Dynamic Reconfiguration of Smart Sensors: A Semantic Web based Approach

Samya Sagar, Maha Khemaja, Maxime Lefrançois, and Issam Rebai

***Abstract*—This paper aims to tackle issues related to the (re-)configuration of Smart Objects. These latter are composed specifically of smart sensors that embed basic sensors, a micro-controller and software snippets as well. In order to make this (re-)configuration feasible at runtime, we propose a semantic Web based approach that relies on a set of ontology modules together with a set of logical rules and reasoning processes to drive the (re-)configuration mechanism while taking into account, at the same time, requirements of the application domain. We validate our approach by means of a prototype that shows relevance of developed ontologies and (re-)configuration mechanism.**

***Index Terms*—Ontology modeling, smart sensor, Reconfiguration, Domain Modeling**

I. Introduction

Nowadays, the Internet of Things integrates sensor networks to the Internet continuously and opens up new opportunities to enable the development of systems or ecosystems mixing virtual and physical worlds, in order to assist people during their everyday lives. The IoT allows objects to be omnipresent and to interact with each other, to cooperate with their neighbors in order to answer to a common goal. These objects are coined Smart Objects, as they are everyday objects which, in addition, integrate smart sensors and provide a set of interesting services to their users. A watch, a car, a cloth are such kinds of Smart Objects.

The number of these Smart Objects has increased considerably, thus inducing the emergence of many research works. These tackle different aspects and target many different but related visions: the Object Oriented Vision (Things-oriented), the Internet Oriented vision (Internet-Oriented) and the Semantic Oriented vision (Semantic-Oriented) [1]. This latter vision attempts to address issues related to semantic heterogeneity and interoperability of resources (mainly data) that are exchanged between Smart Objects. It also addresses the issue related to the smartness of connected objects (i.e. where this smartness is embedded and how).

In the present paper we are concerned with the semantic vision. More specifically, we aim to tackle issues related to semantic modeling and representation of Smart Objects as well as the underlying semantic mechanism that we propose in order to enable (re-)configuration of those objects at runtime. Hence, we propose in this paper, a semantic Web based approach that enables to deal semantically with the (re-)configuration of the embedded behavior (software snippets) of Smart Objects. The (re-)configuration mechanism consists of updating in an automatic and intelligent manner the configuration of the considered objects. It specifically makes decisions about the software snippets to install and the other ones to uninstall in order to answer accurately and on time to the domain's requirements (the right sensed and/or computed or derived data at the right time with the right accuracy). The software snippets represent and implement the logical processing and behavior of Smart Objects. They are stored, retrieved and installed from specific repositories that are created, enriched and shared between different development teams and stakeholders. The semantic (re-)configuration mechanism is driven by the Smart Objects characteristics and their context of use. The approach we propose relies on a set of ontological modules that compose the SMartSensing Ontology (SMS ontology). The main contributions of this paper are therefore twofold: (1) the SMS modules and (2) the semantic mechanism for (re-)configuration of smart sensors.

The reminder of the paper is organized as follows: Section II presents the research context as well as a motivating scenario that illustrates (re-)configuration requirements for Smart Objects or smart sensors. Section III presents and analyzes existing (re-)configuration solutions. It also allows us to highlight needs for a novel (re-)configuration approach which is semantic based. Section IV describes the SMS ontology which is a set of ontological modules describing smart sensing capabilities and Smart Objects characteristics and on which relies the (re-)configuration mechanism we propose. Section V details this mechanism together with the set of logical rules expressing conditions that drive it. Section VI presents a prototype for validation and experimentation showing advantages and efficiency of the SMS ontology and the developed reconfiguration mechanism. Section VII draws conclusions of the present work and gives insights for future work.

II. Motivation and context Description

This section highlights the research context related to Smart Objects adaptation and specifically the (re-)configuration of smart sensors. Moreover, it describes a motivating scenario for (re-)configuration of a smart cloth that is used for sports. Smart clothes are Smart Objects that enable collecting physiological data about their wearer.

Samya SAGAR, Issam Rebai are with IMT Atlantique, Lab-STICC, Bretagne Loire, F-29238 Brest, France (e-mail: samya.sagar@imt-atlantique.fr). (e-mail: issam.rebai@imt-atlantique.fr). (Corresponding author: Samya SAGAR)

Maha Khemaja is with Isitc, Prince Research Group, University of Sousse Sousse, Tunisia (e-mail: maha_khemaja@yahoo.fr).

Maxime Lefrançois is with Univ Lyon, Univ Jean Monnet, Mines Saint-Etienne, IOGS, CNRS, UMR 5516 LHC, Institut Henri Fayol, F.-42023 Saint-Etienne France (e-mail: maxime.lefrancois@emse.fr).

A. Context

Adapting a Smart Object consists of (re-)configuring its electronic components, namely the integrated sensors. These sensors have the capability to process data thanks to software snippets that are embedded and deployed inside them. This characteristic is what we call *smart* in the context of this paper. Software components can be installed or uninstalled depending on the domain and application requirements as well as the usage context. Every smart sensor is composed of a processing unit that is used for local data processing. A Smart Object integrates a set of smart sensors. Unlike basic sensors, smart sensors provide many functions (or services). They are designed in order to serve different domains and contexts. According to the work in [2] "The concept of a smart sensor is based on its ability to (1) acquire data thanks to its embedded sensors, (2) process this data thanks to one or more algorithms its micro-controller implements, (3) output and communicate indicator values, (4) be reprogrammed and reconfigured". One can therefore adapt the smart sensor depending on users' needs by changing the embedded software components (or snippets).

B. Scenario

In order to illustrate the manner a smart sensor functions, we introduce the following motivating scenario. This scenario describes technical details related to the use of an example of a smart sensor that hosts a three-axis accelerometer of type LIS2DH that measures acceleration, and a microcontroller.

In this scenario we consider Abdel, a sportsman that practices cycling and running. Abdel bought a sportswear shorts equipped with this smart sensor. He wants to monitor his activity using a Bluetooth Low Energy connection to his Smart Watch. During a race, he monitors the stride number, an estimation of the running distance and his average stride frequency. When he practices cycling, he monitors the pedaling cadence, the "riding out of the saddle" (or dancing) duration and the road steepness. These six indicators can be computed from the same raw data given by the LIS2DH embedded in the smart sensor. What make the difference are the computer programs implemented and executed by the micro-controller. When Abdel selects the activity he wants to practice, the smart sensor modifies its operation by loading the relevant algorithms. Now, if Abdel wants to start roller-skating, he may browse the web for an algorithm he can load on his smart sensor to get an estimation of his skating rate. That algorithm can be installed on the smart sensor only if: (1) the inputs of the algorithm can be provided by the LIS2DH, and (2) the capabilities of the smart sensor's micro-controller are sufficient for executing this algorithm.

Through the analysis of this scenario, we can infer details about Smart Objects structures and specifically the smart sensors they host. The scenario illustrates how a Smart Object (i.e. the sportswear shorts) could be reconfigured to different sports by downloading over the Web software components or programs that are allowed for public usage. Smart Objects' (re-)configuration requires a set of software components that may be already installed in the smart sensor or ready to be downloaded and installed. In this case, the hardware components remain unchanged. The (re-)configuration is possible if the Smart Object has the capability to reconfigure its smart sensors, change its communication parameters, and change the code that is executed by its micro-controller, to satisfy the new configuration.

III. Related work

(Re-)configuration (or adaptation) implies loading and unloading either statically or dynamically software components or setting up the smart sensors' parameters in order to obtain the required performance. In general, (re-)configuration is related to the smart sensor adaptation by adding new functions (or services) or deleting stale ones. It represents a challenging and difficult task either for software developers or for the smart sensor designers. Indeed, smart sensors are characterized by their constrained and limited resources that very often hamper their (re-)configuration. Therefore, the pending issue is how to trigger the (re-)configuration of the software embedded inside the smart sensor while taking into account its limited constraints and the application or domain requirements.

A literature review and analysis regarding (re-)configuration mechanisms reveals that the main drawbacks of existing solutions are due to the constrained resources of sensors. Component based solutions seem to be the most efficient regarding resource consumption. Indeed, this modular (re-)configuration approach is by far the most efficient and is flexible enough to accommodate for new requirements, while accounting for the severe resource constraints imposed during (re-)configuration. Among those solutions, the FiGaRo [3] approach offers a dynamic (re-)configuration approach for distributed software components in a sensor network. Think [4] is an implementation of the Fractal component model [5] which takes into account the specific constraints of embedded systems, including sensor networks, and provides a fine (re-)configuration of software components. OpenCom [6] provides a platform for (re-)configuring software components at runtime. It is based on a generic component model for creating system applications that are independent of the platform environment. Another component-based distributed (re-)configuration approach is the WiSeKit middleware proposed in [7] and [8]. This middleware provides an abstraction layer for developing applications on the sensor network . The code is developed according to the requirements of adaptability at the application level. REMOWARE [9], RUNES [10], FlexCup [11] and LooCI [12], focused on (re-)configurations of a single sensor.

The (re-)configuration in most of these mechanisms is at the middleware level, which makes any adaptation of the mechanism cumbersome and difficult. The (re-)configuration mechanism also focuses on (re-)configuration related to the physical characteristics of the components, does not take into account their treatments or equivalent treatments, and neglects the semantics of their processing logic. Although these mechanisms are inspired by the literature developed in the IoT domain, we adopt in our approach a higher-level (re-)configuration decision-making. It is also done in a declarative manner something that makes its modification quite easy

TABLE I
EVALUATION OF THE (RE-)CONFIGURATION MECHANISMS

Solutions	Reprogramming paradigm	Runtime plateform	Reconfiguration level	Distributed/local re-configuration	reconfiguration approach
FiGaRo [3]	Components based	Contik	Application Layer	Distributed	Based on programming models
LooCI [12]	Components based	SunSPOT	Services	Local	Based on interaction models
FlexCup [11]	Components based	TinyOS	Application Layer	Local	Based on meta-data
OpenCom [6]	Components based	Indep.	Middleware	Distributed	Based on a components graph
Think [4]	Components based	Indep.	All layers	Distributed	Based on meta-models + components models
RUNES [10]	Components based	Contiki	Middleware	Local	Not specified
WiSeKit [7]	Components based	Indep.	Middleware	Distributed	Based on services API
REMOWARE [9]	Components based	Indep.	Middleware	Local	Based on services API

(adaptation of the (re-)configuration mechanism itself). Table I summarizes the main criteria of these solutions.

IV. PROPOSED APPROACH

A. Modeling requirements

Before exploiting or re-configuring a Smart Object, it is compulsory to specify both its structure and its behavior. We define a Smart Object as a real-world physical object equipped with sensors or actuators that transcend its original use to offer new services and new features. A Smart Object is equipped with electronic components to communicate and exchange data with other physical or digital entities on a local network or on the internet. A Smart Object is therefore described by its components (or its structure) and its behavior (the services it offers or its reactions to external stimuli).

A Smart Object may be represented or may have a conceptual structure composed of three elements: the primary object component, the application domain component and the different electronic components. This latter contains concepts that describe the hardware and software components integrated inside the primary object or that augment it. These concepts include the ones describing sensors, actuators, sensor networks, communication systems and protocols. They also include concepts related to the different possible treatments (processing logic) qualifying the behavior of these electronic components. However, hardware components and software components can be separated into two broad categories.

We identify four entities that guide the modeling of a Smart Object. These entities are the Primary Object, the Electronic Components, the Processing Logic and the Application Domain as shown in Figure 1. The latter can in turn be broken down into several sub-domains that will guide the design of several levels of abstraction. The circles show the different entities that influence the description of the Smart Object. At the center of the model, and at the intersection of the circles, lies the Smart Object (for example, Smart Clothing). The modeling of a Smart Object results from the modeling of all the entities that make it up.

- The Primary Object (for example, Clothing) contains generic concepts for the description of the main component of an object as a simple object (for example, a garment). It includes concepts used to specify the characteristics of primary components (for example, fabric), shape, color, etc. which are necessary for a complete description;

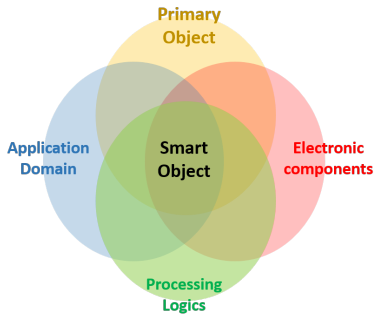

Fig. 1. Representation of Smart Object

- The Application Domain defines the requirements of the domain for which the Object will be manufactured and used. It groups together all the concepts related to the application domain and the context of use including the concepts related to the social context of the application domain (i.e. the users, their profiles, their needs, their preferences, etc.);
- The Electronic Components are distinguished by the nature of the material and features they offer that are included in the object. An electronic component is composed by other electronic components having different levels of granularity (sensors, micro-controller, etc.);
- The Processing logics are processing software chunks (or snippets) embedded inside the micro-controller of a smart sensor. This allows modeling and implementing the identified functions in response to functional requirements.

This analysis step related to the extraction and modeling of requirements of all the entities structuring a Smart Object and any other specific object (for example, a smart garment), makes it possible to infer a set of semantic models which is important to specify, define and implement. This set concerns a decomposition of the initial models into more specific and fine grained modules. In particular, the generic semantic modeling component of a Smart Garment, for example, is based on the previous analysis, mainly on the ontology modules of the SMartSensing ontology (SMS ontology), which we propose, and where generic and reusable modules are distinct from those that are more specific.

B. Overall structure of the SMS ontology

The SMS ontology is generic, modular and can be reused in many different cases related to semantic modeling of

Smart Objects. It defines the scope and purposes of the different modules that are necessary for the usage and (re-)configuration of a Smart Object. As shown in Figure 2, the SMS ontology is logically decomposed into three main modules according to the structural components of a Smart Object, namely: (1) the ontological module of primary object; (2) the ontological module of the application domain; and (3) the ontological module of electronic components. The last module has a higher level of abstraction than the other ones. It is therefore independent from the application domain and the primary object. It can be reused for all Smart Objects and describes the concepts related to smart sensors and the integrated processing logic. The electronic component ontology called the Semantic Smart Sensor Network (S3N) wich was presented in [2] defines the generic and main module of the SMS ontology. The S3N ontology is composed of a set of ontological modules, each of which describe a particular aspect of a hardware or software component . This ontology consists of six modules: S3N-Algorithm; S3N-Core; S3N-Datasheet; S3N-System; S3N-Procedure and S3N-Thing.

Ontological modules for the application domain and the primary object are both very specific. Therefore they are only reusable for similar cases. Consequently, they are either imported or developed as part of a specialization of the SMS ontology. The SMS ontology was developed as part of the SmartSensing project for Smart Clothing dedicated to sports [13]. For the purposes of this project we have developed a sport ontology and an ontology of measurements and indicators.

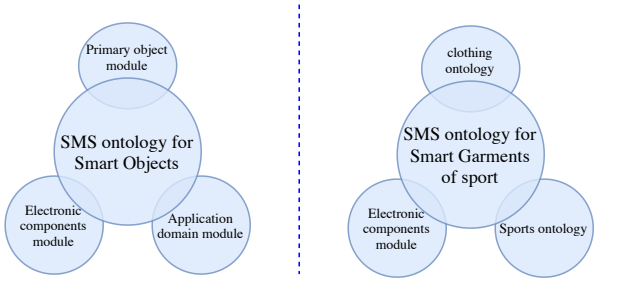

Fig. 2. Model of the SMS ontology

These modules are described in the next subsection. The interaction between all these ontological modules, as well as the reused reference ontologies (i.e. SOSA/SSN [14], the Dolce Ultralite (DUL) upper ontology [15] and Thing Description (TD) ontology [16]), is described by the global structure, illustrated in Figure 3. This structure builds up links and semantic relationships between the different modules. These relationships were defined by the global design of a smart garment, taking into account the scenarios of its exploitation and its (re-)configuration.

C. Conceptualization and development of the SMS ontology for Smart Garments dedicated to Sports

The use and (re-)configuration of a Smart Garment dedicated to a given sport requires knowledge about the chosen sport's practices and activities, the indicators to be calculated, and hence the smart sensors to configure. In this section, we discuss the different ontological modules used during the exploitation and (re-)configuration phases of a Smart Garment. We first describe the modules of the generic S3N ontology, then we describe the two specific modules for the sport application domain.

1) The generic S3N ontology modules: The goal of the present work is to provide a semantic representation that allows to understand and interpret the operations realized by a smart sensor. In this section, we design and implement the generic part of the SMS ontology, namely the S3N (Semantic Smart Sensor Network) module. S3N is a modular ontology, it describes the hardware components and the processing logic of a smart sensor. The S3N modules can be used (or imported) either separately or together, and they all define terms in the same namespace `http://w3id.org/s3n/`. They are published in conformance with the best practices for publication and metadata using the SEAS innovative publication system [17]. In the rest of the paper, we use the shortened form of the namespace which is `s3n:`.

```
@prefix s3n: <http://w3id.org/s3n/>.
```

a) The S3N-Algorithm module: The processing logic integrated into the objects to make them smart and adaptable according to the requirements of their users is specified via a set of algorithms. An algorithm is a logical process (a sequence of operations) used to solve a certain problem. A specification of an algorithm defines all of its operations as well as predicates that describe the initial and final states (s3n:PreCondition and s3n:PostCondition) of its execution.

Thus, the S3N-Algorithm module has been developed to define the processing logic to be deployed inside the electronic components and more precisely within the smart sensors. The S3N-Algorithm module was created as a specialization of the DUL upper ontology. It is represented in Figure 4 and is identified by `http://w3id.org/s3n/S3NAlgorithm`. It describes the flow of data and the dependencies between data.

S3N-Algorithm allows the description of the algorithms in terms of sets of operations. An s3n:Algorithm is linked to its operations by the s3n:hasOperation relationship. An algorithm is composed of at least one operation. An operation can be reused by several algorithms.

$$\text{s3n:Algorithm} \sqsubseteq \forall \text{s3n:hasOperation}.\text{s3n:Operation}$$

$$\text{s3n:Algorithm} \sqsubseteq \geq 1.\text{s3n:hasOperation}.\text{s3n:Operation}$$

Since an algorithm is a sequence of well-defined operations, each operation (s3n:Operation) has a given position during the algorithm execution. It is therefore assigned a sequence number of its order of appearance in the execution process.

$$\text{s3n:SequenceNumber} \sqsubseteq \forall \text{s3n:ofOperation}.\text{s3n:Operation}$$

$$\text{s3n:SequenceNumber} \sqsubseteq \forall \text{s3n:forAlgorithm}.\text{s3n:Algorithm}$$

$$\text{s3n:SequenceNumber} \sqsubseteq = 1.\text{s3n:forAlgorithm}.\text{s3n:Algorithm}$$

$$\text{s3n:forAlgorithm} \circ \text{s3n:hasOperation} \sqsubseteq \text{s3n:ofOperation}$$

The s3n:ofOperation and s3n:forAlgorithm properties are defined as follows:

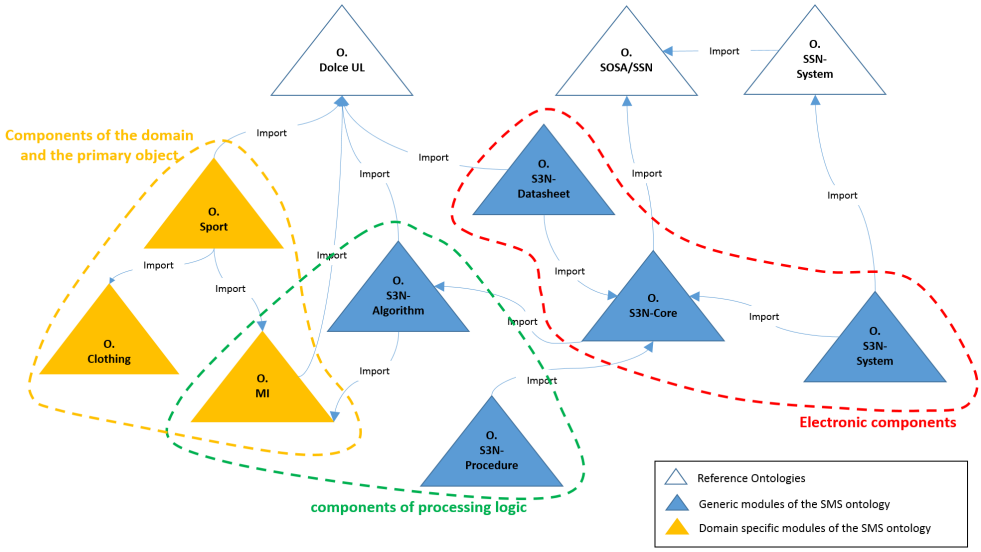

Fig. 3. Dependency graph between modules of the SMS ontology

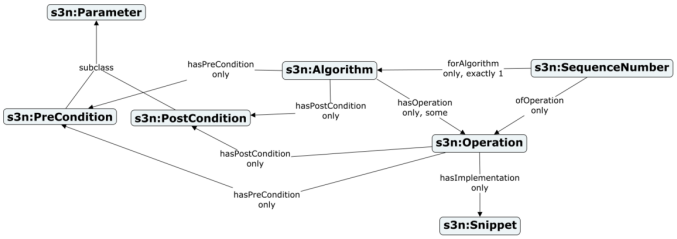

Fig. 4. S3N-Algorithm Module

```
s3n:ofOperation
  schema:domainIncludes s3n:SequenceNumber ;
  schema:rangeIncludes s3n:Operation .
s3n:forAlgorithm
  schema:domainIncludes s3n:SequenceNumber ;
  schema:rangeIncludes s3n:Algorithm .
```

An operation can be implemented by several source codes, which are program fragments and represented by the concept s3n:Snippet.

$$\text{s3n:Operation} \sqsubseteq \forall \text{s3n:hasImplementation}.\text{s3n:Snippet}$$

b) The S3N-Core module: By considering the definition of a smart sensor we have conceptualized the S3N-Core module. This module describes smart sensors in terms of their physical structure as well as their functional adaptability. The S3N-Core module imports and extends the SOSA/SSN ontology and S3N-Algorithm module. It is identified by `http://w3id.org/s3n/S3NCore`. This module is illustrated in Figure 5.

The specification of S3N-Core module classes and properties has been established from the perspectives of (1) components, (2) algorithms and their execution, (3) features of interest and properties, (4) results.

Components of a Smart Sensor: The description of a sensor in SOSA/SSN is defined by the concept sosa:Sensor as a subclass of the ssn:System concept. This sensor representation given by SOSA/SSN meets the requirements of our approach to describe basic sensors. In the SOSA/SSN ontology, an ssn:System is composed of several subsystems. A smart sensor is an assembly of several electronic components, in particular a set of sosa:Sensor, a s3n:MicroController and a s3n:CommunicatingSystem. To represent this specificity, S3N-Core extends SOSA/SSN with three classes:

- s3n:MicroController: A MicroController is a compact integrated circuit containing a processor, a memory, and input/output (I/O) peripherals on a single chip, and is designed to govern a specific operation in an embedded system. It implements and runs procedures.
- s3n:CommunicatingSystem: A Communicating System can be used to exchange information with other s3n:CommunicatingSystem on some network.
- s3n:SmartSensor: A SmartSensor is composed of one or more Sensors together with a s3n:MicroController that implements different Procedures, and make Executions of these Procedures on the result of the Observations these Sensors make to output a resulting value for some Indicator. This value may then be communicated by some s3n:CommunicatingSystem.

A smart sensor hosts different components, and can therefore be considered as a sosa:Platform. Copying the SOSA axiomatic scheme, S3N asserts that the domain of sosa:hosts also includes s3n:SmartSensor, while its range also includes s3n:MicroController and s3n:CommunicatingSystem.

```
sosa:hosts
 schema:domainIncludes s3n:SmartSensor ;
 schema:rangeIncludes s3n:Microcontroller ;
 schema:rangeIncludes s3n:CommunicatingSystem;
 schema:rangeIncludes s3n:SmartSensor .
```

Then taking advantage of the richer axiomatization of SSN, s3n:SmartSensor is defined as a sub-class of both sosa:Platform (*an entity that hosts other entities*) and ssn:System (*unit of abstraction for pieces of infrastructure that implement Procedures*), while both s3n:MicroController and s3n:CommunicatingSystem

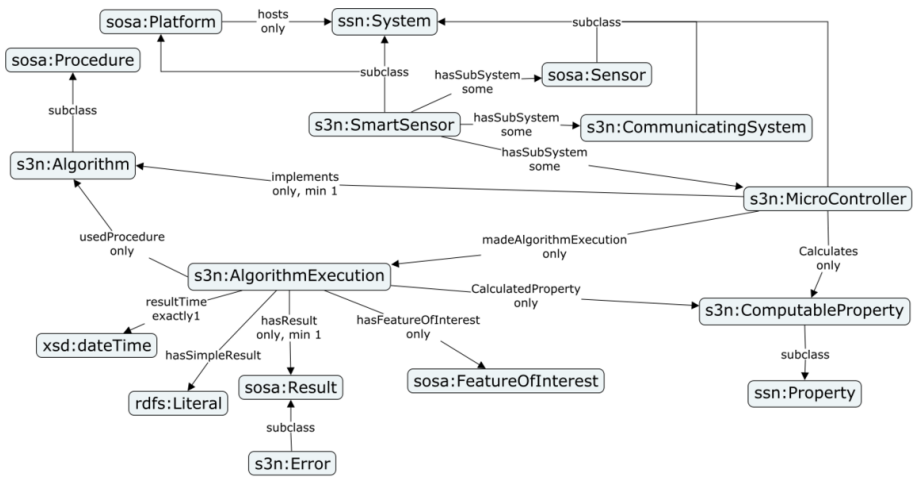

Fig. 5. S3N-Core Module

are defined as sub-classes of ssn:System. Finally, we model the fact that a s3n:SmartSensor contains at least one of each components.

$$\text{s3n:SmartSensor} \sqsubseteq \exists \text{ssn:hasSubSystem}.\text{sosa:Sensor}$$

$$\text{s3n:SmartSensor} \sqsubseteq \exists \text{ssn:hasSubSystem}.\text{s3n:MicroController}$$

$$\text{s3n:SmartSensor} \sqsubseteq \exists \text{ssn:hasSubSystem}.\text{s3n:CommunicatingSystem}$$

Algorithms and their executions: The S3N-Core module of the SMS ontology also describes the functioning mode of a smart sensor. SOSA/SSN follows similar design patterns for Sensors (that implement Procedures and make Observations), Actuators (that implement Procedures and make Actuations), and Samplers (that implement Procedures and make Samplings). As we introduce new sub-types of ssn:System, it is justified to reuse this pattern for s3n:MicroControllers . Unlike for Sensors and Actuators that may implement different kinds of procedures , s3n:MicroControllers are specifically designed to implement sensing-related algorithms as well as adaptation and (re-)configuration ones. We thus specialize sosa:Procedure and propose the following pattern instantiation: s3n:MicroControllers implement some s3n:Algorithm and make s3n:AlgorithmExecution activities (property s3n:madeAlgorithmExecution). Parallel to SSN, the S3N ontology specifies that a s3n:MicroController only makes algorithm execution, implements only and at least one Algorithm. The axiomatization of this pattern is:

$$\text{s3n:Algorithm} \sqsubseteq \text{sosa:Procedure}$$

$$\text{s3n:MicroController} \sqsubseteq \forall \text{s3n:madeAlgorithmExecution}.\text{s3n:AlgorithmExecution}$$

$$\text{s3n:MicroController} \sqsubseteq \geq 1.\text{ssn:implements}$$

$$\text{s3n:MicroController} \sqsubseteq \forall \text{ssn:implements}.\text{s3n:Algorithm}$$

Property s3n:madeAlgorithmExecution is defined as follows:

```
s3n:madeAlgorithmExecution
 schema:domainIncludes s3n:MicroController ;
 schema:rangeIncludes s3n:AlgorithmExecution.
```

The link between an s3n:AlgorithmExecution and the specific algorithm it used can be made explicit using sosa:usedProcedure.

$$\text{s3n:AlgorithmExecution} \sqsubseteq \forall \text{sosa:usedProcedure}.\text{s3n:Algorithm}$$

Features of interest and properties: Properties that are observable (resp. actuatable) by basic sensors (resp. actuators) are represented in SOSA/SSN by the concept sosa:ObservableProperty (resp. sosa:ActuatableProperty). Additionally, other properties are related to the features of interest, such as the number of steps computed during Abdel's race in the Scenario. We define another sub-class of ssn:Property: s3n:ComputableProperty. This class allows the description of properties that are computable by the micro-controller.

The execution of an algorithm by the micro-controller computes a property of a sosa:FeatureOfInterest. An algorithm execution links to the micro-controller that made it and to the procedure that was used. An algorithm execution is linked to the computed property of a feature of interest through the property s3n:calculatedProperty. This property is defined as follows:

$$\text{s3n:AlgorithmExecution} \sqsubseteq \forall \text{sosa:hasFeatureOfInterest}.\text{sosa:FeatureOfInterest}$$

$$\text{s3n:AlgorithmExecution} \sqsubseteq = 1.\text{sosa:hasFeatureOfInterest}.\text{sosa:FeatureOfInterest}$$

$$\text{s3n:MicroController} \sqsubseteq \forall \text{s3n:computes}.\text{s3n:ComputableProperty}$$

$$\text{s3n:ComputableProperty} \sqsubseteq \text{ssn:Property}$$

Results: The result of an s3n:AlgorithmExecution may be a literal (using sosa:hasSimpleResult) or an instance of sosa:Result (using sosa:hasResult), in which case it may additionally be an instance of s3n:Error and potentially have a s3n:hasCause.

c) The S3N-System module: This module describes the operating capabilities related to components deployed on a smart sensor. The SOSA/SSN ontology specifies a set of capabilities in the SSN-System module. The S3N-System module imports SSN-System and adds new concepts for the capabilities related to integrated components in a smart sensor, namely the micro-controller and the communication system: (1) s3n:Memory for s3n:MicroController(s): the memory of the micro-controller under the specified conditions; and (2) s3n:MaximumBandwidth for s3n:CommunicatingSystem(s): the maximum bandwidth of communicating equipment under specified conditions. The S3N-System module has URL http://w3id.org/s3n/S3NSystem. Figure 6 illustrates this module.

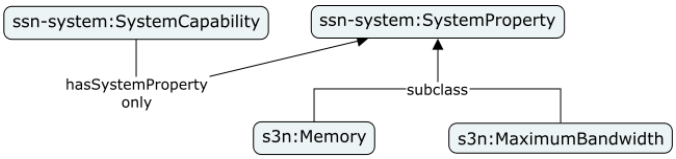

Fig. 6. The S3N-System module

d) The S3N-Procedure module: This module defines the different characteristics of processing logic that can be implemented on a smart sensor, and specifically on its microcontroller. The S3N-Procedure module has the URL http://w3id.org/s3n/S3NProcedure. It is illustrated in Figure 7. It extends SOSA/SSN by adding a module to describe the characteristics of the procedures implemented by the systems. The design of the S3N-Procedure module is strongly inspired by the SSN-System and S3N-System modules. The S3N-Procedure module is designed to describe the properties of procedures such as duration, computational cost, storage cost, and so on, under specified conditions such as the input size. This extension uses the terms s3n:ProcedureFeature, s3n:hasProcedureFeature, s3n:hasProcedureProperty, and s3n:ProcedureProperty. To define a relationship between an entity and one of its properties, the SSN ontology defines the property ssn:hasProperty (and its inverse ssn:isPropertyOf).

$$\text{sosa:Procedure} \sqsubseteq \forall \text{s3n:hasProcedureFeature}.\text{s3n:ProcedureFeature}$$

$$\text{s3n:ProcedureFeature} \sqsubseteq \forall \text{s3n:hasProcedureProperty}.\text{s3n:ProcedureProperty}$$

The class s3n:ProcedureFeature describes normal characteristics of a process under certain conditions such as a calculation exception. Examples of normal characteristics include complexity or calculation cost. The s3n:ProcedureProperty class describes an identifiable and observable property that represents the characteristics of a process to perform an expected result. We reuse the property ssn:forProperty to link a feature of a procedure to the property for which the feature has been described. We then add procedure properties as subclasses of s3n:ProcedureProperty. The s3n:ComputationalCost procedure property is defined for any procedure, and the s3n:TimeComplexity and s3n:SpaceComplexity properties are specifically defined for s3n:Algorithm(s). These properties can be used to model the fact that an algorithm may have different capabilities depending on the activity for which it is used, using s3n:inProcedureCondition.

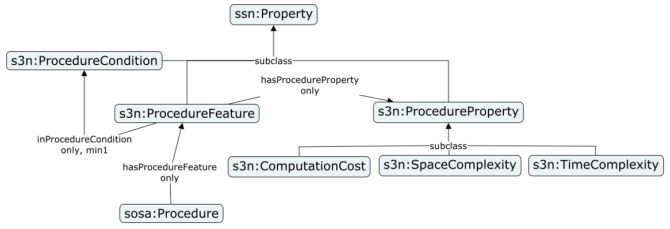

Fig. 7. S3N-Procedure Module

2) Specific modules of the SMS ontology: The specific modules describe concepts related to the primary object and the application domain. In the SmartSensing project the primary object is a garment and the domain is sports. To model garments, we reuse the VetiVoc textile modular ontology [18], which provides a rich vocabulary for clothing. To model the sport domain and the associated performance indicators, we developed two modules: the Sport module and the Measurement and Indicator module.

a) The Sport module: This module specifies the knowledge related to sports activities. It describes sports and classifies them (sport:CollectiveSport or sport:IndividualSport). It links each sport to its practices and indicators. It offers a model for referencing sports and describing the constraints associated with them. Figure 8, shows the conceptual model of the Sport module.

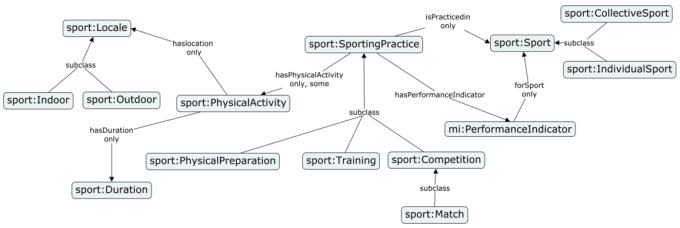

Fig. 8. Sport Module

b) The Measurement and Indicator (MI) module: This module specifies and classifies measures and indicators defined in the project framework. The alignment between the MI module and the DUL ontology is shown in Figure 9. A classification of measures and indicators can be created as a specialization of mi:Measurement and mi:Indicator.

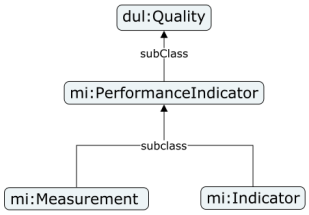

Fig. 9. Measurement and Indicator Module

V. Semantic (re-)configuration of Smart Sensors

The (re-)configuration of a smart sensor is defined by the addition of new functionalities according to the context of use. Our approach is based on two main aspects: (1) the declarative aspect of (re-)configuration that takes advantages of the potentialities offered by ontologies and (2) the fine granularity of the software components to be installed, namely the "snippets". This last point is very useful because smart sensors are constrained devices. In this section, wedescribe the principle of our semantic approach to (re-)configuration of smart sensors, then we describe the proposed mechanism.

A. Reconfiguration Principles

A smart sensor (s3n:SmartSensor) is reconfigured upon a user's request. For example, in the scenario of Section II-B, when Abdel changes his sporting activity, the sensor integrated in his shorts must return new indicator values related to his new sporting activity. Thus, these are new performance indicators

(mi:PerformanceIndicator) to calculate. This infers new requirements that the smart sensor must be able to answer to by installing and executing appropriate algorithms.

The execution of an algorithm is described using the S3N-Algorithm module. This is done by the execution of a set of s3n:Snippets. These snippets are fragments of code which are reusable by several s3n:Algorithms. We define a configuration as a set of snippets to install in a smart sensor to execute a given algorithm (**Configuration = {snippet}**). Because of the limited capabilities of a smart sensor, it is important to reconfigure code only if necessary. In other words, a smart (re-)configuration would install only snippets that do not already exist in the s3n:Memory of the s3n:MicroController of the s3n:SmartSensor. Thus, during the (re-)configuration of a smart sensor (for example, <http://example.org/data/SMSACTI>) two important points must be checked: (1) the smart sensor has adequate basic sensors (sosa:Sensor), i.e., the integrated basic sensors provide the right measurements. For example algorithm `<http://example.org/data/HydrationAlertAlgorithm>` calculates indicator `<http://example.org/data/Hydration-Rate>`. This algorithm will need two measurements, namely, temperature and acceleration, thus `<http://example.org/data/SMSACTI>` should incorporate at least one accelerometer and one thermometer. (2) the capabilities of the microcontroller are satisfying, i.e., it has sufficient memory capacity to store code fragments, temporary data or persistent data for calculation.

B. Semantic mechanism of reconfiguration

Let (C) be the configuration of a smart sensor for a given sport. The sequence of snippets to be installed or already installed in this smart sensor. This configuration (C) refers to a plan (P) of execution of these Snippets. When the user decides to change the use of his smart sensor (integrated with the smart garment) for another sport, the configuration of the smart sensor must change. All the following steps are then performed:

1) The user reports the change of activity;
2) SPARQL queries are executed whose parameters are based on the new usage context (the new sport);
3) The reasoning engine runs the queries to select the new snippets required;
4) The result of the selection is refined considering the constraints of the smart sensor such as the remaining memory space, the size of the snippets needed for the new use, etc.;
5) The final list of snippets is identified with a new execution plan (P');
6) If (re-)configuration is possible, the description of this configuration is sent to the smart sensor;
7) The smart sensor compares snippets available locally with what is required;
8) The smart sensor triggers the operations required to uninstall, and install the new missing snippets.

Algorithm 1 shows the set of instructions to be executed after choosing a sport and a performance indicator to compute. *Smartsensor*, *sport* and *indicator* are input variables. This (re-)configuration mechanism was designed during the construction of the ontological modules S3N-Algorithm, S3N-Core, MI and Sport. A set of SPARQL queries and inference rules has been designed and developed to address the different possible cases of (re-)configuration of a smart sensor (see section VI-B). The developed queries are executed during the different steps of the (re-)configuration mechanism. As an example, the query below returns the list of performance indicators for a given sport, which is passed as parameter by the user using a placeholder variable `?sport`:

```
SELECT DISTINCT ?performanceindicator
WHERE {
?pr a sport:SportingPractice ;
   sport:isPracticedin ?sport ;
   sport:hasPerformanceIndicator ?performanceindicator.
?performanceindicator a mi:PerformanceIndicator.
} ORDER BY ASC (?performanceindicator)
```

Display(*sport*, *indicator*) ;
ontology ← *CreateModel*() ;
Tab ← *Precondition*(*ontology*, *indicator*) ;
*Tab*1 ← *PropSmartsensor*(*ontology*, *Smartsensor*) ;
hard ← *Compare*(*Tab*, *Tab1*) ;
MemorySize_Micro ←
MemorySizeMicro(*ontology*, *Smartsensor*) ;
ComputationalCost_Snippet ← *Snippet*(*ontology*, *indicator*);
if *(MemorySize_Micro* ≤ *ComputationalCost_Snippet)* **then**
  soft ← 0 ;
else
  soft ← 1 ;
end
if *hard* = 0 **then**
  Display"Hardware check failed." ;
else
  Display "Hardware check passed." ;
  if *soft* = 0 **then**
    Display "Memory check failed." ;
  else
    Display "Memory check passed." ;
    ExecutePlan(*ontology*, *indicator*) ;
  end
end

Algorithm 1: Algorithm for the (re-)configuration.

VI. Proof of concept for semantic reconfiguration

This section reports on a SmartSensing application developed to take into consideration the different (re-)configuration scenarios for a smart sensor.

A. SMS ontology assertions examples

Assertions of the knowledge database are instantiations of the SMS ontology modules. They are defined to describe the environment and needs of the SmartSensing project users. Below are the namespaces and prefixes used in the Turtle snippets.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix sosa: <http://www.w3.org/ns/sosa/> .
@prefix ssn: <http://www.w3.org/ns/ssn/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix qudt-1-1: <http://qudt.org/1.1/schema/qudt#> .
@prefix qudt-unit-1-1: <http://qudt.org/1.1/vocab/unit#> .
```

```
@prefix dul: <http://www.ontologydesignpatterns.org/ont
      /dul/DUL.owl#> .
@prefix mi: <http://w3id.org/MI/> .
@prefix s3n: <http://w3id.org/s3n/> .
@prefix sport: <http://w3id.org/Sporst/> .
@prefix ssn-system: <http://www.w3.org/ns/ssn/systems/> .
@base <http://example.org/data/> .
```

Examples of collective sports :

```
<Basketball> rdf:type sport:CollectiveSport ;
  rdfs:label "Basketball"@en ;
  rdfs:comment "Basketball is a Collective Sport."@en .
<Football> rdf:type sport:CollectiveSport ;
  rdfs:label "Football"@en ;
  rdfs:comment "Football is a Collective Sport."@en .
<Rugby> rdf:type sport:CollectiveSport ;
  rdfs:label "Rugby"@en ;
  rdfs:comment "Rugby is a Collective Sport."@en .
```

The performance indicators for the Running :

```
<LongTrailRunning> rdf:type sport:SportingPractice ;
  rdfs:label "Long trail running"@en ;
  rdfs:comment "Long trail running is a practice of
       Running."@en ;
  sport:isPracticedin <Running> ;
  sport:hasPerformanceIndicator <CaloriesConsumedNumber> ,
       <HydrationRate> , <StepNumber> , <AverageSpeed> ,
       <Tiredness> , <Geolocalisation> .
```

Declaration of the algorithm associated with calculating calories consumed :

```
<AlgoCaloriesConsumed> rdf:type s3n:Algorithm ;
  rdfs:label "Algo calories consumed"@en ;
  rdfs:comment "Algo calories consumed is an algorithm."@en
       ;
  s3n:hasOperation <AccelerationOperation> ,
       <SpeedOperation> , <DistanceOperation> ,
       <CaloriesConsumedOperation> ;
  s3n:hasPostCondition <CaloriesConsumedNumber> ;
  s3n:hasPreCondition <Acceleration>.
```

Declaration of an operation of the 'AlgoCaloriesConsumed' algorithm and its implementation by a snippet :

```
<AccelerationOperation> rdf:type s3n:Operation ;
  rdfs:label "Acceleration Operation"@en;
  rdfs:comment "Acceleration operation is an Operation."@en;
  s3n:hasImplementation <AccelerationSnippet>;
  s3n:hasProcedureFeature <AccelerationFeature>.
<SequenceAccOpAlgoCalCons> rdf:type s3n:SequenceNumber;
  s3n:ofOperation <AccelerationOperation> ;
  s3n:forAlgorithm <AlgoCaloriesConsumedNumber> ;
  s3n:hasSequenceNumberValue "1"^^xsd:int .
```

Assigning a value for the memory space needed to execute the 'AccelerationOperation' operation :

```
<AccelerationFeature> rdf:type s3n:ProcedureFeature;
  rdfs:label "Acceleration feature"@en;
  rdfs:comment "Acceleration feature is a Procedure
       Feature."@en;
  s3n:hasProcedureProperty <AccelerationSpace>.
<AccelerationSpace> rdf:type s3n:ProcedureProperty;
  rdfs:label "Acceleration space"@en;
  rdfs:comment "Acceleration space is Procedure
       Property"@en;
  s3n:hasProcedurePropertyValue "16"^^xsd:int ;
  qudt-1-1:unit qudt-unit-1-1:Bit.
```

Addition of a smart sensor and its components :

```
<SmartSportSMS16> rdf:type s3n:SmartSensor ;
  rdfs:label "SmartSportSMS16"@en ;
  rdfs:comment "SmartSportSMS16 is a \rev{smart sensor} of
       Sport."@en ;
  ssn:hasSubSystem <TemperatureLM124/1> ,
       <MicoControllerDS4830/20>.
<TemperatureLM124/1> rdf:type sosa:Sensor ;
  rdfs:label "TemperatureLM124 #1"@en ;
  rdfs:comment "TemperatureLM124 #1 is a Temperature
       sensor."@en ;
  sosa:observes <Temperature> .
<MicoControllerDS4830/20> rdf:type s3n:MicroController ;
  rdfs:label "MicoControllerDS4830 #20"@en ;
  rdfs:comment "MicoControllerDS4830 #20 is a
       Mico-Controller, it is of DS4830 Reference."@en ;
  ssn-system:hasSystemCapability
       <SystemCapabilityMicroDS4830> .
```

Specify the memory capacity of a microcontroller :

```
<MemoryDS4830> rdf:type ssn-system:SystemProperty ;
  rdfs:label "MemoryDS4830"@en ;
  rdfs:comment "MemoryDS4830 is a memory of
       Mico-ControllerDS4830."@en ;
  s3n:hasSystemPropertyValue "16"^^xsd:int ;
  qudt-1-1:unit qudt-unit-1-1:Bit.
```

B. Using the SmartSensing application for reconfiguration

Abdel connects to his application via the interface in Figure 10. He can then choose the sport to exercise and the performance indicator to compute. The result displayed through the application is based on Abdel's choice and the smart sensor he owns. Thus, we have chosen three scenarios to illustrate the use of the (re-)configuration mechanism and its interaction with the knowledge base defined in this work.

Scenario 1. Abdel owns the material 'SmartSportSMS21' that is a smart sensor containing two sensors: a gyroscope and an accelerometer. Abdel chooses to calculate the body temperature for the sport 'Basketball'. The algorithm that computes this indicator needs the measurement of temperature as a pre-condition. Since the smart sensor does not contain a temperature sensor, operation is not possible. Figure 11 shows the displayed result.

Scenario 2. The same user with the same SmartSensor changed the sport and the indicator. In this scenario, Abdel wants to compute his fatigue rate by practicing 'football'. For the calculation of this indicator (Tiredness), we need acceleration as a precondition. The SmartSensor in this case contains the hardware (sensor) needed to compute this indicator, but the memory of its microcontroller is insufficient. Figure 12 shows the displayed result.

Scenario 3. The user chooses now to compute his location during the practice of running sport. The constraints in software and hardware are satisfied. He can thus begin his activity (see Figure 13).

The snippets to be installed on the hardware and their execution plan are sent back to the smart sensor. For example, the execution plan for algorithm `<Geolocalisation>` on smart sensor `<SmartSportsSMS21>` may consist of the following list of snippets: `<PositionSnippet>`, `<LatitudeSnippet>`, `<LongitudeSnippet>`, `<AltitudeSnippet>`, `<LocationSnippet>`.

VII. Conclusion

In the present paper we have tackled issues related to semantic modeling and representation of Smart Objects. We proposed for that aim the SMS ontology, which is a set of ontological modules that extend SOSA/SSN, DUL, and WoT TD, and allows to describe the structure and behavior of Smart Objects. Based on these ontologies, we also proposed a semantic Web based approach that enables to deal semantically

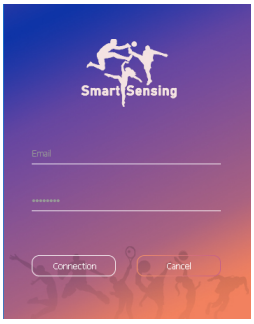

Fig. 10. SmartSensing Application: to connect.

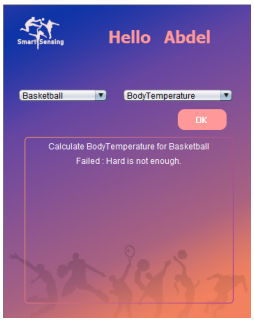

Fig. 11. Result of the Scenario 1.

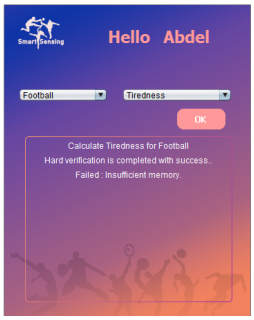

Fig. 12. Result of the Scenario 2.

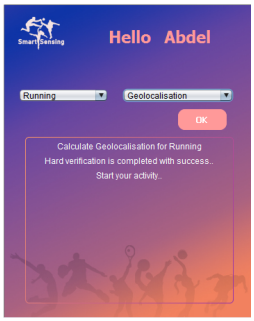

Fig. 13. Result of the Scenario 3.

with the (re-)configuration of the embedded behavior of Smart Objects. The (re-)configuration mechanism makes decisions about the software snippets to install or uninstall in order to answer accurately and on time to the domain's requirements. It is equally driven by the Smart Objects characteristics and their usage context. The novelty of our approach with respect to the related work described in Section III relies in the fact that it is semantic based, therefore the (re-)configuration decision is not hard coded but inferred dynamically according to the characteristics of the Smart Object and its usage context and domain. Its application to the smart sensor is therefore done at runtime. We aim in the future to reuse the set of ontological modules of the SMS ontology for two kinds of applications: the first one is to help developing a Smart cloth ecosystem that is dedicated for sports. The second kind of application is to develop a co-design framework for Smart Objects. Both applications rely on the modularity of the proposed ontologies along with the inference and reasoning facilities.

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] S. Sagar, M. Lefrançois, I. Rebai, K. Maha, S. Garlatti, J. Feki, and L. Médini, "Modeling Smart Sensors on top of SOSA/SSN and WoT TD with the Semantic Smart Sensor Network (S3N) modular Ontology," in *Emerging Topics in Semantic Technologies, {ISWC} 2018 Satellite Events*, E. Demidova, A. J. Zaveri, and E. Simperl, Eds. Berlin: AKA Verlag, 2018.

[3] L. Mottola, G. P. Picco, and A. A. Sheikh, "FiGaRo: Fine-Grained Software Reconfiguration for Wireless Sensor Networks," in *Wireless Sensor Networks, 5th European Conference, {EWSN} 2008, Bologna, Italy, January 30-February 1, 2008, Proceedings*, 2008, pp. 286–304. [Online]. Available: https://doi.org/10.1007/978-3-540-77690-1_18

[4] J.-P. Fassino, J.-B. Stefani, J. L. Lawall, and G. Muller, "Think: A Software Framework for Component-based Operating System Kernels." in *USENIX Annual Technical Conference, General Track*, 2002, pp. 73–86.

[5] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani, "The fractal component model and its support in java," *Software: Practice and Experience*, vol. 36, no. 11-12, pp. 1257–1284, 2006.

[6] G. Coulson, G. Blair, P. Grace, F. Taiani, A. Joolia, K. Lee, J. Ueyama, and T. Sivaharan, "A Generic Component Model for Building Systems Software," *ACM Trans. Comput. Syst.*, vol. 26, no. 1, March 2008. [Online]. Available: https://doi.org/10.1145/1328671.1328672

[7] A. Taherkordi, Q. Le-Trung, R. Rouvoy, and F. Eliassen, "WiSeKit: A distributed middleware to support application-level adaptation in sensor networks," in *IFIP International Conference on Distributed Applications and Interoperable Systems*, 2009, pp. 44–58.

[8] A. Taherkordi, R. Rouvoy, Q. Le-Trung, and F. Eliassen, "Supporting lightweight adaptations in context-aware wireless sensor networks," in *Proceedings of the 1st International Workshop on Context-Aware Middleware and Services: affiliated with the 4th International Conference on Communication System Software and Middleware (COMSWARE 2009)*, 2009, pp. 43–48.

[9] A. Taherkordi, F. Loiret, R. Rouvoy, and F. Eliassen, "Optimizing sensor network reprogramming via in situ reconfigurable components," *ACM Transactions on Sensor Networks (TOSN)*, vol. 9, no. 2, p. 14, 2013.

[10] P. Costa, G. Coulson, R. Gold, M. Lad, C. Mascolo, L. Mottola, G. P. Picco, T. Sivaharan, N. Weerasinghe, and S. Zachariadis, "The RUNES middleware for networked embedded systems and its application in a disaster management scenario," in *Pervasive Computing and Communications, 2007. PerCom'07. Fifth Annual IEEE International Conference on*, 2007, pp. 69–78.

[11] P. J. Marrón, M. Gauger, A. Lachenmann, D. Minder, O. Saukh, and K. Rothermel, "FlexCup: A flexible and efficient code update mechanism for sensor networks," in *European Workshop on Wireless Sensor Networks*, 2006, pp. 212–227.

[12] D. Hughes, K. Thoelen, W. Horré, N. Matthys, J. D. Cid, S. Michiels, C. Huygens, and W. Joosen, "LooCI: A Loosely-coupled Component Infrastructure for Networked Embedded Systems," in *Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia*. New York, NY, USA: ACM, 2009, pp. 195–203. [Online]. Available: https://doi.org/10.1145/1821748.1821787

[13] S. Sagar, I. Reba, M. Khemaja, and J. Feki, "Ontologie modulaire pour la fabrication et l'exploitation de vêtements intelligents dédiés au sport," in *28es Journées francophones d'Ingénierie des Connaissances IC 2017*, 2017, pp. 139–144.

[14] A. Haller, K. Janowicz, S. J. D. Cox, D. Le Phuoc, K. Taylor, and M. Lefrançois, "Semantic Sensor Network Ontology," W3C Recommendation, October 19 2017. [Online]. Available: https://www.w3.org/TR/vocab-ssn/

[15] C. Masolo, S. Borgo, A. Gangemini, N. Guarino, A. Oltramari, and L. Schneider, "The WonderWeb Library of Fundational Ontologies and the DOLCE ontology," LOA-ISTC, Technical report, 2003.

[16] S. Kaebisch, T. Kamiya, M. McCool, and V. Charpenay, "Web of Things (WoT) Thing Description," W3C Candidate Recommendation, May 16 2019. [Online]. Available: https://www.w3.org/TR/wot-thing-description/

[17] M. Lefrançois, "Planned ETSI SAREF Extensions based on the W3C&OGC SOSA/SSN-compatible SEAS Ontology Patterns," in *Proceedings of Workshop on Semantic Interoperability and Standardization in the IoT, SIS-IoT,*, July 2017.

[18] X. Aimé, S. George, and J. Hornung, "VetiVoc: a modular ontology for the fashion, textile and clothing domain," *Applied Ontology*, vol. 11, no. 1, pp. 1–28, 2016.