstract template of an application, allowing multiple instantiations and reuse, easy configuration of applications and automated discovery of matching devices. A semi-automated service composition and instantiation tool is provided, in order to assist the user in creating the composition of placeholders for actual services and devices. Later, these placeholders are replaced with actual services and devices based on suggestions provided by the system using semantic reasoning.

In more detail, Recipes define templates for compositions of *Ingredients* and their interactions. Ingredients are placeholders for devices and services that process and transform data. Interactions describe the dataflow between these ingredients. The Recipe model is a light-weight semantic model that describes ingredients and interactions semantically. An ingredient of a Recipe specifies the requirements that should be fulfilled by concrete devices/services in order to create an application. An Interaction between the ingredients is defined by creating a dataflow between them. That is, by connecting the output data of one ingredient with the input data of another ingredient.

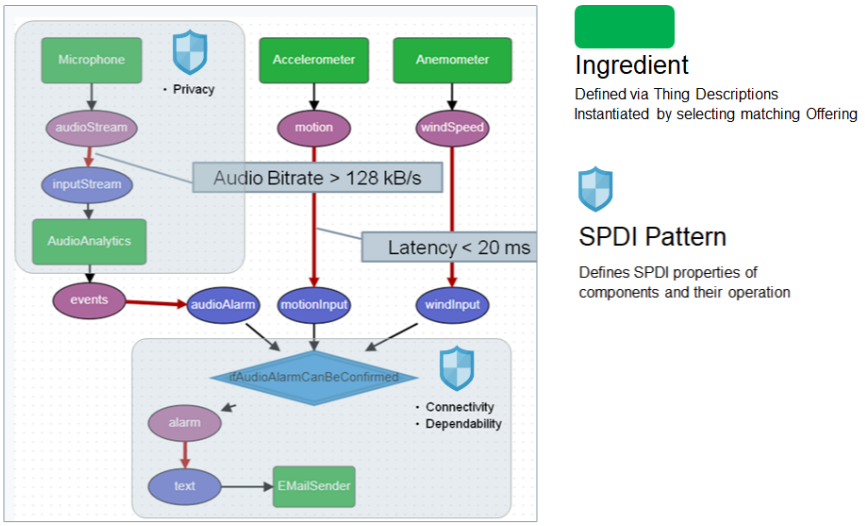

Figure 2. Example of Recipe Definition

In addition to this, an interaction also specifies an operation that needs to be performed to access the ingredient's data or function. A draft of the user interface for the specification of Recipes is depicted in Figure 2.

Previous works on Recipes are exploited in this regard, featuring a script-based centralized orchestrator approach [5], as well as an extension of this approach enabling the distributed execution of instantiated Recipes as choreographies [6]. Moreover, the integration of Recipes with SPDI patterns allows for design-time definition of desired SPDI properties for the IoT orchestrations, as well as the real-time monitoring of said properties at runtime, triggering adaptations, if needed.

C. Runtime Monitoring & Adaptation

Ensuring that IoT applications are secure at runtime is a challenging task due to the complexity and heterogeneity of the sensing, communication and computational infrastructures involved, the changing IoT threat landscape, and the plethora of non-interoperable monitors operating at different layers.

Nevertheless, monitoring the operation of smart objects and IoT applications at runtime is necessary for: (i) ascertaining that conditions, which are necessary for the preservation of the SPDI properties required of them are preserved, and (ii) maintaining an awareness of their operational context that can aid the selection of appropriate adaptation actions for them when the need arises. Existing IoT application enabling platforms offer comprehensive monitoring capabilities, however these offer either standard built-in checks or platform specific languages for defining different checks of specific types (e.g., intrusion or performance checks). Hence, checks realizable in one IoT platform are difficult to be ported to other platforms when dynamic adaptations in the smart objects and structures of IoT applications (e.g., addition and departure of smart devices) occur.

Therefore, SPDI-driven runtime monitoring and adaptations must be enabled by seamless, extensible and adaptive monitoring. This can be achieved through a monitoring management layer for: (a) instantiating the parametric monitoring conditions of SPDI patterns into concrete monitoring conditions regarding the particular smart objects that instantiate a pattern; (b) checking the monitorability of such conditions across different IoT platforms and creating optimal monitoring strategies for this purpose; (c) configuring automatically the monitors of IoT enabling platforms as required for different monitoring strategies; (d) fusing the results of different monitors (possibly in different platforms) as necessary for the checks; and (e) seamlessly adapting the monitoring strategies and monitoring configurations of different IoT enabling platforms following changes in IoT applications and smart objects to enable continuous uninterrupted monitoring.

III. Application Example

As an application example, let us consider an IIoT deployment at a wind park [7], where Data captured by an IIoT sensor on a Wind Turbine triggers, after analytics, a change in an actuator's state (e.g., to avoid possible wind turbine failure). We further assume that the overarching goal of the system owners is to have verifiable end-to-end integrity throughout this process. This scenario is visualized in Figure 3, while the interaction steps involved in this process are: (i) Sensor $P1$ captures and transmits data to IIoT Gateway; (ii) Gateway $P2$ performs analytics on data; (iii) Gateway $P2$ transmits processed data to backend through the SDN network; (iv) Backend system $P3$ performs enhanced analytics on received data; (v) Actionable output is sent from Backend isntance $P3$ to the Gateway $P2$; (vi) Relevant data/processing results stored in Cloud Data Store $P4$ (e.g., to data historian application, for auditing purposes); (vii) Gateway $P2$ transmits trigger command to actuator $P5$.

As depicted in the right side of Figure 3, the end-to-end integrity property can be broken down into individual integrity properties, covering different data states, to be pursued at

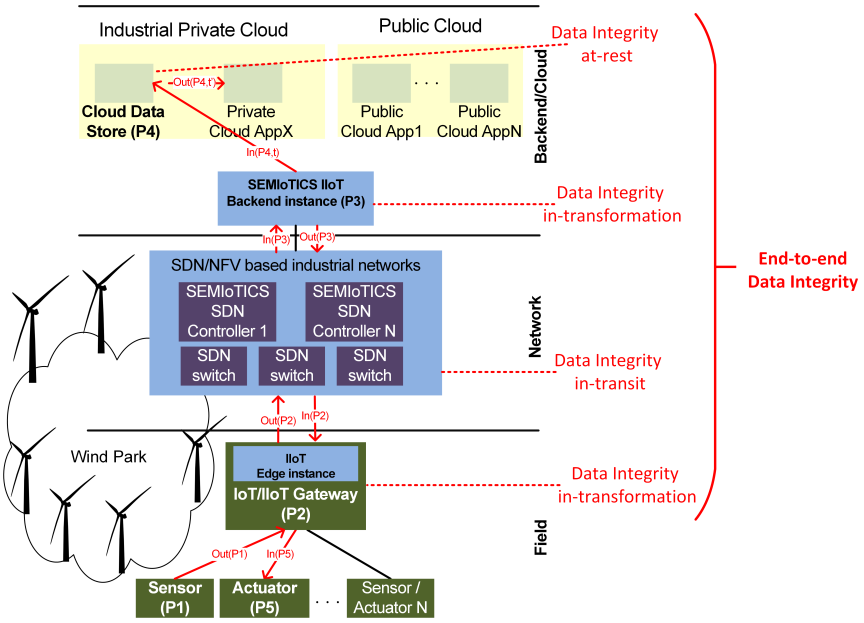

Figure 3. End-to-end Integrity in Wind Park scenario

design time and monitored/verified at runtime. To simplify the example, and without loss of generality, we assume that the connections between sensors, actuators and the Gateway, as well as the ones between the Backend and the Cloud services, are protected by secure communication protocols with integrity provisions, and thus we do not examine the corresponding properties. A high level definition of the integrity properties of interesting, using the notation for inputs and outputs shown in the figure, is as follows:

- Data Integrity in-transformation:
  1) $Out(P2) = F(Out(P1))$
  2) $Out(P3) = F(In(P3))$
  3) $In(P5) = F(In(P2))$
  4) $In(P4) = F(In(P3) \wedge Out(P3))$
- Data Integrity in-transit:
  5) $In(P3) = Out(P2)$
  6) $In(P2) = Out(P3)$
- Data Integrity at-rest:
  7) $Out(P4, t') = In(P4, t)$
- End-to-end Data Integrity:
  8) $In(P5) = F(Out(P1))$

The above covers integrity in all data transformations that happen at the Gateway $P2$ (properties #1, #3) and the Backend $P3$ (properties #2, #4), as well as the exchange of information between Gateway and Backend through the SDN communication infrastructure (properties #5 and #6). Moreover, data integrity at rest is covered (property #7), where we assume that a record of the input and output of the Backend system is stored in a private cloud data store $P4$ at time t for auditing purposes, and this is retrieved by another application at time t'. Finally, the end-to-end property is also defined for the whole process (property #8), which only holds if all the individual properties defined above (with the exception of property #7, which refers to future interactions) also hold.

In terms of verification of the individual properties, these can be based on testing or monitoring evidence, a certificate that a component may hold, or formal verification, as appropriate for the type of properties and the smart objects/components involved. Some hypothetical cases for each of the properties will be presented here, aiming to cover diverse types of verification.

For data integrity in transformation taking place at the Gateway (component $P2$, properties #1 and #4), we assume that it comes with a static certificate by the manufacturer, for the properties carried out internally. An interface should be declared, where the validity of the certificate can be verified.

In contrast, in the case of transformations taking place at the backend (component $P3$, property #2 and #3), we assume that the IIoT Backend instance has a dynamic certificate, which is verified at runtime by monitoring internal processes. To achieve this, data integrity must be monitored internally at all compute nodes participating in the computation, and these nodes must have event emission capabilities, therefore the application code will have to be decorated to enable this (e.g., with internal hash checks in critical parts of the internal program functions). An example of such a monitoring condition that could be evaluated internally could be:

– $Happens(writes(n1, M1), t1, [t1, t1]) \wedge$
$HoldsAt(Hash(M1, h1), t1)$
– $Happens(reads(n2, M1), t2, [t1, t2]) \wedge$
$HoldsAt(Hash(M1, h2), t2)$
$\rightarrow h2 = h1$

In the above, $n1$ is the data being stored in memory position $M1$ at $t1$, with a hash of $h1$, and $n2$ is the data being read from the same memory position at $t2$. So, if the two hashes,

$h1$ and $h2$ pass the match check, then the integrity property holds within the processing functions.

In the case of the integrity in transit property at the SDN network (properties #5 and #6), we can apply a sequential integrity in-transit pattern that applies to serial communication between 3 entities, $(C1, C2, C3)$. Therefore, serial communication $C1 \rightarrow C2 \rightarrow C3$ has integrity, when: $C1$ has integrity; $C1 \rightarrow C2$ communication has integrity; $C2 \rightarrow C3$ communication has integrity.

Then, the individual properties a, b, and c can be verified independently, using one of the methods defined above. It should be noted that this sequential pattern can be applied to serial communications across layers (e.g., communication from edge, to SDN and backend, as in the scenario here), as well as within layers (e.g., within the SDN layer, to verify properties in exchanges between the SDN Controller and network switches under its control).

In addition to the above, a simple monitoring-based verification rule can be defined for the integrity of data at rest (property #7), where a hash is calculated when data is stored and when it is retrieved, checking that the two hashes match (to verify integrity). This monitoring could be implemented through an external monitoring agent, or within the data store application, in cases where access to the source code is available.

Finally, as an example of a potential adaptation that could be triggered in this scenario we can consider the communication within SDN which, when verifying properties #5 and #6, is decomposed to a serial communication such as $VNF1 \rightarrow vSwitch1 \rightarrow VNF2$. Then, when applying Sequential Integrity Pattern (as previously presented), $vSwitch1$ is found not to satisfy the integrity requirement (e.g., its certificate has expired). This could trigger an adaptation action at the SDN Controller (where the SDN pattern engine resides) to reroute traffic within the SDN network to skip $vSwitch1$ and use $vSwitch2$ instead, which has a valid certificate. The resulting serial communication $VNF1 \rightarrow vSwitch2 \rightarrow VNF2$ would satisfy the integrity property, and consequently the end-to-end integrity property (assuming all other integrity property checks verified as true), returning the system to the desired state.

IV. Related Work

Property Verification: The pattern-driven approach of SEMIoTICS follows the *"security-by-design"* concept, which aims to guarantee system-wide security properties by virtue of the design of the involved systems and their subsystems. This is leveraged to provide orchestration-level SPDI guarantees, while encompassing all involved components and entities which are composed to create the orchestrations (e.g., physical devices and software). A key capability required in security-by-design is the ability to verify the desired security properties as part of the design process. A typical way to achieve this is using model-based techniques [8], [9], [10], whereby software component and service compositions are modelled using formal languages and the required security properties are expressed as properties on the model [11]. The satisfiability of the required properties is based on model checking [12], [13]. Other approaches focus on software service workflows using business process modelling languages (e.g., Sec-MoSC [14]). Pino et al. [15] use secure service orchestration (SSO) patterns to support the design of service workflows with required security properties, leveraging pattern-based analysis to verify security properties. This avoids full model checking that is computationally expensive and non-scalable to larger systems, such as the IoT. Moreover, some model-based approaches (e.g., [15]) support the transformation of security requirements to code for automated checking of the required properties, both at design and at runtime. The SEMIoTICS pattern-driven framework's operation is inspired from similar pattern-based approaches used in service-oriented systems [4], [16], cyber-physical systems [17], and networks [18], [19], covering more aspects in addition to Security, and also providing guarantees and verification capabilities that span both the service orchestration and deployment perspectives, as detailed in Section II above.

Service Composition: There have been attempts to fully automate the generation of service compositions based on some user defined request or goal. For example [20] present a service composition system that enables the goal-driven configuration of smart environments based on semantic metadata and reasoning. Such fully automated approaches are still facing challenges. The key difficulty lies in the unambiguous semantic description of the goal and states that lead to the goal, which is challenging when dealing with more complex environments, where a lot of devices can interact in a lot of ways and each has a lot of possible states. In such environments semantic models become complex, and reasoners turn to be inefficient when solving goals over them. Due to these challenges, a semi-automated approach that supports users in creating service compositions, rather than to fully automate the process, seems to be the most promising direction. Previous work has been done in this direction, such as [21], where optimal service compositions are automatically computed with support of composition templates. Commercially successful systems such as "If This Then That" (at ifttt.com) use simple composition techniques similar to the Recipe concept, but create and execute centralized orchestrations instead of decentralized choreographies [22]. The IFTTT platform lacks systematic engineering support leading to widely duplicated recipes [23]. Node-RED is another tool that follows a similar approach, through a browser-based editor that makes it easy to wire hardware devices, APIs and online services, thereby creating application flows (specified in JSON). In this work a semi-automated approach is followed for the definition of IoT service compositions, via the integration of dependability patterns (in addition to security, privacy and interoperability), as detailed above, based on previous work on *Recipes* [5], [6].

Runtime Monitoring & Adaptation: There is a plethora of different types of monitors that may operate at different levels; e.g., at the network level there are Network Management Systems (NMS) and Security Information and Event Management (SIEM) [24]. Computational infrastructures include their

own monitors of different types and with varying topologies (e.g., on host systems or virtual machines). Typically, these monitors focus on performance monitoring (e.g., [25]) or monitor incidents and performance at the hypervisor level (e.g., [26]). This segmented landscape makes it difficult to obtain an integrated and holistic run-time picture of IoT applications' security due to the lack of information sharing and correlation among the different types of monitoring data that they produce. System adaptation has also been studied extensively in the context of service-based and agent-based systems, where availability of components, functionality, and operational and context conditions change frequently. Depending on the type of the adaptation actions supported, existing approaches can be distinguished into those which modify the behaviour of the systems (e.g., [27]) and those that alter the compositional structure of the system and/or the partner services involved in it (e.g., [28]). Fewer approaches support the adaptation in ways that can preserve and/or restore security (e.g., [29]). The runtime monitoring and adaptation mechanisms adopted herein are inspired from a more comprehensive approach supporting the runtime adaptation of service compositions in ways that can guarantee desired security properties, as described in [4] and [30]. These also align with the pattern-driven IoT service orchestration pursued by SEMIoTICS, as they leverage secure service orchestration (SSO) patterns, specifying primitive ways of composing services that guarantee the satisfaction of specific security properties (e.g., confidentiality).

V. Conclusions

This paper presented the SEMIoTICS framework's approach towards the development of patterns for orchestration of smart objects and IoT platform enablers in IoT applications with guaranteed security, privacy, dependability and interoperability properties.

Following the finalisation of the framework's architecture, the definition of the pattern language and the development of a first set of SPDI patterns, future work will focus on the development of concrete mechanisms for automated translation from application-level IoT recipes into executable SPDI pattern configurations, the automated reasoning based on said patterns, as well as their monitoring of those at runtime, through the implementation of the corresponding pattern engine components at all layers of the SEMIoTICS framework. Moreover, an extended evaluation path will be pursued, comprising of analytical assessments, followed by simulations of network and applications, as well as final experiments in real-world testbeds in the areas of industrial networks and healthcare.

Acknowledgment

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 780315 (SEMIoTICS).

References

[1] Hatzivasilis, G. et al., "The industrial internet of things as an enabler for a circular economy Hy-LP: a Novel IIoT protocol, evaluated on a wind park's SDN/NFV-enabled 5G industrial network", Computer Communications, 119, 127-137, 2018.

[2] Katsikeas S. et al., "Lightweight & secure industrial IoT communications via the MQ telemetry transport protocol" 2017 IEEE Symposium on Computers and Communications (ISCC), Heraklion, 2017, pp. 1193-1200.

[3] Botta A. et al., "Integration of Cloud computing and Internet of Things: A survey," Futur. Gener. Comput. Syst., vol. 56, pp. 684–700, 2016

[4] Pino L. et al., "Discovering Secure Service Compositions". 4th International Conference on Cloud Computing and Services Sciences (CLOSER 2014), Barcelona, Spain, April 2014

[5] Thuluva, A.S. et al., "Recipes for IoT Applications". In Proc. of the 7th Int. Conference on the Internet of Things (IoT 2017), 22.-25. October 2017, Linz, Austria. ACM.

[6] Seeger, J., R.A. Deshmukh, A. Bröring, "Running Distributed and Dynamic IoT Choreographies". Global Internet of Things Summit (GIoTS 2018), 4.-7. June 2018, Bilbao, Spain. IEEE.

[7] Sakic E. et al., "VirtuWind – An SDN- and NFV-Based Architecture for Softwarized Industrial Networks". In proc. Measurement, Modelling and Evaluation of Computing Systems. MMB 2018. LNCS, vol 10740, Springer, 2018.

[8] Bartoletti M, et al. "Semantics-based design for secure web services." Software Engineering, IEEE Trans. on, 2008.

[9] Deubler M., et al. "Sound development of secure service-based systems." In Proc. of the 2nd Int. Conf. on Service oriented computing. ACM, 2004.

[10] Geor G., et al. "Verification and trade-off analysis of security properties in UML system models." IEEE Trans. on Software Engineering, 36(3): 338-356, 2010.

[11] Dong, J., et al, "Automated verification of security pattern compositions". Inf. Softw. Technol., vol. 52, no. 3, 2010.

[12] Siveroni I., Zisman A., Spanoudakis G.: "A UML-Based Static Verification Framework for Security, Requirements", Engineering Journal, 15(1): 95-118, 2010.

[13] Rossi, S.; "Model checking adaptive multilevel service compositions"; International Workshop of Formal Aspects of Component Software 2010.

[14] Andre R. Souza, et al. "Incorporating Security Requirements into Service Composition: From Modelling to Execution", in ICSOC-ServiceWave '09, 2009.

[15] Pino L., Mahbub K., Spanoudakis G., "Designing Secure Service Workflows in BPEL", International Conference on Service-Oriented Computing, 2014.

[16] Pino L., et al. "Pattern Based Design and Verification of Secure Service Compositions." IEEE Trans. on Services Computing (2017).

[17] Maña A et al., "Extensions to Pattern Formats for Cyber Physical Systems." Proceedings of the 31st Conference on Pattern Languages of Programs (PLoP'14. Monticello, IL, USA. Sept. 2014.

[18] Petroulakis N. et al., "Patterns for the design of secure and dependable software defined networks." Computer Networks 109 (2016): 39-49

[19] Petroulakis N. et al., "Fault Tolerance Using an SDN Pattern Framework", 2017 IEEE Global Communications Conference (GLOBECOM), 2017

[20] Mayer S. et al., "Smart Configuration of Smart Environments". IEEE Trans. on Automation Science and Engineering 13, 3 (2016), 1247–1255.

[21] Lécué F. et al., "SOA4All: An innovative integrated approach to services composition". In Web Services (ICWS), IEEE International Conference on. IEEE, 58–67, 2010.

[22] Ovadia S., "Automate the Internet With If This Then That (IFTTT)". Behavioral & Social Sciences Librarian 33, 4 (2014), 208–211

[23] Ur B. et al., "Trigger-Action Programming in the Wild: An Analysis of 200,000 IFTTT Recipes", In Proc. of 2016 CHI Conference on Human Factors in Computing Systems (CHI '16). 3227–3231, 2016.

[24] Network Monitoring Tools, https://stanford.io/1KZttzT

[25] Nagios, http://www.nagios.org/

[26] Amazon Cloudwatch, http://aws.amazon.com/cloudwatch

[27] Console L. et al., "WS-DIAMOND: an approach to Web Services - DIAgnosability, MONitoring and Diagnosis," in e-Challenges Conf. 2007, The Hague, Oct. 2007.

[28] Ardagna D. et al., "Adaptive Service Composition in Flexible Processes," IEEE Trans. Software Eng., vol. 33, no. 6, pp. 369–384, 2007.

[29] Khan K.M. et al., "Security oriented service composition: A framework". In Proc. of International Conference on Innovations in Information Technology (IIT), pp. 48-53, 2012.

[30] Pino L. et al., "Finding Secure Compositions of Software Services: Towards A Pattern Based Approach", 5th IFIP Int. Conf. on New Technologies, Mobility & Security, Turkey, 2012