SEMIoTICS Architectural Framework: End-to-end Security, Connectivity and Interoperability for Industrial IoT

Nikolaos E. Petroulakis*, Eftychia Lakka*, Ermin Sakic†, Vivek Kulkarni†, Konstantinos Fysarakis‡, Iason Somarakis‡, Jordi Serra§, Luis Sanabria-Russo§, Danilo Pau¶, Mirko Falchetto¶, Domenico Presenza‖, Tobias Marktscheffel**, Kostas Ramantas††, Prodromos-Vasileios Mekikis††, Lukasz Ciechomski‡‡, Karolina Waledzik‡‡
*FORTH, Greece, †Siemens, Germany, ‡Sphynx Technology Solutions AG, Zug, Switzerland, §CTTC, Spain, ¶ST, Italy, ‖Engineering, Italy, **University of Passau, Germany, ††Iquadrat, Spain, ‡‡BlueSoft, Poland

***Abstract*—Next generation networks, as the Internet of Things (IoT), aim to create open and global networks for connecting smart objects, network elements, applications, web services and end-users. Research and industry attempt to integrate this evolving technology and the exponential growth of IoT by overcoming significant hurdles such as dynamicity, scalability, heterogeneity and end-to-end security and privacy. Motivated by the above, SEMIoTICS proposes the development of a pattern-driven framework, built upon existing IoT platforms, to enable and guarantee secure and dependable actuation and semi-autonomic behaviour in IoT/IIoT applications. Hence, in this paper, we describe the design of the SEMIoTICS architecture that addresses the aforementioned challenges. Specifically, the functional components of the proposed architecture are presented including also an overview of the appropriate realization mechanisms. Finally, we map two verticals in the areas of energy and health care and one horizontal in the areas of intelligent sensing use-cases scenarios to the suggested architecture in order to demonstrate its applicability to different IoT enabling platforms, types of smart objects, devices and networks.**

***Index Terms*—Internet of Things (IoT); Software Defined Networking (SDN); Network Function Virtualization (NFV); Design Patterns**

I. Introduction

Global networks like IoT create an enormous potential for new generations of IoT applications, by leveraging synergies arising through the convergence of consumer, business and industrial Internet, and creating open, global networks connecting people, data, and *things*. A series of innovations across the IoT landscape have converged to make IoT products, platforms and devices technically and economically feasible. However, despite these advancements the realization of the IoT potential requires overcoming significant business and technical hurdles such as dynamicity, scalability, Heterogeneity and end to end security and privacy.

All the above challenges give rise to significant complexities, and relate to the implementation and deployment stack of IoT applications. To address them, the overall aim of SEMIoTICS[1] is 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/IIoT applications. The SEMIoTICS framework will support cross-layer intelligent dynamic adaptation, including heterogeneous smart objects, networks and clouds. To address the complexity and scalability needs within horizontal and vertical domains, SEMIoTICS will develop and integrate smart programmable networking and semantic interoperability mechanisms. The above will be validated by industry, using three diverse usage scenarios in the areas of renewable energy and healthcare, and smart sensing and will be offered through an open API. The SEMIoTICS proposed concept is illustrated in Figure 1.

In this work, we present the development of the SEMIoTICS Architectural Framework consisting of a number of different components and technologies able to satisfy the above challenges in the IoT domain. The remainder of this paper is organized as follows. In Section II, an overview of SEMIoTICS Architectural Framework and the components of each is presented. In addition, Section III describes the different SEMIoTICS use cases. Finally, Section IV provides conclusions and future work.

II. SEMIoTICS Architectural Framework

SEMIoTICS vision in delivering smart, secure, scalable, heterogeneous network and data-driven IoT is based on two key features:

- **Pattern-driven approach**: As re-usable solutions to common problems and building blocks to architecture, patterns are leveraged in SEMIoTICS to encode proven dependencies between security, privacy, dependability and interoperability (SPDI) properties of individual smart objects and corresponding properties of orchestrations (composition) involving them. As a result enables verification of correct behavior at run time and at creation.
- **Multi-layered Embedded Intelligence**: Effective adaptation and autonomic behavior at field (edge) and infrastructure (backend) layers will be achieved by bringing the intelligent analysis locally as per each layer to enable semi-autonomous, prompt reaction as well as broadened intelligence at higher levels.

SEMIoTICS Architectural Framework aims to leverage generic architecture components combined in layered structure

[1]https://www.semiotics-project.eu

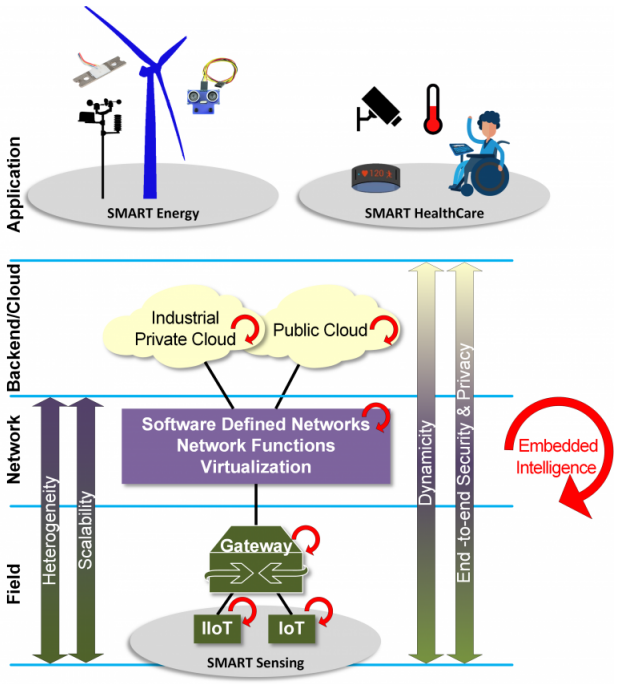

Figure 1. SEMIoTICS

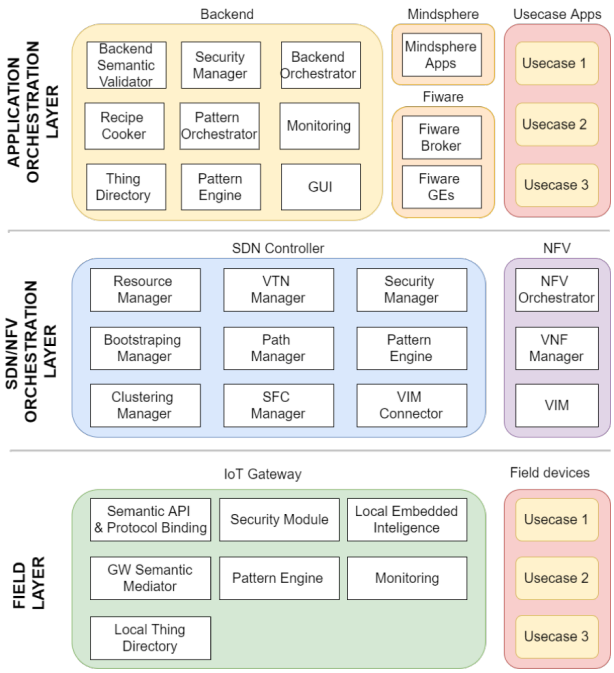

Figure 2. SEMIoTICS Architecture

in order to deliver an Embedded Intelligence at all layers of the framework with the mechanisms empowering SPDI patterns verification across all layers. Created SEMIoTICS pattern-driven framework will be capable of supporting diverse scenarios with specific focus on Smart Energy, Healthcare and Smart Sensing use cases. Figure 2 presents the component logical architecture consisting of three layers: *Application Orchestration Layer*, *SDN/NFV Orchestration Layer* and *Field Layer*; while more details of each layer is presented further.

A. Application Orchestration Layer

Application orchestration layer consist of all applications receiving the communication from field layer. Backend orchestrator will be leveraged for the application orchestration purposes and to provide common functionalities across all deployed applications. Additionally, application orchestration layers hold the use case flows as well as SPDI patter definition.

1) Backend Orchestrator: Backend Orchestrator component is responsible for provisioning all applications and components residing in the backend. As a result a created framework will offer application availability (health checks) and application resource consumption monitoring as well as a set of common API's for: pattern enforcing components, monitoring components and continuous integration (CI) and continuous delivery (CD) tools. Moreover autoscaling features will be provided for applications, ease application/component deployment and one centralized place for backend component management. Approaches taken into consideration include: (i) Kubernetes/Openshift [1] on bare metal, (ii) OpenStack [2] on bare metal, and (iii) Kubernetes/Openshift on OpenStack.

2) Recipe Cooker: Recipe Cooker (RC) is a module able to instantiate recipes. A recipe is a template for a workflow of interactions between multiple ingredients, i.e., devices or services. When a recipe is instantiated, ingredients are replaced with concrete things, described with their own respective Thing Description. Besides the workflow of the recipe, QoS constraints and SPDI patterns can be defined on the interactions. The user of this tool would be typically an IoT application developer. This user wants to focus on the logic of the application flow. Specifically, the user does not have to be an expert in configuring the network and physical connections between the involved IoT devices. The benefit of the recipe approach is that these configurations are automatically done by the tool and the underlying technologies, user only sets SPDI properties (e.g. latency, rate).

3) Thing Directory: The Thing Directory hosts Thing Descriptions of registered things. The Thing Description (TD) model is a recommendation of the W3C Web of Things [3] working group to describe things. The directory features an API to create, read, update and delete (CRUD) a TD. The directory can be used to browse and discover Things based on their TDs.

4) Pattern Orchestrator: The Pattern Orchestrator module features an underlying semantic reasoner able to understand *cooked* recipes, as received from the Recipe Cooker module (see above) and transform them into architectural patterns. The Pattern Orchestrator is then responsible to pass said patterns to the corresponding Pattern Engines (as defined in the Backend, Network and Field layers), selecting for each of them the subset of patterns that refer to components under their control (e.g. passing Network-specific patterns to the Pattern Engine

present in the SDN controller). Through the above functions, the module achieves automated configuration, coordination, and management of the SEMIoTICS patterns across different layers and service orchestrations.

5) Backend Pattern Engine: The Pattern Engine features the pattern engine for the SEMIoTICS framework. Variants of pattern engine can be found at the backend, at the network (SDN controller) and field (IoT gateway) layers. As such, it will enable the capability to insert, modify, execute and retract patterns at design or at runtime in the backend; these interactions will happen through the interfacing with the Pattern Orchestrator (see above), though additional interfaces may be introduced to allow for more flexible deployment and adjustments if needed.

6) Backend Monitoring: Responsible for monitoring, learning and predictive analytics. It receives low-level events as the messages generated by the gateway monitoring component. The reception of a set of low level events can lead to the generation of a new high-level event. Finally, it issues configures the egde monitoring component to properly select and configure the signaling mechanisms as needed.

7) Backend Security Manager: Main security component responsible in the backend of SEMIoTICS' for providing the following services: (1) authentication of users and components, (2) key distribution, (3) management of users, roles, and access rights, and (4) storing and taking decisions on security policies. The Security Manager at the backend layer is the global Policy Decision Point (PDP) for all security policies. In contrast, the Security Managers at SDN and edge level are Policy Enforcement Points (PEP), taking local and enforcing global decisions. For authentication, the Security Manager supports both local authentication and external identity providers using OAuth2.

8) Backend Semantic Validator: The aim of Backend Semantic Validator component (see SEMIoTICS deliverable D4.4) is to tackle the semantic interoperability issues that arise in the SEMIoTICS framework, at the application orchestration layer. The Backend Semantic Validator can receive a request from IoT application for interaction between two Things (i.e. sensor, actuator), which are described with two different TDs (based on W3C Thing Descriptions that are serialized to JSON-LD standard format), respectively.

9) Graphical User Interface: Responsible for the presentation layer, giving meaningful insights into the platform and centralized visualization of the whole framework. Several approaches are to be considered: GUI dedicated to the given application or more generic user interface that embeds view from external application. The other considered approach is GUI that communicates with an external application through the API and presents the content in the standardized way.

B. SDN Orchestration Layer

SDN Orchestration Layer provides data and control plane decoupling resulting in a cloud computing approach that facilitates network management and enables pro grammatically efficient network configuration.

1) VTN Manager: Responsible for assignment of individual network services to various network tenants. It further ensures a separation of L2 traffic in scope of a virtual tenant network.

2) Path Manager: Main network path computation engine of the SDN Controller, responsible for identification of nodes and ports combined into a path that fulfills the pattern requirements (i.e., considers the fault-tolerance or bandwidth/delay constraints).

3) Resource Manager: Provides the Path Manager with a resource view of the network (i.e., the available topology resources such as the port speed, number of queues etc.) exposing the metrics observable using the available interface (e.g., using OpenFlow).

4) SDN Security Manager: The main role of the Security Manager is the support for authentication and accounting services for administration of tenants and assignment of applications with respective tokens used for fast authentication during runtime. Security Manager should accomplish the authentication and accounting services to the rest of the SDN Controller as well as the users and applications that interact with the controller. Moreover, it exposes interfaces for the administration of local SDN Controller accounts, in order to achieve authentication.

5) VIM Connector: In SEMIoTICS, the SDN Controller is considered an external entity to the NFV Management and Orchestration (MANO) framework. This brings benefits in terms of resilience, due to the isolation of network services to separate hosts, but also allows for optimization of both overlay and physical network paths, which could help satisfy SEMIoTICS use cases requirements. Infrastructure flexibility is one of the most relevant features provided by NFV [4], and network overlays play a crucial role in network virtualization.

6) SFC Manager: Service Function Chaining (SFC) Manager used in SFC given the ordering and IP addresses of the nodes that are to be traversed by a tenant's traffic [5]. SFC Manager will handle service function chaining of network functions. It identifies an abstract set of service functions and their ordering constraints that should be applied to packets and/or frames selected as a result of classification. Furthermore, the implied order could not be a linear progression, due to the fact that the architecture allows for nodes which copy to more than one branch; also, the architecture allows for cases where there is flexibility in the order in which services need to be applied.

7) Clustering Manager: A component comprising an underlying registry used in state-keeping of other component's knowledge base, as well as for its strong consistent replication across the SDN controller instances for the purpose of fault-tolerance and high-availability. The design is adaptable to support Byzantine Fault Tolerance [6] depending on the strictness of requirements on dependability in the respective domain.

8) Bootstraping Manager: A component used in initial flow configuration of just-connected switches, so to allow for seamless interaction with IoT devices (e.g., to enable flow rules for propagation of unmatched application packets up

to the controller for the purposes of ARP-based end-device discovery and an automated addition of infrastructural network services).

9) SDN Pattern Engine: Enables the capability to insert, modify, execute and retract patterns at design or at runtime in the SDN controller [7]. PE can be based on a rule engine which will be able to express design patterns as production rules. Enabling reasoning, driven by production rules, appeared to be an efficient way to represent SEMIoTICS patterns. More specifically, since Drools rule engine is based on Maven, it can support the integration of all required dependencies with the ODL controller, as well as the integration of the entities that interact with the controller to run Drools at design and at runtime.

C. NFV Orchestration Layer

NFV provides a flexible, programmable, dynamic and scalable networking paradigm, making it ideal for satisfying the QoS demands of SEMIoTICS use cases.

1) NFV Orchestrator: NFVO allocates (or decommissions) the necessary resources for NS instantiation (or termination). Also, it provides VNF lifecycle management, allows modification of VNF parameters, and modification of VNF interconnection. These are complex endeavours, mainly because VNF's requirements and constrains need to be satisfied simultaneously on top of a very dynamic environment. These tasks are performed by the NS and Resource Orchestration functions (NSO and RO, respectively) inside NFVO. Capabilities of each function are exposed via standard interfaces consumed by other elements of the NFV MANO.

2) VNF Manager: VNF Manager is responsible for the creation and management of the needed virtualized resources for the VNF, as well as the traditional Fault, Configuration, Accounting, Performance, and Security Management (FCAPS). VNF Manager's functions make sure requirements are met at instantiation time. Furthermore, they maintain the virtualized resources that support the VNF functionality without interfering with its logical functions. Like NFVO, VNF Manager functions are exposed through APIs to other MANO functions.

3) VIM: NFVI defines two administrative domains: the Infrastructure and Tenant domains. The former contemplates the physical infrastructure upon which virtualization is performed. The latter makes use of virtualized resources to spawn VNFs and create NSs. A VIM lies in the Infrastructure Domain. It takes care of abstracting the physical resources of the NFVI and making them available as virtual resources for VNFs. It also enables communication with external SDN controllers in order to provide virtual or physical network resources to NSs.

D. Field Layer

Field layer is responsible for hosting all types of IoT devices such as sensors and actuators as well as IoT gateway which provides common way for communication and ensures enforcement of SPDI patterns in this layer. Generic gateway components are capable to work with any set of IoT devices what ensures ability to deliver diverse use cases in various sectors.

1) Semantic API & Protocol Binding: This component exposes the brownfield devices in a common IoT access layer and will be based on W3C Web of Things (WoT) building blocks, i.e., the Thing Description, Binding Templates and the WoT Scripting API. The Thing Description will be used to semantically describe field device resources, their interfaces, security meta-data, etc. Reuse and extention of existing semantics to provide a semantic integration into IoT semantic models is planned. The WoT Scripting API exposes Things (field devices) that have been integrated over Binding Templates and described with Thing Descriptions. It provides uniform standardized access to Things and their data.

2) Security Manager: In the gateway serves as local frontend for the Security Manager at the backend layer. Its two main purposes are: (1) facilitating authentication of sensors and actuators towards SEMIoTICS, and (2) enforcing security policy decisions locally. Sensors and actuators in many cases will be connected to the gateway using low-level protocols and technologies such as MQTT, Bluetooth, etc.; in these cases it simplifies authentication if the gateway contains its own SM.

3) Gateway Semantic Mediator: Gateway Semantic Mediator realizes a common semantic access layer between brownfield devices and new IoT devices. It maps and integrates semantics from existing brownfield devices into IoT or IIoT application semantics. Thus, it enables the discovery of Things representing the legacy applications, so to allow for the definition of their interactions in the Recipe Cooker. To realize the semantic mediation of legacy devices, the mediator must provide a mapping knowledge (*Knowledge Packs*) to map particular brownfield semantic standard into another IIoT standard (e.g. W3C Thing Description format serialized to JSON-LD [3]). SEMIoTICS IoT Gateway is capable of installing and extending the current mappings with new modular Knowledge Packs and thus enable integration of arbitrary field devices into a harmonized IoT access layer. The semantically enriched models are then made accessible over a local and backend semantic repository.

4) Gateway Monitoring: The SEMIoTICS monitoring module in the gateway provides: (1) generation of specific messages in response to the reception of a set of messages generated by IoT application components and matching some condition specified by a client application (monitoring requirement); (2) guarantee that the messages needed to decide whether to generate a message can be produced by an IoT application and received by the monitoring component (observability property). Monitoring module will allow the execution of some of the monitoring tasks close to the field devices and it's aggregation of the low level events. Delivery only aggregated results to backend monitoring module reduces number of transmissions and saves resources (i.e. energy, bandwidth) which are scarce in edge nodes, e.g. a mobile phone.

5) Gateway Pattern Engine: Pattern Engine in the gateway is able to host design patterns as per Pattern Orchestration located in the backend layer. Due to gateway capability limitations, Pattern Engine will host patterns in an executable

form compared to the pattern rules as provided in the other layers. Hence it will guarantee SPDI properties locally based on the data retrieved and processed by the Monitoring module, the Thing Directory in the IoT gateway and based on the interaction with other components in the field layer. Finally, the Pattern Engine in gateway will store the executable patterns locally and will be updated by the pattern orchestrator on request.

6) Local Embedded Intelligence: Local Embedded Intelligence is component able to (i) execute a use case specific application logic (ii) rely on at least one of the services provided by the SEMIoTICS platform and (iii) deployed on a field device. The Controller on board of the Robotic Rollator part of the SARA UC is an example of Local Embedded Intelligence since: (i) it addresses a requirements specific of the UC (i.e. to power the hub wheels in order to balance the user's weight) (ii) relies on the Semantic Mediator to discover how to address the hub wheels available on the specific rollator (iii) is deployed on the Single Board Computer (SBC) (i.e. a Raspberry Pi 3) of a Robotic Rollator.

7) Local Thing Directory: The local repository of knowledge containing necessary Thing models. The purpose is to store semantic description of Things locally in the Generic IoT Gateway. Semantic Mediator will allow providing these descriptions in accordance to W3C WoT standard and iot.schema.org in order to verify if specific Thing shall be used for a new Edge application development. However, not only humans but software components or machines may query Local Thing Directory too, e.g., when automatically generating a user interface for a Thing or matching Recipe requirements with capabilities of Things. The component keeps all semantic meta-data up to date (device capabilities, configuration parameters of devices, contextual information (e.g., location, feature of interest etc.) and will be synchronized with the Thing Directory running in the Backend.

E. External IoT Platforms

SEMIoTICS will interface third party IoT platforms, most notably FIWARE. In FIWARE, the Orion Context Broker, which keeps track of the sensor and actuator Context Information, will be be federated with the SEMIoTICS architecture. FIWARE leverages the NGSIv2 Data Model and API, which relies on JSON representation to make data from multiple providers accessible for data consumers. The interaction with both data providers and data consumers is taking place via the FIWARE NGSI 10 context data API. SEMIoTICS will leverage this API for context queries, context subscription and context updates to interact with the respective context elements (i.e., sensors and actuators) in the FIWARE domain and add them at the local SEMIoTICS Thing Directory.

III. SEMIoTICS Use Cases

This section showcases the demonstration scenarios (use cases) to be presented within SEMIoTICS framework. Each use case is described from the perspective of the generic

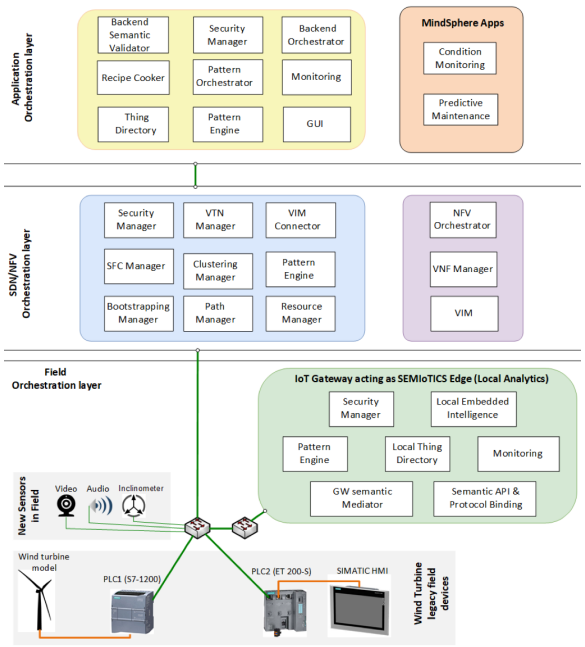

Figure 3. Use Case 1 in SEMIoTICS architecture

SEMIoTICS architecture with special focus on showing specificity of each use case and how dedicated components are leveraged in order to be follow generic architectural guidelines.

A. Use Case 1 - Wind Energy

State of the art wind turbine controllers in wind park control domain are embedded, highly integrated control systems, requiring rigorous development and pre-qualification prior to deployment. Adding new sensors, actuators and associated features may require years of testing and integration before introduction in field operations. Use Case 1 (3 portrays how SEMIoTICS IIoT integration can benefit the wind park control domain. Currently, O& M personnel in a remote control center needs to monitor the inclination of the steel towers on a number of specific wind farms to prevent the deformation and fatigue of the steel. SEMIoTICS will apply localized edge analytics: only the data container containing the algorithm and result of the inclination calculation is transferred between the wind turbine and the remote control center; measurements can be done more frequently than currently. Thus, unnecessary traffic between turbines and remote control center is reduced and abnormalities are detected more quickly.

B. Use Case 2 - e-Health

This use case employs the SEMIoTICS technologies to develop a solution aimed at sustained independence and preserved quality of life for elders with Mild Cognitive Impairment or mild Alzheimer's disease, with the overall goal of delaying institutionalization: supporting both 'aging in place' (individuals remain in the home of choice as long as possible) and 'community care' (long-term care within the community rather than in hospitals or institutions). The SARA UC design