A Framework for Semantic Description and Interoperability of CPS Modules

Amita Singh[1], Fabian Quint[2], Patrick Bertram[3] and Martin Ruskowski[2]

***Abstract*— Assistance systems are machines which assist humans in completing tasks. With the advent of Industry 4.0 and human-in-the-loop paradigms, they are now increasingly common in production facilities to help workers, and, consequently, there has been a surge of interest in the field. However, most recent designs proposed in the literature suggest are monolithic and allow only limited modifications.**

In contrast, we will treat an assistance system as consisting of a central entity containing the process description (or instructions) and one or more plug-and-play Cyber-Physical Systems (CPS modules) to retrieve relevant information from the physical environment. Such a design will allow the overall system capabilities to be adapted to the needs of workers and tasks. We will provide a framework for designing the CPS modules and describing them semantically using ontologies which will allow (i) interpretation of all data, and (ii) interoperability among the modules, from the very outset. We will show, with the help of examples, how to use the framework to design a module, how the modules can exchange data among themselves, and how to build a sustainable ecosystem of ontologies which will enable rapid development of third-party CPS modules. We will provide a prototype implementation using Protégé and will discuss future direction of research.

I. Introduction

An ever growing catalogue of products, short product life-cycle, competitive product costs, and changing demographics have led to a demand of reactive and proactive production systems that can adapt to the changing needs. Neither complete automation nor manual assembly can alone cater to the flexibility required to adapt to the volatility of production instructions, thus making the collaboration of human and automated systems a necessity. Human-centered CPS (H-CPS) are an example of such a collaborative automation where human is a part of the control loop. This means both CPS and human affect each other: CPS helps workers during processes while taking feedback from the worker's actions at the same time.

Assistance systems are H-CPS used in production systems. These systems assess the process and provide instructions to workers. Although humans have unparalleled degree of flexibility, i.e. humans can adapt to varying production, major focus is being placed on increasing the flexibility of automation systems that help workers during processes. Emerging developments like modularity [1], service-oriented architecture (SoA) [1], interoperability by the virtue of common semantic description (e.g. administrative shell [2], [3]), and edge-computing [4] are rarely applied to H-CPS.

We envision a CPS-based assistance system which adapts to the worker's need by exploiting the benefits of such techniques. Our assistance system has a central system and one or more CPS modules. CPS modules feed the central system data extracted from the environment. The central system, in turn, processes this data to assess the state of the process and worker's needs. This paper presents a framework for designing CPS modules of an assistance system which promotes interoperability across modules.

In the remainder of the paper, we will first review the related work in Section II and then discuss the concepts of modularity and interoperability in detail in Section III. In Section IV, we will present our Semantic Description and Interoperability (SDI) framework. Next, using an example of an assistance system consisting of a weighing module and an eye-tracking module, we will discuss how to achieve interoperability across CPS modules in Section V. Finally, we will discuss the implementation of an assistance system using our framework in Section VI, and conclude with potential future work.

II. Related Work

Our work brings together two different areas of research: development of CPS for production as well as the semantic design of these systems. We will discuss related work in both areas separately and will discuss some aspects in detail.

Assistance Systems. There is significant contemporary research interest in using sensor technology for developing context-aware CPS [1], [5]–[7]. Nelles et al. have proposed assistance systems for planning and control in production environment [5]. Gorecky et al. have also explored cognitive assistance and training systems for workers during production [6]. Zamfiresu et al. have also integrated virtual reality and a hand-tracking module to help workers during assembly processes [7]. However, they do not consider the modular design of these modules and interoperability between such modules. Very recently, Quint et al. have looked into the architecture of such a system as composed of a central system and modules which can handle heterogeneous data [1]. However, they do not explore standardizing the design of such modules. In our work, we provide the SDI framework for designing CPS modules and an ecosystem for ensuring interoperability across these modules.

Semantic Design. Wahlster et al. [8] describe the importance of semantic technologies in production systems. Grangel

[1]Amita Singh is with Technische Universität Kaiserslautern, Germany singh@rhrk.uni-kl.de

[2]Fabian Quint and Martin Ruskowski are with German Research Institute for Artificial Intelligence (DFKI GmbH), Kaiserslautern, Germany mail@fabian-quint.de, martin.ruskowski@dfki.de

[3]Patrick Bertram is with Technologie-Initiative SmartFactoryKL e.V., Kaiserslautern, Germany bertram@smartfactory.de

et al. [3] discuss semantic web technologies in handling heterogeneous data from distributed sources using light-weight vocabulary. Semy et al. [9] described ontologies as the key enabler for building pervasive context-aware system wherein independently developed devices and softwares can share contextual knowledge among themselves. Recently, Negri et al. [10] also discussed requirements and languages of semantic representation of manufacturing systems and conclude that ontology are the best way of such representation in the domain.

In his seminal work, Nocola Guarino formally defined ontologies both as a tool for knowledge representation and management as well as a database for information extraction and retrieval [11]. In particular, he describes how ontologies can play a significant role both during development and run-time for information systems. Negri et al. [10] highlighted importance of ontologies in providing intuitive and human-readable format, standardization not only in terms of definitions and axioms using IRIs but also standardizing web-services and message-based communication which not only makes engineering of the system streamlined but also facilitates interoperability between parts of the system.

Further, Niles et al. [12] highlight the usefulness of upper ontologies in facilitating interoperability between domain-specific ontologies by the virtue of shared globally unique terms and definitions (IRIs) in a top-down approach of building a system. Semy et al. [9] also describe mid-level ontologies as a bridge between upper ontologies and domain-specific ontologies which encompass terms and definitions used across many domains but do not qualify as key concepts. Furthermore, Sowa et al. [13] discuss ontology integration and conflicts of data in the process. They conclude that ontology merge is the best way of ontology integration as it preserves complete ontologies while collecting data from different parts of the system into a coherent format.

In the remainder of the paper unless otherwise stated we will follow the definition of ontologies and standards as given by W3C [14].

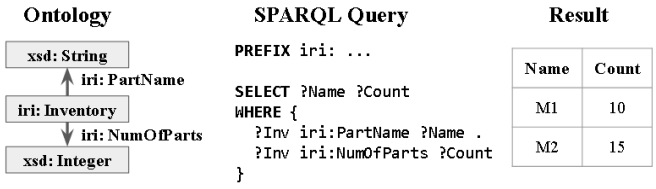


Fig. 1: An example of ontology definitions and relations, SPARQL query and results.

Ontologies. In this section, we will briefly discuss some features of ontologies which will be relevant for our design. Ontologies are used to explicitly define entities and relations between entities. Figure 1 shows an example of a very small ontology. Ontologies provide unique global addresses, called Internationalized Resource Identifiers (IRIs), to all entities and relations. Hence, with the virtue of IRIs, entities and relations can be referred to easily within and outside the system. Ontologies can also be *imported*, which is how definitions of entities and their relationships can be re-used during development time. This feature will be key in creating an ecosystem of ontologies in Section V. During run-time, *individuals* of the entities along with their relationships with each other are created.

To know more about ontologies, we direct the reader to the W3C standards [14]. The features we have described will be essential while designing and implementing our proposed SDI framework.

III. Modular Design and Interoperability

The assistance system should be designed to be flexible, such that it should be possible to combine different CPS with very varying capabilities without the human to conduct extensive configuration. This flexible design will make it possible to scale the intelligence of the overall system by adding/removing CPS. We will consider each CPS as a module which feeds information to a central decision-making entity, which we will call the central system as shown in Figure 2.

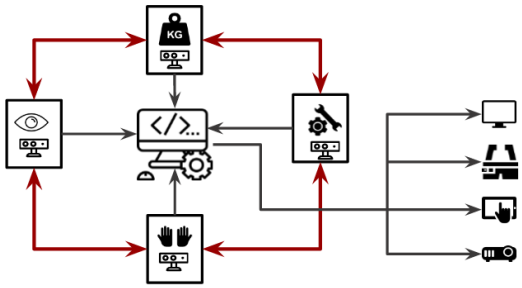


Fig. 2: Schematic description of an assistance system. It has a central system and four CPS modules. These modules feed data to the central system. Based on the data provided to the central system, it helps the worker by sequentially communicating next steps in the process using visualization devices. Black lines indicate data sent to the central system and red lines indicate interoperability across modules.

A modular design enforces separation of concerns: the central system will only rely on the information *provided* by the modules. As per the traditional modular design, the internal state of the modules, i.e. the implementation details would ideally be made completely opaque and inaccessible to the central system and other modules. In contrast, we propose a framework for designing the modules using ontologies which will allow the modules to access and use information from each other.

There are several hurdles which we will need to cross before we can allow for such interoperability and, as we shall show, these can be overcome by semantically annotating the information in each module using ontologies. As discussed in the previous section, an outright advantage of using ontologies is that they can give a unique name, i.e., IRIs, to *each* piece of information in the complete system and make it immediately accessible using a simple declarative querying language (SPARQL) as shown in Figure 1 [15].

Additionally, we will get other advantages which come naturally with using ontologies for free, viz expressive self-documentation, automatic reasoning using description logic, etc.

Using ontologies as our tool of choice, we will now consider the following two questions:

(i) **How to design and semantically annotate a CPS module?** We will answer this question in Section IV.
(ii) **How to share information using ontologies?** We will address this issue in Section V.

Remark Note that the decision-making algorithm in the central system should be in such an abstract way that it does not need to be adapted to accommodate the underlying frequently changing CPS modules, i.e., the assistance system should be able to function without all modules being attached to the system and the modules should be plug-and-play. However, the problem of designing the algorithm is out of scope of this work.

IV. Framework for designing a CPS

In this section, we will propose a framework for designing a CPS module and its ontology. We will start with *what* we want to achieve by adding a particular CPS to the system, and then determine its boundary, or scope, with respect to the central system. Next, we will have to make decisions about the *intelligence* of the system, which will influence the hardware choices we make. Finally, we will create a bottom up ontology of a CPS and describe how to integrate it with the central system ontology.

Requirements. At the outset, it is important to understand *why* a CPS module is required. This decision will determine the metric we use for measuring the effectiveness of a module finally. This objective may range from very general, e.g. "increasing the efficiency of a factory", to very specific, e.g. "decreasing the number of errors for a particular assembly station". Keeping the requirements as specific as possible will help with the next step of our design.

System Boundary. With an objective in mind, we will need to translate it to a concrete piece of information that the central system needs from the CPS. An analogy can be drawn between the information which the central system needs and the idea of *sufficient statistic*: the information should be *sufficient* for the central system to arrive at its objective. This information will be the *interacting variable* between the CPS module and the central system. In terms of ontologies, the interacting variable will need to have the same IRI in both the central system ontology as well as the ontology of the CPS module. How this can be ensured will be discussed in Section V.

CPS Intelligence. Once we know the system boundary, i.e. the interacting variable for a CPS module, it is necessary for the CPS to be *intelligent* enough to calculate this information from raw sensor readings. This *intelligence* is manifested in the accuracy/update frequency of sensors and the computational power afforded by the hardware used to create the CPS module. Calculation of the value of the interacting variable

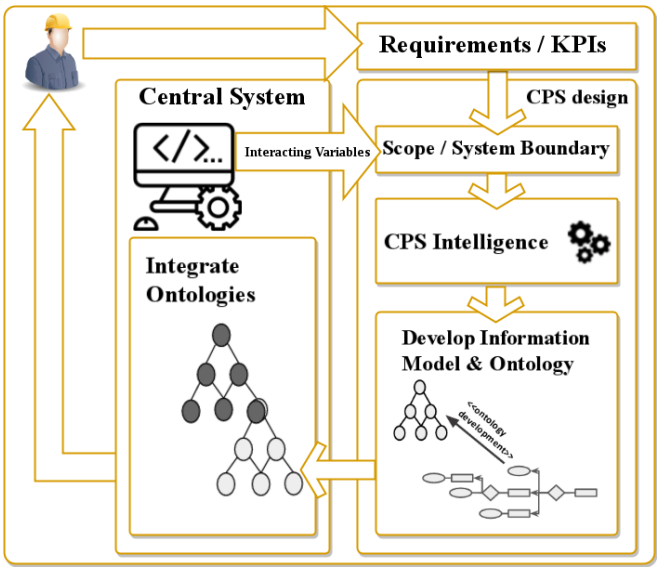


Fig. 3: Schematic description of the SDI framework for designing a CPS.

effectively sets a lower bound on this system intelligence, i.e. a CPS should be able to process the data received through sensors to communicate the interacting variable whenever it is needed by the central system. We can improve the system intelligence further by using more sophisticated hardware and/or applying better algorithms while processing data, which improve the *quality* of the values calculated by the CPS module for the interacting variable.

Another thing to note is that the CPS module should have the computational power to use ontologies during run-time. However, the restrictions placed by this requirement are mild because ontologies can be made light-weight during run-time [3].

Developing information model & ontology. After deciding on the hardware we will use for the module, we start by creating an *information model*, i.e. an abstraction of the physical layer created by listing all the physical entities present in a module: it lists all sensors, computational units, communication units and relations between them [16].

We will develop the ontology of the CPS using the information model as a reference, as illustrated in Figure 4. In addition to the entities defined in the information model, the ontology may also contain variables which are the result of *processing* the data gathered by sensors. Finally, we will add the the interacting variable, which we had arrived at while determining the system boundary, to the ontology with appropriate relationships with other entities.

Ontology Integration. In the final step, ontology of the CPS module will be merged with the central system ontology. The central system will use the interacting variable for making its own decisions, but will also act as a database for the complete assistance system during run-time. The modules, hence, will be able to inquire the central system for not only the interacting variables of other modules, but also about the

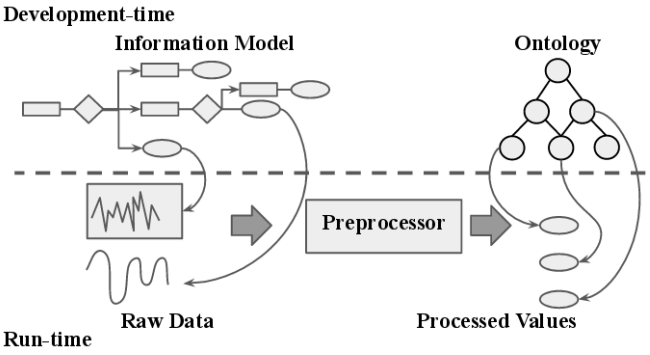


Fig. 4: The information model (top-left), which is based on the physical setup of the system, is used to design the ontology (top-right) during the development-time. During implementation of the model, in the run-time, the sensors in the information model will produce raw data. This raw data will be pre-processed by the CPS module (this is where system-intelligence comes into play) and will be made ready for the ontology.

internal entities, which the central system does not explicitly use. In the Section V, we will address the problem of how can the CPS modules be made aware of the various entities which *may* be accessed.

V. Ontology Sharing

In the previous section, we saw how to design a module. However, the benefit of semantic design will be observed when these ontologies are shared and understood by other parts of the system, i.e. CPS modules and central system. Firstly, for ease of exposition, we will assume that the central system can answer queries sent to it in SPARQL. In a real implementation, such queries can be written in a different language or may be provided using an alternate API. However, the use of unique IRIs to refer to entities in the ontologies will be crucial to facilitate interoperability in all implementations.

Our proposed design for assistance system has one ontology for each CPS module and an ontology for the central system. There also are certain [*interacting variables*] which are shared between the central system and the modules. We propose to abstract these shared interacting variables, and their corresponding constraints, to an "upper ontology", which we will hereby refer to as the assistance system ontology. Both the central system ontology and the ontology of the modules will `<<import>>` the assistance system ontology, thereby ensuring that the definitions of the interacting variables, i.e. their IRI, are the same. The assistance system ontology is kept minimal and is only extended when there are new interacting variables, i.e. when a new potential CPS module is identified which can aid the intelligence of the assistance system.

However, the modules may be collecting and reporting values beyond the interacting variables. These are values which the central system does not need, but which other CPS modules can use. In the remainder of this section, we will discuss how such values can be shared across modules.

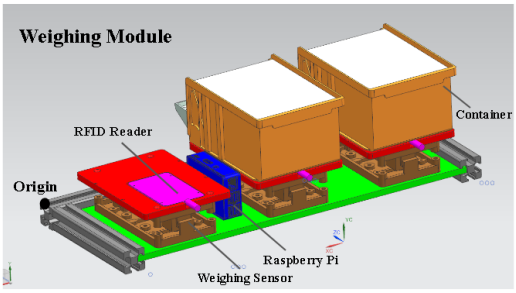


Fig. 5: Schematic description of the weighing module. It has three weighing sensors, each with an RFID reader and a container, and a Raspberry Pi (RPi). Weighing sensors and RFID readers provide raw data to the RPi which then processes the data and sends it to the central system (not shown in the figure). [Source: DFKI]

In more detail, we will take an example inspired by Figure 2. We will take a weighing module and an eye-tracking module as examples for demonstrating data sharing between CPS modules, which is especially important if the modules are being developed by two independent third-party vendors.

Before we discuss data sharing between the aforementioned modules, let us first define the constituents of a weighing module. As shown in Figure 5, the considered weighing module consists of three weighing sensors, each with an RFID reader, a container kept on each sensor, and a Raspberry Pi. RFID readers are used to read the part information stored in RFID tags attached to the containers, which contain the corresponding parts. A reasonable assumption is that the weighing module needs to report the part name and total number of parts corresponding to each container to the central system. Therefore, these data properties, `hasNumOfParts` and `hasPartName`, will be the interacting variables of the weighing module and will be defined in the assistance system ontology. Additionally, the ontology of the weighing module will contain definitions which are required by the weighing module itself, but not the central system. We assume that the weighing module ontology contains the relative positions `(x, y, z)` of the containers with respect to the Origin shown in the Figure 5.

Now, the eye-tracking module can use the information about the position of containers kept on the weighing module to calibrate itself. We will describe two different ways of making such sharing of knowledge across modules possible.

Decentralized Organizational Scheme. Figure 6 shows the hierarchy among the ontologies for a completely decentralized system. To facilitate interoperability of data between modules, ontologies of both interacting module should have definitions, i.e., the IRIs, and axioms of entities pertaining to the required data, i.e., the positions of the containers. We have assumed that this information was mapped in the weighing module ontology as data property `(x, y, z)` of

the class `Container`.

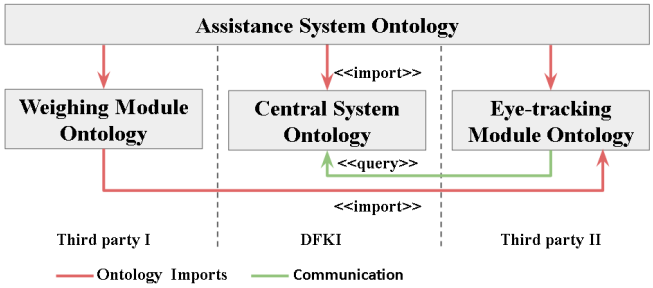


Fig. 6: Schematic description of decentralized organization. In this scheme, the central system and modules' ontologies import the assistance system ontology. To access entities of weighing module, an eye-tracking module will first need to import the weighing module ontology to obtain the definitions of the required entities.

Once the eye-tracking module `<<import>>`s the ontology of the weighing module during development time, it can compose a SPARQL query with the correct IRIs to query the central system during run-time for the values. If the weighing module attached to the central system has indeed reported the `(x, y, z)` coordinate values, they will be returned to the eye-tracking module. This method, hence, allows any two teams to communicate and to mutually agree on how best they can share data and promotes rapid prototyping.

However, such a setup has the disadvantage that independent teams may reinvent properties which will have unique IRI (e.g. `TeamA:hascoordinateX`, `TeamB:hasX` and `TeamC:hasPositionX`) but with the same semantic meaning. This would complicate interoperability across modules because for the same information from weighing modules developed by N different parties, the eye-tracking module will have to make N queries. The next scheme we propose addresses these short-comings.

Centralized Organizational Scheme. Under this scheme, we trade-off complete decentralization in favour of a W3C committee like setup [14] where a formal specification (i.e. a mid-level ontology) is maintained by a committee which consists of all stakeholders. Figure 7 shows the hierarchy among ontologies for the Centralized Organizational Scheme. The key feature of this scheme is the existence of a mid-level ontology which imports the assistance system ontology. Additionally, the mid-level ontology describes the entities of interest for *all* other modules.

The idea behind creating a mid-level ontology is to create a repository of all relevant entities described in all CPS modules. This simplifies the search by engineers for variables provided by other modules. In contrast to the minimal nature of the assistance system ontology, the mid-level ontology will be larger as it will contain *optional* variables which may be reported by the modules to complement the interacting variables.

Discussing the previous example in the context of new scheme, if the position of containers is considered to be of common interest, it will be added to the mid-level ontology as variables which may be reported by weighing modules. This will *standardize* the way different weighing modules report the position of their containers to the central system. Now, the eye-tracking module no longer needs to import N different ontologies for constructing its SPARQL queries. Instead, it can construct a query using the standardized IRIs in the mid-level ontologies.

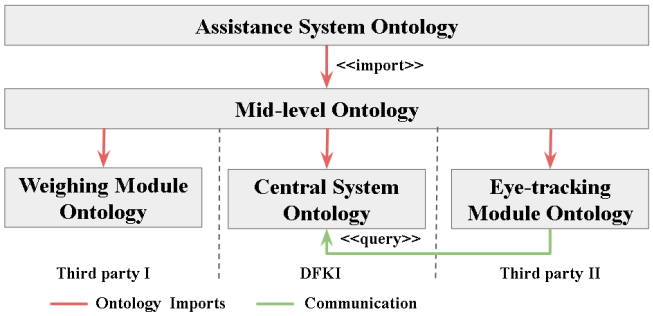


Fig. 7: Schematic description of centralized organization. In this scheme, mid-level ontology imports the assistance system ontology. Module ontologies import mid-level ontology. Entities of one module required by other modules are defined in the mid-level ontology. Hence, the eye-tracking module can directly access the weighing module entities.

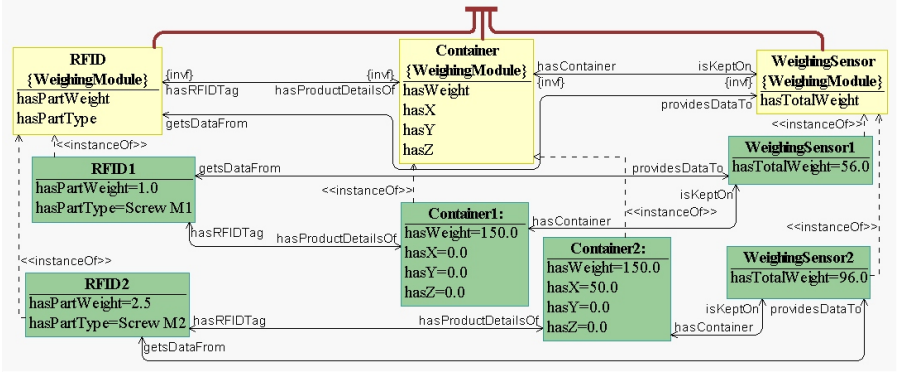
In comparison with the more decentralized structure given in the previous section, the benefits and costs of this design are apparent. First of all, reinvention of entities is prevented as a cursory check through the mid-level ontology will show whether standardized properties already exist for the module being developed. Further, each module can individually extend the entities imported from mid-level ontology. These extensions are local and are not propagated to the mid-level ontology. However, if such an extension is considered to be of common interest, it can be proposed as an addition to the mid-level ontology.

VI. Implementation

In the previous section, we discussed how to design ontologies during development phase. In this section, we will make the ideas more concrete by showing an example of how to *populate* and *query* ontologies during run-time. We will only give a high level description of how the data exchange. First, we assume that the ontologies of the assistance system, central system, weighing module and eye-tracking module have been developed using the SDI framework. An example implementation is available for download on GitHub[1].

Now consider the state of the assistance system when a weighing module is attached to the central system. The weighing module will create *individuals* (shown in green) to *populate* its ontology as shown in Figure 8a. The resulting ontology will contain individuals corresponding to `Weighing Sensor`, `RFID` and `Container` and data properties representing the state of the individuals. Now the

[1]github.com/AmitaChauhan/SDI-Framework



(a) Schematic description of populating an ontology for weighing module. Similarly, other ontologies are populated during run-time by creating individuals.

```
SPARQL query:
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX WM: <http://www.semanticweb.org/amita/WeighingModule#>
SELECT ?Container ?X ?Y ?Z
        WHERE { ?Container WM:hasX ?X .
                ?Container WM:hasY ?Y .
                ?Container WM:hasZ ?Z }
```

Results

Container	X	Y	Z
Container2	"50.0"^^	"0.0"^^	"0.0"^^
Container1	"0.0"^^	"0.0"^^	"0.0"^^

(b) An example of query that the eye-tracking module will make to access data of the weighing module. The eye-tracking ontology gets definitions by $<< import >>$-ing the weighing module ontology. Here prefix WM is used for the weighing module namespace.

Fig. 8: Description of partial implementation of the SDI framework in Protégé. We have not shown ontologies of the assistance system, central system, eye-tracking module and mid-level for the ease of exposition. Classes are shown using yellow boxes whereas individuals are shown using green boxes. All relations between entities are labelled. An example of a SPARQL query which can be used during run-time is also shown.

weighing module will send its ontology (i.e. individuals) to the central system for a *merge*. After the merge, the central system will act as a database which can answer SPARQL queries. For example, the eye-tracking module may query the central system for the `(x, y, z)` coordinates of the containers using a SPARQL query shown in Figure 8b.

When the central system requests an update (event trigger, timer trigger, etc. [1]) the weighing module will send its ontology or only the values which have changed to the central system which, in turn, update its ontology.

VII. Summary & Future Work

In this paper, we proposed the Semantic Description and Interoperability (SDI) framework for designing CPS for an assistance system. The framework presents the principles for semantic description of a CPS with the aim of facilitating interoperability across modules. Then we discussed two different sharing schemes for creating an ecosystem of ontologies. Thereby, the capabilities of the assistance system can be adapted to the needs of the worker and the task. We used the SDI framework for the design of a weighing module in Protégé and demonstrated interoperability between two CPS, viz. a weighing module and an eye-tracking module.

However, the SDI framework can be extended to design CPS in general: we limited our discussion to designing a CPS for an assistance system for ease of both exposition and demonstration. Moreover, the ecosystem of ontologies discussed can be used to maintain knowledge bases in different domains.

The next important step in the development of assistance systems is to develop a plug-and-play algorithm for CPS modules as alluded to in Section III. This paper lays the foundation to allow for design of plug-and-play CPS modules and, with the SDI framework, designing and implementing the plug-and-play algorithm is relatively straightforward.

Another important step to make the system deployable would be to create global standards: either by defining design and communication standards specific to assistance systems, or by investigating the suitability of existing standards, e.g. RAMI.

References

[1] F. Quint, F. Loch, M. Orfgen, and D. Zuehlke, "A system architecture for assistance in manual tasks." in *Intelligent Environments*, 2016.

[2] E. Tantik and R. Anderl, "Integrated data model and structure for the asset administration shell in industrie 4.0," *Procedia CIRP*, 2017.

[3] I. Grangel-González, L. Halilaj, G. Coskun, S. Auer, D. Collarana, and M. Hoffmeister, "Towards a semantic administrative shell for industry 4.0 components," in *IEEE 10th International Conference on Semantic Computing (ICSC)*, 2016.

[4] V. Gezer, J. Um, and M. Ruskowski, "An extensible edge computing architecture: Definition, requirements and enablers."

[5] J. Nelles, S. Kuz, A. Mertens, and C. M. Schlick, "Human-centered design of assistance systems for production planning and control: The role of the human in industry 4.0," in *IEEE International Conference on Industrial Technology (ICIT)*, 2016.

[6] D. Gorecky, S. F. Worgan, and G. Meixner, "Cognito: a cognitive assistance and training system for manual tasks in industry," in *29th Annual European Conference on Cognitive Ergonomics*, 2011.

[7] C.-B. Zamfirescu, B.-C. Pirvu, D. Gorecky, and H. Chakravarthy, "Human-centred assembly: a case study for an anthropocentric cyber-physical system," *Procedia Technology*, 2014.

[8] W. Wahlster, "Semantic technologies for mass customization," in *Towards the Internet of Services: The THESEUS Research Program*, 2014.

[9] S. K. Semy, M. K. Pulvermacher, and L. J. Obrst, "Toward the use of an upper ontology for us government and us military domains: An evaluation," MITRE CORP BEDFORD MA, Tech. Rep., 2004.

[10] E. Negri, L. Fumagalli, M. Garetti, and L. Tanca, "Requirements and languages for the semantic representation of manufacturing systems," *Computers in Industry*, vol. 81, pp. 55–66, 2016.

[11] N. Guarino, *Formal ontology in information systems: Proceedings of the first international conference (FOIS)*, 1998.

[12] I. Niles and A. Pease, "Origins of the ieee standard upper ontology," in *IJCAI workshop on the IEEE standard upper ontology*, 2001.

[13] J. F. Sowa, "Building, sharing, and merging ontologies," *web site: http://www.jfsowa.com/ontology/ontoshar.htm*, 2001.

[14] S. Bechhofer, "Owl: Web ontology language," in *Encyclopedia of database systems*, 2009.

[15] E. Prud and A. Seaborne, "Sparql query language for rdf," 2006.

[16] P. P.-S. Chen, "The entity-relationship modeltoward a unified view of data," in *Readings in artificial intelligence and databases*, 1988.