Semantic Plug and Play: An Architecture Combining Linked Data and Reconfigurable Hardware

Constantin Wanninger[1],
Luca Alfano[2], Martin Schörner[3], Alwin Hoffmann[4], Oliver Kosak[5], Wolfgang Reif[6]
Institute for Software and Systesm Engineering
University of Augsburg
Augsburg, Germany
Email: {wanninger[1], alfano[2], schoerner[3], hoffmann[4], kosak[5], reif[6]}@isse.de

***Abstract*—Through the mechanisms of the Semantic Web, it is possible not only to describe web content syntactically but also to relate it semantically. The properties and capabilities of hardware, instead, are hidden in documents, code documentations, repository descriptions, etc. This paper presents a methodology and architecture that can be used to describe and relate the properties and capabilities of hardware. The decentralized storage of the descriptions on a hardware adapter allows the information to be evaluated at runtime. For domain-specific applications a Model-Domain-Domainmodel Architecture (MDDM) is presented so that code can also be executed at runtime using these hardware descriptions. The architecture is presented using a home automation system with single-board computers and microcontrollers, in which sensors and actuators can be exchanged and integrated.**

I. Introduction

The origin of the Word Wide Web (WWW) is based on the markup language HTML, the transfer protocol HTTP and the resource description language URL [1]. A major weakness of these technologies is that the information they provide is not machine-readable and cannot be described explicitly. To overcome this disadvantage 2006 Lee coined the term *Linked Data* in a design note about the *Semantic Web* project [2]. Linked Data is interlinked structured data and built upon standard web technologies such as RDF and URIs to enrich the unstructured content of the WWW with semantic, machine readable information with a best practice strategy.

This paper introduces Semantic Plug and Play, a methodology and architecture for distributing semantic annotations to concrete hardware elements and using them domain-independently in object-oriented programming languages. For this purpose, a procedure is presented in which domain-specific requirements can be defined in the code, which are evaluated at run-time using the concrete hardware setup. Thereby in particular the close connection between hardware and software is to be loosened in order to support e.g. configurability, expandability and modularity, which is necessary for modular systems with physical configuration possibilities. In the architecture to be proposed, **capabilities** and **properties** of the hardware are supposed to be abstracted and provided at runtime by a **distributed knowledge base**. In the object-oriented programming language instead capability **requirements** are defined, which can be fulfilled at runtime or require a reconfiguration of the hardware.

This research was funded by the German Research Foundation (DFG), grant number 402956354.

II. Related Work

For plug and play strategies, Shanley et al. [3] set up an architectural paradigm as early as 1995. The required drivers of an externally connected hardware element are mapped via IDs with a data storage existing on the operating system. Generic drivers, like device classes in the USB protocol [4] or in protocols like the Transducer Electronic Sheets [5] can be used if the hardware elements are similar in their characteristics. Another variant is to store the corresponding drivers on the hardware element itself (e.g. USB Plug and Play [4]). All mentioned variants are based on (generic) drivers as interface between hardware and software, which have to be connected either individually implemented (e.g. Kinect with an SDK) or generically (e.g. USB-keyboard), without the consideration of special characteristics.

The Industry 4.0 initiative encompasses the intermeshing of modern information and communication technology in the domain of industrial production. As a subgoal of Industry 4.0, self-description mechanisms within the documentation of an administrative asset shell are also defined [6] and taken up in research [7], [8]. AutomationML [9], Semantic Web technologies [10] or the meta-model of the OPC-UA architecture [11] are suggested as data structure for a self-description [6], [8]. The focus, however, is not on the concrete use of the self-description in high-level programming languages but rather on the (interdisciplinary) networking and information exchange of the production cells and the provision of this information.

In addition to the predicted use in the industrial domain, self-description through Semantic Web technologies is already being used in domains where measured values play a major role, such as geography or meterology [12], which can be seen in the Linked Open Data Cloud [13]. Ontologies for measured values and their assignment to sensors and deployed platforms like the Semantic *Sensor Network Ontology* (SSN) [14] can be used to share measured values over the Semantic Web.

To our knowledge, an architecture for storing and using the self-description in object-oriented programming languages at runtime of modular sensors and actuators does not yet exist. In earlier publications of our research group [15], [16], [17] the foundation for this architecture was laid and shown in single and multi robot systems.

III. Semantic Hardware Web

The following section is dedicated to the data model of self-description of hardware elements, which is called Semantic Hardware Web. Semantic Web technologies are used for this, such as the *Resource Description Framework* (RDF) and the Web Ontology Language (OWL). Previous existing descriptions like SSN [14] are designed for sensors, and are more suitable for describing measured values than actual functionality. For easy categorization within capabilities and properties, existing OWL ontologies can be enhanced with the Semantic Hardware Web.

The main characteristic of the ontology is the division between descriptive properties of hardware elements and the executable capabilities (see fig. 1). *Properties* can be device specific, such as the weight of a temperature sensor module, but they can also describe capabilities in more detail, such as the fact that the temperature sensor measures in degrees Celsius. The granularity of the properties is explicitly not firmly defined, since the sensor data and the necessary self-descriptions are highly domain-specific. For example, a home automation system requires only an average room temperature value, whereas for geographic measurements [18] the accuracy and latency of the sensor is essential. The different requirements of the domains also led to many models [12], [14], which are not standardized in some places. The presented Semantic Hardware Web Ontology on the other hand offers only an abstract basis for descriptions, the essential new aspect consists in the *capabilities*.

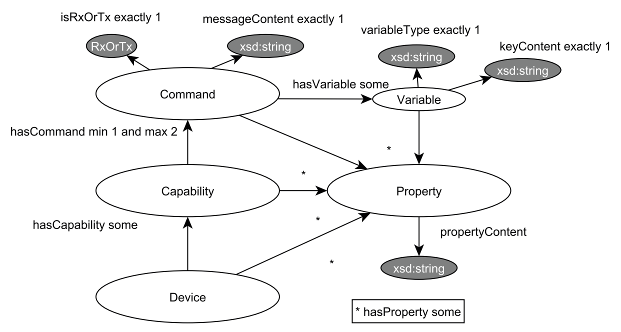

Fig. 1: The Semantic Hardware Web ontology with the most important core concepts (white boxes), dependencies (arrows) and constraints (arrow descriptions). The data types are indicated in the gray boxes

Capabilities are actual operations that can be performed on the hardware, such as the *measure():float* capability of the temperature sensor. They can be triggered via *commands* (see fig. 1) and used within the architecture of Semantic Plug and Play. There are two types of commands, a send and a receive command with different variables, where a capability has a maximum of two commands. *Variables* can also be extended with properties, e.g. to convert the usual code documentation into a completely machine-readable form. Properties that can change at runtime (as shown in a previous work [15]), such as the current offset of a calibrated sensor can only be assigned to a concrete sensor. The decentralized storage mechanisms of the Semantic Web are ideally suited for storing instant-specific descriptions on the hardware itself and linking them to generic descriptions on the Internet.

IV. Architecture

This section introduces the architecture behind Semantic Plug and Play. Starting with the decentralized storage of the Semantic Hardware Web up to the use of self-description in object-oriented programming languages. In this section the software components are in the foreground, the distribution on real hardware is presented in section V afterwards. The most important software components as shown in fig. 2 are the **semantic adapter**, the **semantic controller**, and the Object Oriented Design interface **(OOD interface)**.

A. Semantic Adapter

The semantic adapter is an abstraction layer that unifies the multitude of different devices. By implementing one of the existing interfaces it offers further the functionality of adjacent devices through its description to the outside world. However, the semantic adapter does not provide an interface for high-level languages. Therefore, another module is needed, which has the task to bundle the self-descriptions of all semantic adapters and make them available for high-level programming languages.

B. Semantic Controller

Consisting of several components, the semantic controller forms the bridge between the semantic adapters and an high-level program (see fig. 2), that will utilize the hardware and its

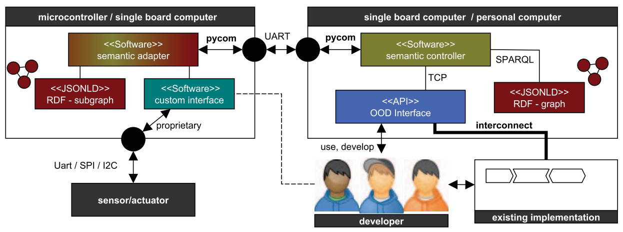

Fig. 2: The distribution of the software artifacts of Semantic Plug and Play to supported hardware nodes (microcontroller, single board computer, etc.) with physical (UART, SPI, etc.) and software interfaces (pycom, SPARQL, etc.).

description. The task of the semantic controller is to collect, complete and provide the self-descriptions, to assign them to the corresponding sensors and actuators, and to address the communication between a high-level language and the used hardware. Thus, semantic controller forms the final abstraction layer and provides the end user interface, which is assumed to be written in an object-oriented language with capabilities and properties of the underlying physical devices.

C. OOD interface

The Object Oriented Design interface (OOD interface) is a programming interface written in an object-oriented language, which allows the use of physical devices and its self description. The capabilities and properties of physical devices are provided by classes within the OOD interface. These can be instantiated at runtime depending on the domain preference. On the other hand, a domain-specific implementation can also make requests via capabilities or properties and check at runtime whether the available hardware meets the requirements. For example, if the temperature sensor needs a certain latency for a specific measurement, this can be defined as a requirement. This request is checked at runtime and the result is communicated to the user. The functionality behind requirements-based programming is hidden in a new architecture, in which the domain of development is in focus and not another generic solution needs to be established.

D. MDDM architecture

The above mentioned concepts of semantic adapter and controller and the underlying self-description are based on an architecture developed exclusively for semantic plug-and-play, the **Model-Domain-DomainModel** (MDDM) architecture which was derived from the *MVVM* pattern used for GUI development [19]. Analogous to the MVVM-pattern, this pattern uses data-bindings (see fig. 3) to link information from self-descriptions to the functionality required in the domain, in which the application logic is situated. In addition to the application logic, the domain also hosts a mechanism to request specific hardware capabilities. To accomplish this,

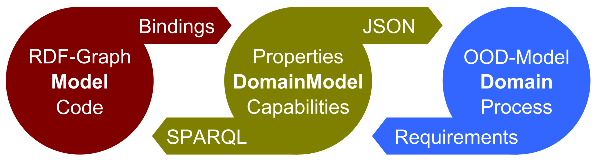

Fig. 3: The Model-Domain-DomainModel (MDDM) architecture in a nutshell. The model stands for an overall graph of all self-descriptions of the system, as well as the corresponding hardware connections. The domain on the other hand wants to use hardware elements in its own structures and has corresponding requirements. The mapping between the requirements of the domain and the possibilities offered by the current system takes place in the DomainModel.

requirements are passed to the DomainModel, which is used to map them to capabilities and properties derived from the model using SPARQL, a protocol for querying Semantic Web data. The model, similar to the model from the MVVM-pattern, offers its information in the form of an RDF-graph to the DomainModel to be bound. After a successful match, a JSON file with capability and property instances, as well as the corresponding bindings are created and made available to the domain. For example a temperature sensor, which will be implemented by a semantic adapter, has one capability (measuring) and some properties. The domain requires a temperature sensor that measures in degree celcius. This requirement will be compared to the information gathered by the semantic controller as soon as a semantic adapter is connected to it. If it matches, the full data of the self-description will be sent to the end user interface to connect the software representation of the temperature sensor which now can be used to generate measurements via bindings.

V. Implementation/Realization

For the verification of the architecture presented in the previous section, the distribution of hardware nodes, the communication between different components as well as the mapping procedure are described in this section. The deployment of the components as well as the fundamental communication between them are shown in fig. 2.

A. OOD-API

For realizing the OOD interface (see sec. IV) a software called **OOD-API** is proposed. This software establishes a connection via TCP to a semantic controller that provides the capabilities and properties of the attached hardware devices. The class structure of the API is based on the ontology of the Semantic Hardware Web (see fig. 1) Central component is the class *device*, which maintains the capabilities and acts as a virtual representative of the real hardware. *Capabilities* can be used to trigger send commands that initiate measurements, for example. When a measurement is performed, a receive command triggers an event that calls a predefined routine (see *"insert routine"* in listing 1). Devices are initiated by mapping the RDF-graph (using SPARQL) to requirements which can be defined in the instantiating process of a *device* (e.g. "temperature" in the constructor of the *ts* device).

```
ts = new Device("temperature");
ts.registerCapabilityListener
 (0,(Object... o)->{ insert routine });
```

Listing 1: A (java-) lambda expression determining the behavior on receiving a measurement.

B. Semantic Controller

The semantic controller is written in Python and handles both RDF-graph and the management of the connected semantic adapters. Further it communicates with the OOD-API via TCP serving as middleware. It pipes down commands (such as invoking a measurement) from the OOD-API by addressing the right sensor/actuator and translating it to a form understandable to the semantic adapter. To detect and analyze a new physical device, the subgraph is requested, whereupon a (JSON) serialization of the class device of the OOD-API, is associated with information about the new device.

C. Semantic Adapter

This software is used on a microcontroller on which the interfaces of the sensors or actuators are implemented. For the communication between the Semantic Controller and Adapter, a self-developed protocol called **pycom** is used to ensure a fixed latency, a correct order of possible parameters (e.g. measurement results) and an interrupt-based streaming of data. To connect a sensor or actuator to the Semantic Adapter, the physical interfaces of the microcontroller can be used (In the example from fig. 2: UART, SPI and I2C). In the Semantic Adapter, the custom interface must be converted to pycom. For this purpose, appropriate C/C++ pycom libraries are available, which also guarantee the storage and exchange of RDF graphs on the microcontroller.

VI. Proof of concept

To illustrate the concepts implemented in the previous section, a small experiment has been conducted using real hardware with a variety of sensors and actuators. In this experiment a home automation system was developed, which controls actuators on the basis of sensor values. The semantic controller distributed on a personal computer provides the interface for the developer through the OOD-API. Different sensors (soil, ultra sonic, air quality, temperature, motion) and actuators (thermostat, display, lights) are connected to several microcontrollers (ESP 8266, Arduino Nano and Uno), each running an instance of Semantic Adapter. On this basis, both automated rules, such as temperature-based heating control when people are in the room, and suggestions via the display, such as the air quality is poor, it must be ventilated, were developed using requirements in Java. Developers can request an overview of the connected hardware via the console to define these requirements. The OOD-API can even be used if hardware elements are replaced with ones with similar capabilities and code can be written without knowledge of the concrete hardware it is using.

VII. Conclusion

In this paper, mechanisms were presented to connect different sensors and actuators, describe them semantically and use them in high-level languages (like Java). By abstracting the capabilities and properties, the direct implementation of the hardware becomes independent from the process logic. The development of more complex programs based on hardware data (e.g. measured distance) and functions (e.g. driving a robot) is greatly simplified by the developed framework. This kind of abstract programming can be trend-setting, similar to the ever increasing abstraction in programming languages, such as the development of object-oriented languages.

In addition, the self-description necessary for this mechanism enables the decentralized storage of this data, whereby failure-prone interfaces, such as Internet connections, are not absolutely necessary. This means that systems implementing the framework can function in locations that do not have an Internet connection. For example, the software for in situ measurements during geography campaigns [18] could be programmed in this way. By simply replacing faulty or more current hardware while retaining the same process logic, development and integration costs can also be saved and maintainability improved.

References

[1] The original proposal of the www. W3C Consortium. [Online]. Available: https://www.w3.org/History/1989/proposal.html

[2] T. B. Lee. W3C Consortium. [Online]. Available: https://www.w3.org/DesignIssues/LinkedData.html

[3] T. Shanley, *Plug and play system architecture*. Addison-Wesley Professional, 1995.

[4] D. Anderson, *USB system architecture*. Addison-Wesley Professional, 1997.

[5] J. E. Higuera and J. Polo, "Ieee 1451 standard in 6lowpan sensor networks using a compact physical-layer transducer electronic datasheet," *IEEE Transactions on Instrumentation and Measurement*, vol. 60, no. 8, pp. 2751–2758, 2011.

[6] Plattform Industrie 4.0, "Structure of the administration shell," Apr. 2016. [Online]. Available: https://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/structure-of-the-administration-shell.html

[7] F. Palm and U. Epple, "openAAS - Die offene Entwicklung der Verwaltungsschale," in *Automation 2017 : technology networks processes*, vol. 2293. VDI Verlag GmbH, 2017, pp. 103–104. [Online]. Available: https://publications.rwth-aachen.de/record/691900

[8] I. Grangel-González, L. Halilaj, G. Coskun *et al.*, "Towards a semantic administrative shell for industry 4.0 components," in *Semantic Computing (ICSC), 2016 IEEE Tenth Intern. Conf. on*. IEEE, 2016, pp. 230–237. [Online]. Available: http://arxiv.org/pdf/1601.01556

[9] R. Drath, *Datenaustausch in der Anlagenplanung mit AutomationML: Integration von CAEX, PLCopen XML und COLLADA*. Springer-Verlag, 2009.

[10] T. Berners-Lee, J. Hendler, O. Lassila *et al.*, "The semantic web," *Scientific american*, vol. 284, no. 5, pp. 28–37, 2001.

[11] T. Hannelius, M. Salmenpera, and S. Kuikka, "Roadmap to adopting opc ua," in *2008 6th IEEE Int. Conf. on Industrial Informatics*, 2008, pp. 756–761.

[12] M. Dibley, H. Li, Y. Rezgui, and J. Miles, "An integrated framework utilising software agent reasoning and ontology models for sensor based building monitoring," *Journ. of Civil Engineering and Management*, vol. 21, no. 3, pp. 356–375, 2015.

[13] J. P. McCrae. The lod cloud. [Online]. Available: https://lod-cloud.net/

[14] "Semantic Sensor Network Ontology," W3C, 2017. [Online]. Available: https://www.w3.org/TR/vocab-ssn/

[15] C. Wanninger, C. Eymüller, A. Hoffmann *et al.*, "Synthesising Capabilities for Collective Adaptive Systems from Self-Descriptive Hardware Devices - Bridging the Reality Gap," in *8th Int. Symp. On Leveraging Appl. of Formal Methods, Verification and Validation*, Sept 2018.

[16] C. Eymüller, C. Wanninger, A. Hoffmann *et al.*, "Semantic Plug and Play – Self-Descriptive Modular Hardware for Robotic Applications," in *International Journal of Semantic Computing (IJSC)*.

[17] O. Kosak, C. Wanninger, A. Hoffmann *et al.*, "Multipotent systems: Combining planning, self-organization, and reconfiguration in modular robot ensembles," *Sensors*, vol. 19, no. 1, 2018. [Online]. Available: http://www.mdpi.com/1424-8220/19/1/17

[18] B. Wolf, C. Chwala, B. Fersch *et al.*, "The scalex campaign: Scale-crossing land surface and boundary layer processes in the tereno-prealpine observatory," *Bulletin of the American Meteorological Society*, vol. 98, no. 6, pp. 1217–1234, 2017. [Online]. Available: https://doi.org/10.1175/BAMS-D-15-00277.1

[19] A. Syromiatnikov and D. Weyns, "A journey through the land of model-view-design patterns," in *2014 IEEE/IFIP Conference on Software Architecture*, 2014, pp. 21–30.