A Framework for Semantic Description and Interoperability across Cyber-Physical Systems

Amita Singh
Technical University
Kaiserslautern
Email: amitas@kth.se

Fabian Quint
German Research Center
for Artificial Intelligence (DFKI)
Email: mail@fabian-quint.de

Patrick Bertram
Technologie-Initiative
SmartFactoryKL e.V.
Email: bertram@smartfactory.de

Martin Ruskowski
German Research Center
for Artificial Intelligence (DFKI)
Email: martin.ruskowski@dfki.de

***Abstract*—With the advent of Industry 4.0 and human-in-the-loop paradigms, Cyber-Physical Systems (CPS) are becoming increasingly common in production facilities, and, consequently, there has been a surge of interest in the field. In production systems, CPS, which assist humans in completing tasks are called assistance systems. Most recent designs proposed for assistance systems in the production domain are monolithic and allow only limited modifications. In contrast, this work considers an assistance system to have a hybrid architecture consisting of a central entity containing the process description (or instructions) and one or more plug-and-play Cyber-Physical Systems to retrieve relevant information from the physical environment. Such a design allows the overall system capabilities to be adapted to the needs of workers and tasks. In this paper, a framework is presented for designing the CPS modules using Semantic Web technologies, which will allow (i) interpretation of all data, and (ii) interoperability among the modules, from the very outset. Furthermore, a knowledge description model and ontology development of a CPS module is described. Two different models of maintaining ontologies and the ecosystem are described along with their advantages and disadvantages. An approach is illustrated with the help of a use case for implementing the framework to design a module, data exchange among modules, and to build a sustainable ecosystem of ontologies, which enables rapid development of third-party CPS modules. An implementation is provided discussing hardware, software and communication design of such a module and future direction of research is discussed.**

Keywords*—*human-centered CPS; assistance systems; adaptive automation; ontology; interoperability.

I. Introduction

An ever growing catalogue of products, [1] 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 [2]–[4]. According to the European Factories of the Future Research Association, human-centricity is a prerequisite for the production systems to be flexible and adapt to the changing demographics [5][6]. Thus, major efforts are being made to make adaptive human-centered CPS (H-CPS) where machines and automation adapt to the physical and cognitive needs of humans in a dynamic fashion [7][8].

In this paper, assistance systems are considered as H-CPS in production systems. Assistance systems assess the production process using sensors embedded in the environment and, based on the state of the process, provide instructions to workers through visualisation devices attached to them [9]. 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, Service-Oriented Architecture (SOA), interoperability by the virtue of common semantic description (e.g., administrative shell [10][11]), and edge-computing [12] are rarely applied to H-CPS.

In this paper, a CPS-based assistance system, which adapts to a worker's need by exploiting the benefits of such techniques is proposed. Such an assistance system has a central system and one or more CPS modules attached to it as shown in Figure 1. CPS modules feed information extracted from the environment to the central system. The central system, in turn, processes this information to assess the state of the process and the worker's needs.

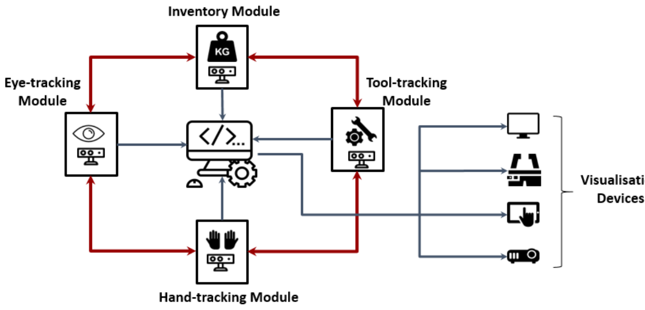


Figure 1. Schematic description of an assistance system.

To the best of the authors' knowledge, no design so far allows one module to access and use the data from other modules. In this paper, semantic design of modules and interoperability between different parts of an assistance system are discussed in detail, and consequently, a Semantic Description and Interoperability (SDI) framework is proposed.

In the remainder of the paper, first the state-of-the-art and related work is presented in Section II and then the concepts of modularity and interoperability are discussed in detail in Section III. Next, Section IV discusses the SDI framework for design of modules and Section V further explains ecosystem of ontologies. In Section VI, the development of such modules is discussed. Finally, the implementation of an assistance system is simulated using the proposed framework in Section VII, followed by the conclusion and potential future work.

II. Related Work

The vision of Computers in Manufacturing (CIM) of creating completely automated factories could not be realized due to the complexity involved in production processes [13]. The effort to implement CIM made it clear to engineers that completely automated factory is not a plausible solution as per the state-of-the-art. Humans are an indispensable part of production systems but automation at different stages of product is a necessity and a practical approach to the problems of increasing product variants, reducing product life-cycle and rising labour costs [5]. Thus, CIM established that it is important that the CPS systems developed should help humans instead of trying to replace them because with the current state of the technology it is difficult to replicate human cognitive skills. This has lead to human-centric workplaces and the approach was coined human-in-the-loop.

Human-system interaction is an indispensable part of the production systems and acts as an enabler of the intelligent decision making process [14]. In complex production scenarios, the symbiotic man-machine systems are the optimal solution. This change in the nature of human-machine led to the paradigm shift from independently automated and manual processes towards a human-automation symbiosis called human cyber-physical systems. These systems are characterized by collaborative effort of workers and machines and aim at assisting workers being more efficient and effective [5]. These systems are based on a trusting and interaction-based relationship, which has human supervisory control and human situation awareness, leading to adaptive automation to improve knowledge of worker and help the process.

Production facilities are focusing on cyber-physical systems (CPSs) that can interact with human through many modalities. CPSs are a combination of interacting embedded computers and physical components. Both computation and physical processes work in parallel to bring about the desired output. Computers usually monitor the physical processes via sensors in real-time and provide feedback to actuators [15], [16]. A CPS consists of one or more micro-controllers to control sensors and actuators which are necessary to collect data from and interact with its environment. These systems also need communication interface to exchange data with other smart devices and a cloud. According to Jazdi [16], data exchange is the most important feature of cyber physical systems. CPSs connected over internet are also known as Internet-of-Things.

Its vision is to bring automation in the field of production and help combat the problems of increasing catalogue and labour costs. The information and communication technologies have trickled their way down to the production systems, paving the way for monolithic production systems to become modular and have decentralized control architectures. It is one of the most significant directions in computer science, information & communication technologies and manufacturing technologies. With the increasing sophistication of actuators and sensors available in the market, availability of data has increased many folds. The CPSs used to create flexible and re-configurable production systems called Cyber-Physical Production Systems (CPPSs). CPPSs are build on the principle of modularity and decentralized control. Thus, these modules are loosely coupled with each other.

Neither traditional nor fully automated systems can respond effectively and efficiently to dynamic changes in the system [17]. Hence, workers should be assisted, as needed, in their work, thus, including automation with human aptitude as a trouble shooter. Manual assembly stations with assistance systems are developed based on this concept. These stations are modular units, one in the chain of many automated/semi-automated stations. Products are assembled by workers at each station. Usually, production facilities are one piece flow. Depending upon the assembly plan of a product, one or more processes can be performed on a station.

This work brings together two different areas of research: development of CPS for production, as well as the semantic design of these systems. Related work in both areas are discussed separately and some aspects are discussed in detail.

A. Assistance Systems

Production facilities are focusing on CPS that can interact with human through many modalities. There is significant contemporary research interest in using sensor technology for developing context-aware CPS [9][18]–[20]. Nelles et al. have looked into assistance systems for planning and control in production environment [18]. Gorecky et al. have explored cognitive assistance and training systems for workers during production [19]. Pirvu et al. [21] talk about the engineering insights of human centered, yet highly automated, cyber-physical system while (i) keeping in mind adaptive control, (ii) cognitive assistance, and (iii) training in manual industrial assembly. The aim of such a system is to design a mobile, personal assembly work station which assists the worker in task solving in real time while understanding and inducing work-flows. Standardized abstractions and architectures help the engineers in the design phase by reducing the complexity involved in building such systems [22]. Zamfirescu et al. have also integrated virtual reality and a hand-tracking module to help workers during assembly processes [20]. Figure 1 shows a schematic description of such an assembly station. These stations are equipped with different visualization techniques and sensor technologies. Visualization techniques, like projectors and smart glasses as shown in Figure 1, help workers during assembly process by displaying instructions. Interactive screens can also be installed at assembly stations using which workers can interact with stationery computers when required. Assembly stations have areas dedicated for storing tools and parts used during assembly. Sensors can be employed to track usage of tools and control inventory of parts. RFID readers are installed on products and to know the current status of products in addition to the tools and parts as in the traditional workstations.

Assistance system derives its principles from the Operator 4.0 principle where workers are provided machines to aid their work. It helps workers by reading the product status available with products in machine readable format, collecting other information about the environment and helping the worker to decide the next step to be taken based on the information it receives. Hence, this system can be seen as a context aware human-centric cyber-physical system. As shown in Figure 1, this system consists of a central system and one or more CPS modules.

These CPS modules are built on the principle of plug-and-produce. The analogy is drawn from plug-and-play concept in computer science [23]. Plug-and-produce means a smart device can be easily added or removed, replaced without disrupting functioning of the system. The system should continue working while a CPS module is being added or removed. Additionally, the system should be able to recognize the newly added CPS. This process is different from the traditional processes in which systems need to be reprogrammed and machines are stopped for reconfiguration. Time taken in the complete process is counted as downtime. Similarly, in case of plug-and-produce systems maintenance can be done by removing only the required CPS module while the complete system continues working.

For this purpose, each CPS module should have its own environmental information and it should provide this information to the system to which it is being attached [23]. This gives central system the leeway to reconfigure and requires CPS modules to be smart and adaptive which demands CPS modules to have certain level of intelligence.

Very recently, Quint et al. have proposed a hybrid architecture of such a system, which is composed of a central system and modules which can handle heterogeneous data [9]. However, they do not explore standardizing the design of such modules. In this work, a framework for designing CPS modules and an ecosystem for ensuring interoperability across these modules is proposed.

B. Semantic Design

The Semantic Web is an extension of World Wide Web that promotes common data formats and exchange protocols on the Web through standards. Wahlster et al. [24][25] use Semantic Web technologies to represent and integrate industrial data in a generic way. Grangel et al. [11] discuss Semantic Web technologies in handling heterogeneous data from distributed sources using light-weight vocabulary. Semy et al. [26] describe these technologies as the key enabler for building pervasive context-aware system wherein independently developed devices and softwares can share contextual knowledge among themselves.

Recently, there have been some efforts towards discussing the need of bringing more semantics and data-driven approaches to Industry 4.0. Cheng et al. [27] identify varying degree of semantic approach and further provide guidelines to engineers to select appropriate semantic degree for different Industry 4.0 projects. Wahlster et al. [24] talk about the importance of semantic technologies in mass production of smart products, smart data and smart services. Semantic service matchmaking in cyber-physical production systems is presented as a key enabler of the disruptive change in the production logic for Industry 4.0. Obitko et al. [28] introduce the application of semantic web technologies in handling large volumes of heterogeneous data from distributed sources. Grangel et al. [11] describe an approach to semantically represent information about smart devices. The approach is based on structuring the information using an extensible and light-weight vocabulary aiming to capture all relevant information. Semantic Web technology formalisms, such as Resource Description Framework (RDF), RDF Schema and Web Ontology Language (OWL), help solve the major hurdle towards description and interoperability between CPS by annotating the entities of a system. Some of the major advantages of using RDF-based semantic knowledge representation are briefly discussed here:

Global unique identification. Semantic Web describes each entity within a CPS and its relations as a global unique identifier. According to the principles of Semantic Web, HTTP URIs/IRIs should be used as the global unique identifiers [29]. This ensures disambiguation, and retrieval, of entities in the complete system. As a consequence, a decentralised, holistic and global unique retrievable scheme of CPS can be established.

Interoperability. Interoperability is the ability to communicate and interconnect CPS from different vendors. It is vital in order to have cost effective rapid development. According to domain experts [11][25][30], RDF and Linked Data are proven Semantic Web technologies for integrating different types of data. Gezer et al. [31] mention that OWL-S ensures better interoperability by allowing services to exchange data and allowing devices to configure themselves.

Apart from the above mentioned advantages, by using RDF representation different data serialization formats, for example RDF/XML, RDF/OWL can be easily generated and transmitted over the network [11]. Further, data can be made available through a standard interface using SPARQL, a W3C recommendation for RDF query language [32].

Recently, Negri et al. [33] discussed requirements and languages of semantic representation of manufacturing systems and conclude that ontologies are the best way of such representations in the domain. The authors also highlighted importance of ontologies in providing system description in an intuitive and human-readable format, standardization not only in terms of definitions and axioms, but also standardizing Web-services and message-based communication. This not only makes engineering of the system streamlined but also facilitates interoperability between parts of the system. 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 [34]. In particular, he describes how ontologies can play a significant role during development, as well as run-time, for information systems.

Further, Niles et al. [35] highlighted the usefulness of upper ontologies in facilitating interoperability between domain-specific ontologies by the virtue of shared globally unique terms and definitions (HTTP URIs/IRIs) in a top-down approach of building a system. Semy et al. [26] also described 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. [36] discussed 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, the definition of ontologies and standards as given by W3C [32] are followed.

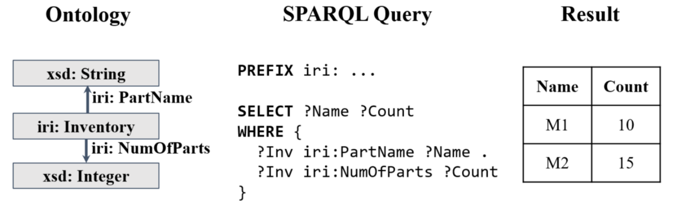
C. Ontologies

In computer science, ontologies were developed for the Semantic web. The aim of Semantic web is to help software agents interact and share information over the internet. This is done by encoding the data in a machine interpretable language using constraints defined in the domain ontology. This lets software agents locate resources to extract and use information on the web. This differentiates ontologies from other traditional languages, like UML and SysML, used to describe software structure.

Ontologies conceptualise a domain by capturing its structure. In this section, some features of ontologies, which are relevant for the proposed design are discussed. Ontologies are used to explicitly define entities and relations between entities. Figure 2 shows an example of a small ontology, an associated SPARQL query language, and query results obtained during a run-time. Ontologies provide unique global addresses to all entities and relations using HTTP URIs/IRIs. Hence, with the virtue of HTTP URIs/IRIs, entities and relations can be referred to easily from 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, as shown in the work later, is crucial in creating an ecosystem of ontologies. During run-time, *individuals* of the entities along with their relationships with each other are created.

In the remainder of the section, important features relevant for this work are discussed:

Upper ontologies are high-level, domain-independent ontologies, providing a framework by which disparate systems may utilize a



```
PREFIX iri: ...

SELECT ?Name ?Count
WHERE {
  ?Inv iri:PartName ?Name .
  ?Inv iri:NumOfParts ?Count
}
```

Name	Count
M1	10
M2	15

Figure 2. An example of ontology definitions and relations, SPARQL query and results.

common knowledge base and from which more domain-specific ontologies may be derived [26]. Thus, upper ontologies facilitate interoperability between domain-specific ontologies by the virtue of shared common terms and definitions [37]. They contain definitions and axioms for common terms that are applicable across multiple domains and provide principled forms of conceptual inter-linkage between data [38]. Thus, provide semantic integration of domain ontologies.

On the other hand, **domain ontologies** have specific concepts particular to a domain and represent these concepts and their relationships from a domain-perspective. Multiple domains can have the same concept but their representation may vary due to different domain contexts. Domain ontologies inherit the semantic richness and logic by importing upper ontologies.

Another important feature of upper ontology is the structure that they impose on the ensuing ontologies: they promote modularity, extensibility, and flexibility. According to Semy et al. [26], upper ontologies can be built using two approaches: top-down and bottom-up. They discuss benefits and limitations of both approaches. In a top-down approach domain ontology uses the upper ontology as the theoretical framework and the foundation for deriving concepts [26]. In a bottom-up approach, new or existing domain ontologies are mapped to an upper ontology. This approach also benefits from the semantic knowledge of upper ontology but the mapping can be more challenging as inconsistencies may exist between the two ontologies. For example, two teams may have different vocabulary for a similar semantic variable. In this case, mapping the two ontologies to an upper ontology would have inconsistencies. These inconsistencies are resolved as and when needed. However, usually a combination of both approaches is used to design upper ontologies.

The solution proposed to the problem of interoperability across modules relies heavily on the idea of upper ontologies. Upper ontology starts with defining a set of high level entities and then successively adding new content under these entities [35]. The solution incorporates both the top-down and bottom-up approaches. Depending on the need entities are added to the high level ontology.

Mid-level ontologies act as a bridge between basic vocabulary described in the upper ontology and domain-specific low-level ontology. This category of ontologies may also encompass terms and definitions used across many domains.

Ontology development can be seen as defining structure, constraints and data for other programs to use. Software agents and other problem solving methods can use these ontologies as ready-made data that can be fed to the program in order to understand the vocabulary and basic principles of the domain. The independently developed ontologies need to join to exchange data.

Ontology integration is the process of finding commonalities between two ontologies, for example Ontology A and ontology B, and a third ontology C is derived from it. This new ontology C facilitates interoperability between software agents based on ontologies A and B. The new ontology C may replace the old ontologies or may be used as only an intermediary between systems based on ontologies A and B are merged in a third ontology C. Ontologies can be integrated primarily in three ways depending on the amount to change required to derive the new ontology [39] [40]. In this work, we recommend ontology merge to integrate ontologies.

To know more about ontologies, the reader is encouraged to visit the W3C standards [32]. The described features are essential while designing and implementing the proposed SDI framework.

III. Modular Design and Interoperability

The assistance system should be designed to be adaptive and flexible, such that it should be possible to combine different CPS with very varying capabilities without requiring extensive configuration from the worker. This flexible design makes it possible to scale the intelligence of the overall system by adding/removing CPS. The paper assumes that the central system contains a process description model, which describes the instructions for a process. The model remains unchanged irrespective of addition or removal of CPS modules. Adding new CPS modules to the central system makes the complete assistance system more aware of its environment and consequently more intelligent.

An assistance system, considered in this work, has hybrid architecture which consists of CPS modules and a central system where each CPS module collects and preprocesses data and feeds information to a central decision-making entity as shown in Figure 1. The central system collects information from all the modules attached to it and decides the next step of the process depending upon the process description model. Next step in the process is conveyed to a worker with the help of visualisation devices as shown in Figure 1. In contrast to a completely centralised or decentralised architecture, in a hybrid architecture, the burden of *making* sense from the raw-data is divided between the CPS modules and the central system: the modules need to preprocess raw data and make minor decisions before reporting it to the central system. The preprocessing step may include operations like analog to digital conversion, computing a parameter which is a function of data from more than one sensor (e.g., numberofParts from totalWeight and weightPerPart), calculating a moving average of a sensor reading, etc. This avoids any computing overhead on both the central system and CPS modules, and consequently makes them more intelligent and context-aware. This division is discussed in detail in Section IV.

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, in this work, a framework for designing the modules using ontologies is proposed, which will allow the modules to access and use information from each other.

There are several challenges which need to be addressed in order to allow for such interoperability. The paper shows how 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., URIs/IRIs, to *each* piece of information in the complete system, thus, making it immediately accessible using a simple declarative querying language (SPARQL) as shown in Figure 2 [41]. Moreover, other advantages come naturally with using ontologies, viz. self-documentation, automatic reasoning using description logic for free.

Using ontologies as the tool of choice, the following two questions are considered.

(i) **How to design and semantically annotate a CPS module?** This question is answered in Section IV.

(ii) **How to develop such modules using ontologies?** This issue is discussed in Section VI and in Section VII.

Remark. Note that the decision-making algorithm in the central system should be designed in such a 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 the scope of this work.

IV. Framework for designing a CPS

In this section, a framework for designing a CPS module and its ontology is proposed as shown in Figure 7. It starts with *what* the module designer wants to achieve by adding a particular CPS to the system, and then determines its boundary, or scope, with respect to the central system. Next, decisions about the *intelligence* of the system are made which, in turn, influence the hardware choices for the module. Finally, a bottom up ontology of a CPS is created and its integration with the central system ontology is described. Examples inspired from Figure 1 are discussed throughout the paper. The framework, and its implementation, are explained with the help of a use case of an inventory module which is shown in Figure 4.

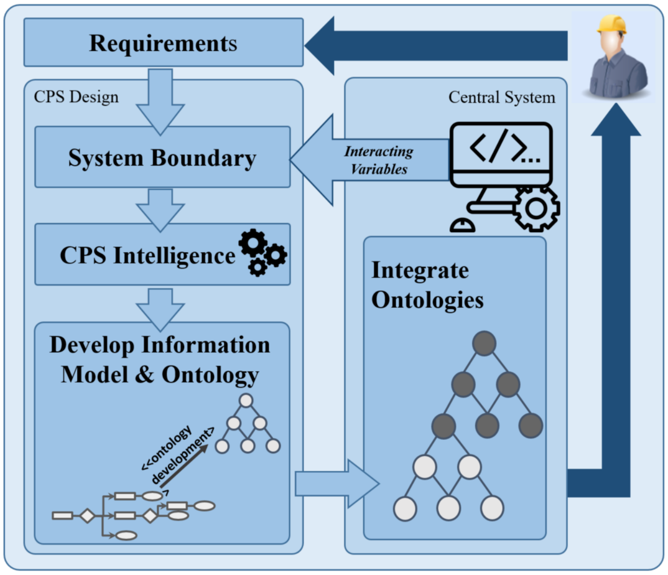


Figure 3. SDI framework for designing a CPS module.

A. Requirements

At the outset, it is important to understand *why* a CPS module is required. This decision determines the metric used for measuring the effectiveness of a module finally. This objective may range from general, e.g., “increasing the efficiency of a factory”, to specific, e.g., “decreasing the number of errors for a particular assembly station”.

For example, the requirement behind adding an inventory module can be to make the assistance system more aware of the environment in order to better understand the state of the process by the virtue of parts used in the process. This, in turn, improves the ability of an assistance system to help the worker. Keeping the requirements as specific as possible helps with the next step of the design.

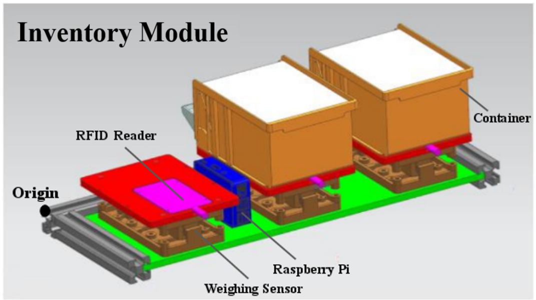


Figure 4. Schematic description of an inventory module

B. System Boundary

In the next step, the objective needs to be translated into 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 *minimal sufficient statistic*: the information should be *sufficient* for the central system to arrive at its objective. This information is the *interacting variable* between a CPS module and the central system.

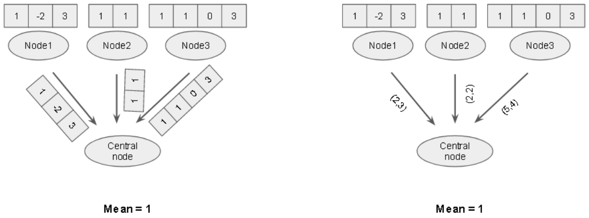


Figure 5. Two ways of calculating mean: in the first case complete raw data is provided to the central node to calculate mean whereas in the second case, only the sufficient statistic is provided.

In statistics, a statistic is sufficient with respect to a parameterized statistical model if no other statistic that can be derived from the same sample (e.g., raw sensor data) provides any additional information as to the value of the parameter. For example, consider the sufficient statistic to calculate mean of samples which are distributed across multiple nodes as shown in Figure 5. Each node only needs to report the sum of its samples and the number of samples to the central node doing the calculations. The central node then can calculate the total sum and the total number of samples and produce the mean without having the complete raw data (thereby saving computation and communication costs).

In terms of ontologies, the interacting variable needs to have the same URI/IRI in both the central system ontology as well as the ontology of the CPS module. The choice of sufficient static is driven by the data required by central system. In other words, the vocabulary, i.e., the terms defined by the central system, decide the

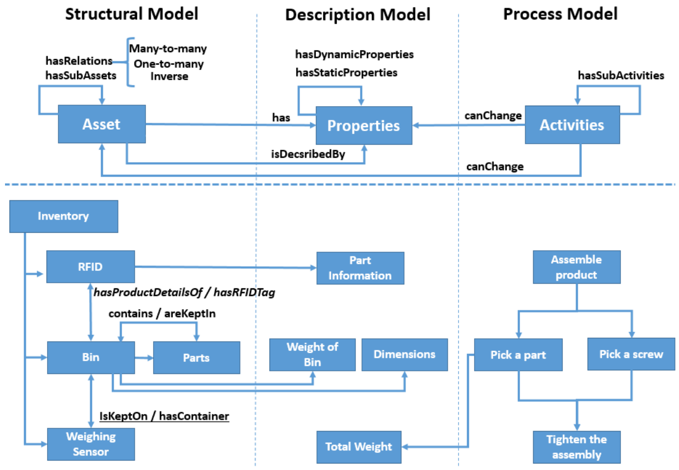


Figure 6. Information model contains structural, description and process models.

system boundary. This is ensured by defining the interacting variable in the upper ontology of an assistance system and the CPS module *importing* it.

For example, the central system may need the `total number of parts` for each part on the assembly station from an inventory module. This is the interacting variable for the CPS module.

C. CPS Intelligence

Once the system boundary is known, 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 (e.g., Raspberry Pi or Arduino) used to create the CPS module. Calculation of the value of the interacting variable 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, e.g., calculating moving average of raw data every millisecond. The system intelligence can further be improved by using more sophisticated hardware and/or applying better algorithms while processing data, which improves the *quality* of the values calculated by the CPS module for the interacting variable.

Also, note 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 [11].

D. Developing the Information Model & Ontology

After deciding on the hardware to use for a module, an *information model* which is an abstraction of the physical layer is created based on the structural and description models of the physical units present in a CPS module (as shown in Figure 6). The structural model defines physical assets present in a module: it lists all sensors, computational units, communication units and relations between them. The description model describes the properties of these assets. The process model is the process description that exists in the central system and is not changed on addition/removal of CPS modules. Structural and description models of the information model are used to explicitly define the hardware that was decided in the above steps. Figure 6 also shows the structural and description models of an inventory module and the process model contained by the central system.

The ontology of a CPS module is developed using the information model as a reference. In addition to the entities and relations defined in the information model, the ontology may also contain variables which are the result of *processing* the data gathered by sensors. Finally, the interacting variable(s), which were decided while determining the system boundary, are added to the ontology with appropriate relationships with other entities.

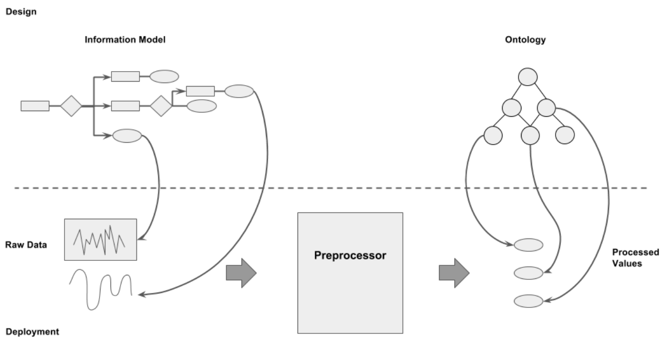


Figure 7. The information model (top-left), which is based on the physical setup of the system, is used to design the ontology (top-right) during the design phase. During implementation of the model, in the deployment phase, the sensors in the information model produce some raw data. This raw data is preprocessed by the CPS module (this is where system-intelligence comes into play) and is made ready for the ontology.

E. Ontology Integration

In the final step, ontology of the CPS module is merged with the central system ontology. The central system uses the interacting variable for making its own decisions, but also acts as a database for the complete assistance system during run-time. The modules, hence, can query the central system for not only the interacting variables of other modules, but also about the internal entities, which the central system does not explicitly use. The problem of how can the CPS modules be made aware of the various entities which can be accessed is addressed next.

As discussed before, the interacting variables are described in an upper ontology and a mid-level ontology contains descriptions of the entities of *all* modules. To help the ecosystem develop, a committee which consists of all shareholders (central system designers, deployment point managers, module developers, etc.) which oversees the addition to new modules to the ontology would be needed. The upper ontology is kept minimal and is only extended with new interacting variables, i.e., when a new potential CPS module is identified which can aid the intelligence of the central system. The other entities which can be provided by the new module, but which are not needed by the central system, are described in the mid-level ontology. The mid-level ontology acts as a repository of all relevant entities described in all CPS modules. This simplifies the search by engineers for variables provided by other modules. CPS modules `<<import>>` the upper ontology to get the URIs/IRIs of interacting variables and mid-level ontologies to get the URIs/IRIs of the entities of *all* modules.

Instead of having a mid-level ontology, it is possible to have only an upper ontology and ontologies of CPS modules. In such a setting, if one module needs to query for the variables of other CPS module, it then `<<import>>s` the ontology of that particular module. However, this scheme of ontology development may result in reinvention of entities. Thus, a centralised W3C committee like setup [32] which consists of all stakeholders is favoured.

V. Ecosystem of Ontologies

This section describes two possible ways of creating and maintaining ontologies for a complete assistance system. These ecosystems can be classified mainly into the two following ways:

A. *Decentralised scheme of ontologies*

In this section, a decentralized organizational scheme for the ontologies is described. As shown in Figure 8, upper ontology of assistance system is designed. To recap, upper ontologies of modules are created from their information models. CPS module ontology is described using its information model. These upper ontologies consists of definitions of entities and relationships between them.

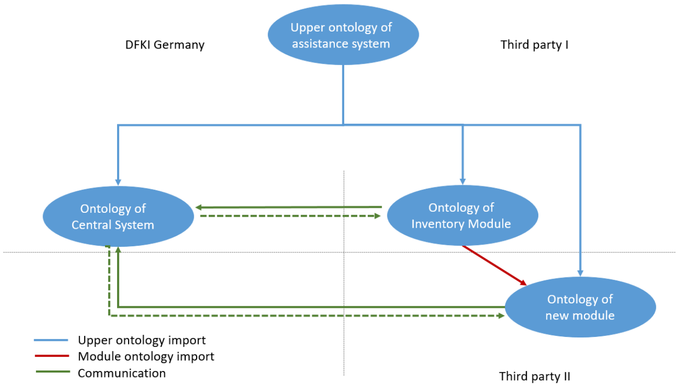


Figure 8. Schematic description of a decentralized ecosystem of ontologies.

So, the basic vocabulary described by upper ontology of assistance system is imported by the modules' ontologies. As information model maps all possible data, inventory module upper ontology contains definitions which are needed for the inventory module itself, but are not required by the central system ontology. An example of this can be position of container (x, y, z). This allows for flexibility in implementation of inventory module. If another module requires data regarding container position, ontology of that particular module can import the inventory module ontology.

Pros & Cons. This design focuses on building a completely decentralized system. The central system's ontology only contain the minimal taxonomy of entities and properties which are necessary for the Central System to function, i.e., be able to use the information from the modules effectively. However, the individual modules are free to report any variable which they can measure and to report it to the central system. The central system will store that information even if it may not have explicit uses for the variables but can produce this knowledge if a different third party module requests for it through SPARQL queries.

However, such a setup has the disadvantage that independent teams may reinvent properties independently and since these properties 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 and for the same information from different inventory modules the a new module would have to query independently.

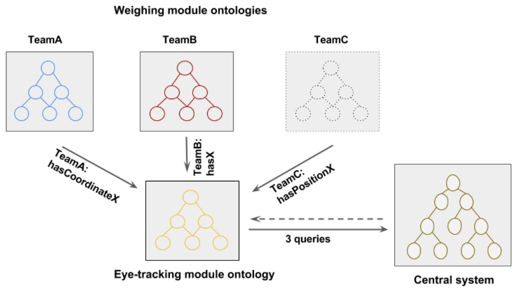


Figure 9. Example of possible reinvention of entities with same semantic meaning.

Figure 9 shows an example of such a situation where teams A, B and C independently define the variable for position of container as `hasCoordinateX`, `hasX` and `hasPositionX`. On the other hand, if the teams follow a particular nomenclature for defining variables would avoid reinventing similar variables which reduces the number of both imports and queries. Consolidation of the property names may also suffer due to the Not-Invented-Here syndrome [42].

A more subtle, and potentially more dangerous, side-effect of this design is compromised security of the data stored in the central system. In this design, the central system is completely unaware to the features which are being developed by independent modules. Hence, the central system needs to be excessively permissive when it comes to allowing arbitrary SPARQL queries by third party modules. A malicious module can very easily take advantage of vulnerability to obtain data on the central system.

B. *Centralised scheme of ontologies*

This section describes a centralized organizational scheme for the ontologies. As shown in Figure 10, upper ontology of assistance system is created which consists of the basic vocabulary for the complete system. Then a mid-level ontology is created. This mid-level ontology imports the upper ontology of assistance system. Further, the mid-level ontology describes the entities of all other modules. Depending on the engineers describing the mid-level ontology, all or some of the significant entities used by other modules are defined in the ontology.

The idea behind creating a mid-level is to create a repository of all relevant entities described in any CPS module. This simplifies the search by engineers for variables required by other modules. Mid-level ontology collects entities and their definitions described by upper ontologies of modules to facilitate exchange of data and this differentiates the approach from the previous approach. An assistance system upper ontology defines the minimal variables requires by modules to send data to the central system. This ontology is governed by the highest level committee and usually changes to it will be made when new modules are attached to the assistance system. On the other hand, modifications can be done easily in mid-level ontology which gives engineers the freedom to extend and access variables.

All modules' upper ontologies import the mid-level ontology. All modules need to import the mid-level ontology only once as it has all entities defined in the complete system.

During the design of the module, the interacting variable(s) were added in the upper ontology while the mid-level ontology was updated to include all entities which the module could provide, as agreed by all the stakeholders. For the purpose of exposition and to maintain complete generality, it is assumed in this section that the developer creating the module is a third party who intends to develop a newer version of the module from the specification.

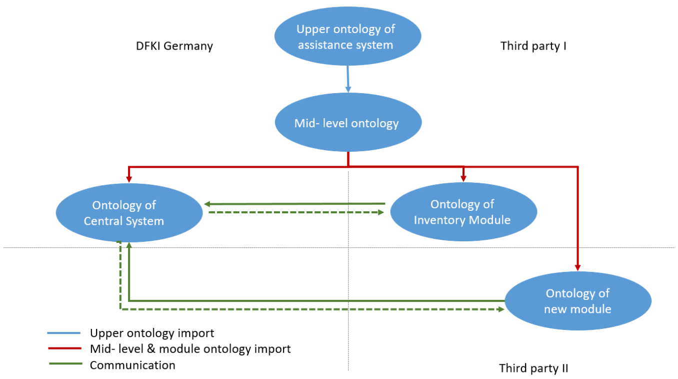

DFKI Germany
Upper ontology of assistance system
Third party I
Mid- level ontology
Ontology of Central System
Ontology of Inventory Module
Ontology of new module
Upper ontology import
Mid- level & module ontology import
Communication
Third party II

Figure 10. Schematic description of centralized ecosystem of ontologies .

Pros & Cons. In comparison with the more decentralized structure given in the previous section, the benefits and costs of this design are apparent. First of all, the mid-level ontology can be viewed as a *white-list* of entities and properties which can be read from the central system during execution through SPARQL queries. This prevents reinvention because a cursory check through the mid-level ontology will show that the 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, thus, modules may again reinvent variables. These extensions can be made directly in the mid-level ontology to avoid reinvention on the next level. However, whether these extensions should be a part of the mid-level ontology is not discussed in detail in this work and can be seen as a future work.

Because of the white-list provided by the mid-level ontology, the central system can also put in place a system for authorizing certain (known) modules to have access to information which is not available to other modules. This allows security sensitive data to be *inaccessible* from potentially malicious or unknown modules. The exact authentication mechanism will depend on public key cryptography [43], which is out of scope of this work.

Pair-wise collaboration of teams is not encouraged in this setup. This can be a potential downside of the organizational scheme. This may increase the development time for a particular module if the properties it needs to import are not in the white-list already and the decision and procedure of whether to add these properties might take longer compared to the previous design scheme.

Based on the discussion is this section, centralized structure of ontolgies with mid-level onotlogies is selected for development of CPS module for the proof of the concept.

VI. Module Development

This section describes at a high level the development of a CPS module and the central system after the design for the module has been included into the upper and mid-level ontologies. During the design of the module, the interacting variable(s) were added in the upper ontology while the mid-level ontology was updated to include all entities which the module could provide, as agreed by all the stakeholders. For the purpose of exposition and to maintain complete generality, it is assumed in this section that the developer creating the module is a third party who intends to develop a newer version of the module from the specification.

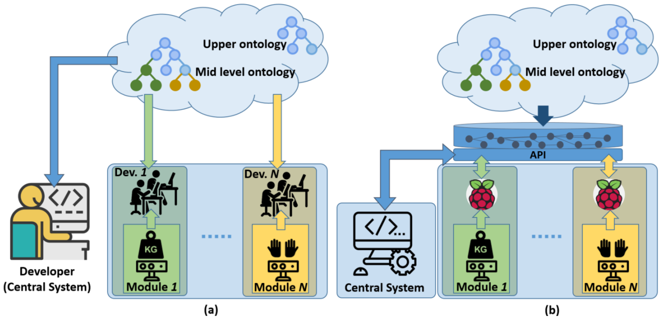


Figure 11. Ontology development of CPS modules.

In the next step towards development of the module, on the one hand, the developer (say, *Dev.* 1) studies the capabilities of the hardware available to her. Here, the developers can leverage the information model and ontology created during the design phase. On the other hand, the developer studies the upper (mid-level) ontology to determine what entities/values they should (could) provide to the central system. This part of the development process is illustrated in Figure 11(a). It should be noted that there is no need for communication or synchronisation between the developers of the different modules or between the developers and the central system developer. The developer `<<import>>s` the upper and mid-level ontologies and creates the module ontology with the remaining (local) ontological entities, and writes code which uses the central system's Application Programmable Interface (API) and SPARQL queries to update the central system database (as shown in Figure 11(b)).

Lastly, it is advised that Protégé should be used to create the module ontology as (i) it enhances interoperability by using OWL-S, and, (ii) it can automatically generate code using OWL API, which can ease the burden on the developer.

In this work, Protégé is used to create ontologies and the code generated is used to update the ontologies. Figure 12 shows an ontology created in Protégé and Figure 14 shows code generated by the API. In the next section, simulation of the central system and an inventory CPS module using Protégé is discussed.

VII. Implementation

This section describes implementation of a CPS module developed during this work. An inventory module as shown in Figure 1 is attached to the assistance system. The hardware implementation deals with reading data from sensors and RFID tags, processing the data and sending the required information to central system of assistance module. Sensor data and part information from RFID tags are used to provide information about the part in each container. In the presented scenario, Rapsberry Pi (RPi) attached to weighing module provides information about part name, total number of parts contained in each container and change in the number of parts for each container to a central system. RPi also sends a message to signal low inventory for parts if number of parts in a container drops below a threshold level.

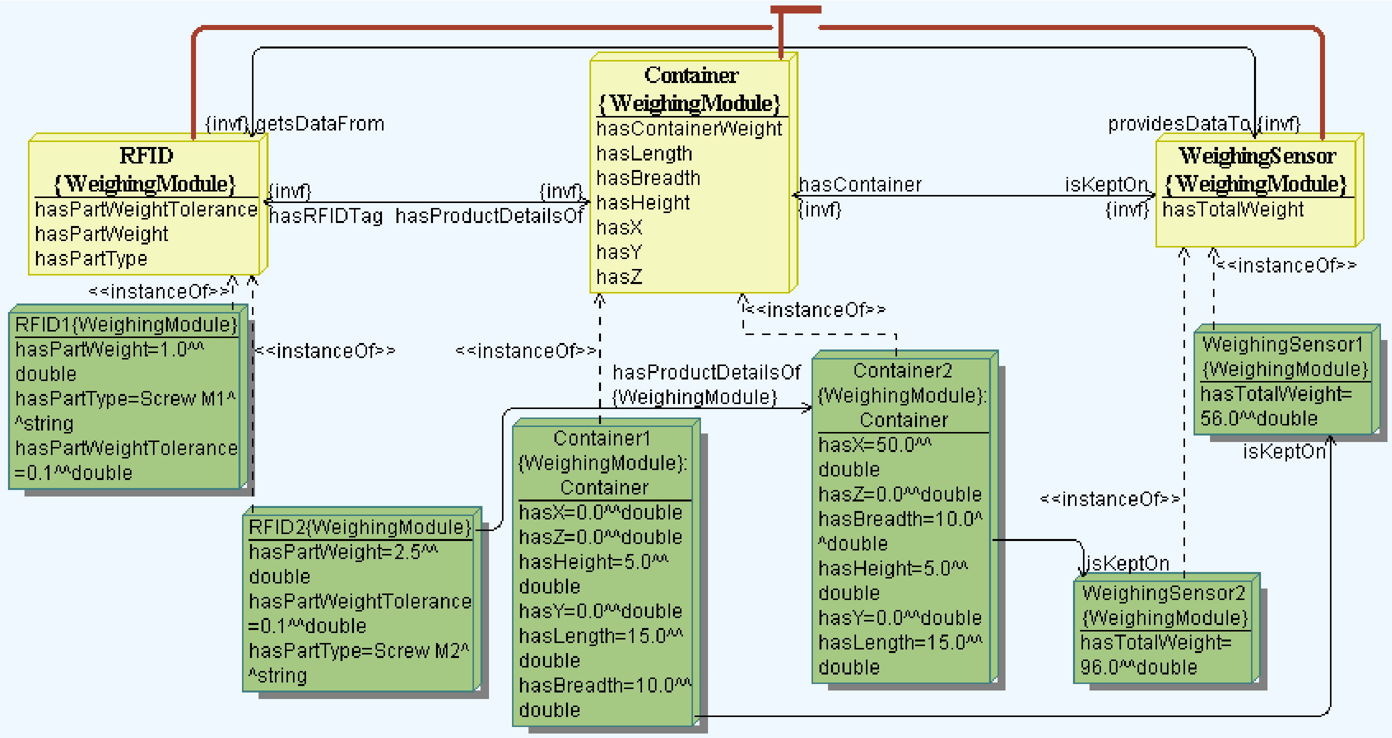


Figure 12. Example of classes and their relations as described in Protégé.

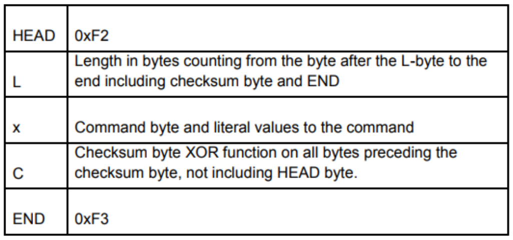
Figure 4 shows the setup of weighing module used in implementation. Weighing module has three weighing sensors with container kept on each sensor. Further, an RFID tag is attached to each bin. RFID tags contain data regarding parts, for example type of part, part name which can be read by RFID readers kept on weighing sensors as shown in the Figure 4. However, it is noteworthy that there is a scope of human error in this scenario as the part details are entered manually and while filling the container it should be ensured that container has the corresponding parts. This, indeed, can be one area of further improving the system and making it error proof.

This section discusses various hardware and communication choices available for implementation and makes recommendations based on the issues faced chronologically during implementation. First, different hardware options and procedures are discussed followed by communication between the central system and the inventory module and a few implementation recommendations are made.

A. Hardware Design

As discussed in Section I, an assistance system consists of a central system and CPS modules. In this implementation, weighing modules are the only kind of module attached to the central system.

Calibration of the sensors avoids any discrepancies in weights. Calibration follows a procedure wherein are given to calibrate the weighing sensor against dead-load. Since each sensor has a standard container and an RFID reader, the weights of these two entities are included in the dead-load of weighing sensor for the ease of calibration procedure. Including the weight of container and RFID reader in dead-load would lessen the complication in finding the number of parts as the weighing sensor will report only the weight of parts as opposed to the weight of the whole setup. However, weight of container is still a data node in our information model and ontologies in order to capture as much data as possible.



HEAD	0xF2
L	Length in bytes counting from the byte after the L-byte to the end including checksum byte and END
x	Command byte and literal values to the command
C	Checksum byte XOR function on all bytes preceding the checksum byte, not including HEAD byte.
END	0xF3

Figure 13. Example of definitions of command bytes.

This software implementation can be done for a micro-controllers, a Raspberry Pi (RPi) or a computer. RPi has more computational power than micro-controllers. It also has lower cost & lower power consumption than computers and is easy to use for programming. Hence, RPi was used for the implementation.

HEAD | L | x | C | END

Listing 1. An schematic layout of an encoded message to the weighing module. See Figure 13 for an explanation of each part of the command.

The inventory module uses communication protocol RS485 whereas the de-facto protocol for computer communications is `RS232`. There are multiple ways of converting `RS485` signals to `RS232`. An `RS485` shield, which sits on RPi, receives voltage corresponding to `RS485` and converts it RPi standard I/O voltages, is used in the implementation.The inventory module uses a LAN (*RJ45*) cable to power and to send/receive data to RPi.

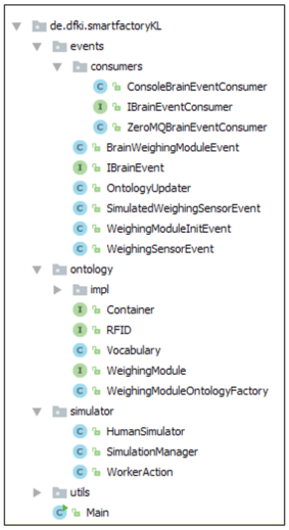
A Python program is written to calibrate the module. Raw data from sensors are read in hexadecimal form and is converted to decimal for the ease of reading and understanding. The program sends byte-encoded messages to the sensor which responds by sending bytes back. The byte code message for different commands are provided as the documentation for invnetory module. The documentation and code written for reading sensor data are provided on GitHub [44]. The module works on the principle that it gets a predefined encoded message from the user/RPi and depending on the value of message it returns the desired bytes. Listing 1 shows the general encoding of messages to/from sensors. Definitions regarding the command are provided thereafter in Figure 13.

Part information is read from RFID tags which are placed at the bottom of containers. RPi collects the data from sensors, part information from RFID tags and extract information regarding total number of parts and change in number of parts for each container. Python code also incorporates the detail of inventory threshold for each part. If inventory for a part goes below this threshold, a flag is raised to signal that refilling of parts is required.

B. Software Design

In the previous section, the development phase of ontologies was discussed. In this section, the simulation of an assistance system during run-time is discussed. The simulator in our implementation consists of two parts: a Python program for the central system and a Java program for an inventory module (see Figure 16) for the setup. The communication between the module and the central system uses ZeroMQ [45] and can be transparently done on a single machine or multiple machines. The details of the communication will be given in the next section.

Assistance system ontology is developed in Protégé, a free, open source ontology editor. The code generated via OWL API using Protégé (as shown in Figure 14) is used to simulate the behaviour of the CPS module. This implementation is written in Java. During execution, the ontology is *populated* by creating *individuals* locally on the module during execution. The central system may answer queries sent to it in SPARQL or may provide it using an alternate API. However, the use of unique URIs/IRIs to refer to entities in the ontologies is crucial to facilitate interoperability in all implementations.



```
▼ de.dfki.smartfactoryKL
  ▼ events
    ▼ consumers
        C ConsoleBrainEventConsumer
        I IBrainEventConsumer
        C ZeroMQBrainEventConsumer
      C BrainWeighingModuleEvent
      I IBrainEvent
      C OntologyUpdater
      C SimulatedWeighingSensorEvent
      C WeighingModuleInitEvent
      C WeighingSensorEvent
  ▼ ontology
    ▶ impl
      I Container
      I RFID
      C Vocabulary
      I WeighingModule
      C WeighingModuleOntologyFactory
  ▼ simulator
      C HumanSimulator
      C SimulationManager
      C WorkerAction
  ▶ utils
    C Main
```

Figure 14. Classes generated by Protégé, based on OWL API. Central system discussed in the paper is referred to as `Brain`.

C. Communication Design

During execution, the system goes through three primary stages: (i) intialization, (ii) trigger, and (iii) update, which are shown in Figure 15, and are briefly discussed here:

- **Initialization.** When an assistance system is started, the central system sends an *init()* request to all CPS modules attached to it. This request contains the URI/IRI of the central system. This URI/IRI is address with which all modules identify the central system through the lifetime of the process. In case of hardware malfunction, system restart, or when a new module is attached to the system, the initialization step is executed again.
- **Trigger.** Triggers can be either timer-driven or event-based. Event-based triggers are reported by CPS modules to the central system whereas timer-driven triggers are generated by the central system. Event-based triggers can be events that change the present state of a system to another (valid) state of the system [9]. In case an event occurrence renders no valid state of the system, triggers are not generated. *Trigger()* request is either sent from modules to the central system, as shown in Figure 15, or may be generated internally by the central system clock.
- **Update.** Communication between the central system and CPS modules is pull-based. Upon a trigger, the central system sends a *getUpdate()* request to all modules. Modules send the complete, or a part of, ontologies with the new data values to the central system which, in turn, update its own ontology.

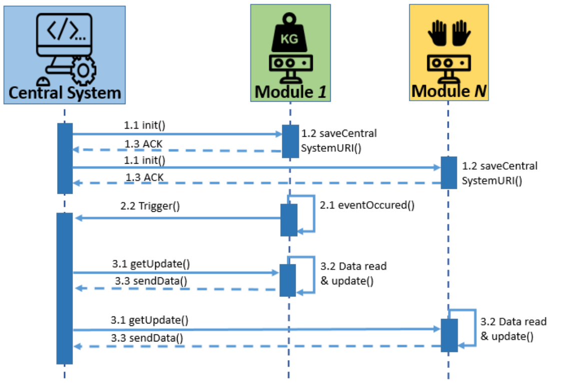


Figure 15. Communication between the central system and CPS modules.

An example implementation is available for download on GitHub [44]. The implementation therein simulates an inventory module (using code generated from Protégé), a central system, and then simulates human actions, updates the ontology on the inventory module using the OWL API, and shows the communication between the module and the central system.

D. Additional Recommendations

The inventory module must be developed keeping in mind the failures and errors that might happen while deploying the system. Thus, the system must have certain properties which make it error proof. Intertwined with the desirable properties of the system, some practical recommendations regarding implementation are also made in this section. The inspiration for these recommendations comes from design of concurrent systems [46].

Safety property asserts that *nothing bad* happens. The foremost requirement to implement this property is the system should not be in an invalid state at any point in time. For CPS, it means that the module should never report values which are not computed from its sensor values. A discrepancy may result from the scenario that the module reports a value to the central system after it has probed one sensor but before probing another sensors, if the report contains values which were computed using both the sensor's readings (one of the sensors may have out-dated value). An invalid state can also be a deadlocked state where there are no outgoing transitions, such as an error state. Semaphores, mutex and locks should be judiciously used during development to avoid such scenarios. As a side-note, handling missing values (which is a subset of error states) gracefully is a potential future extension of the work.

Liveness property asserts that the system will perform its intended use *eventually*. In other words, liveness means that the system will continue to make progress. This implies ensuring that the semaphores and mutexes will be unblocked and the module will, eventually, send data to the central system. Though several race conditions can be avoided simply by using atomic operations exclusively, it is possible to end up in a live-lock. Say the module has a *hard* parameter which controls after how long a sensor's data is considered *stale*. Then say the module reads data from one sensor, and then while it is reading data from another sensor, the data from the first sensor becomes *stale*. So, the module will go back to re-reading data from the first sensor and in the meanwhile data from the second sensor becomes *stale*. This could bind the module into a sensor reading infinite loop. During implementation, such situations should be carefully thought about and the liveness of the module should be tested/verified under the most extreme of conditions.

Encapsulation is another way of making system more reliable. Encapsulation is restricting direct access of software components so that they cannot interfere with other subsystems or privileged-level software. It keeps faults from propagating which increases the reliability of the overall system.

Finally, in case everything fails, a **watchdog timer** (or a Heartbeat) can be used to detect the catastrophic failures and recover from it. The timer is regularly reset by computer during normal operation, but it timeouts if there is a hardware or software error. The timeout signal initiates a corrective measure by placing the system in a safe state or restoring normal operation. One of the ways this can be accomplished is by using a Hypervisor [47] which can simply restart the entire module in case the timer timeouts.

These are some necessary properties that the system must have, but not sufficient to ensure that it functions properly. In the end, the deployment and user feedback would be the final test of the module.

VIII. Conclusion

This work is focused on designing a human-centric assistance system used in production which can dynamically adapt to the needs of the workers and tasks using Semantic Web technologies. Assistance systems are considered as consisting of a central system and one or many CPS modules. An SDI framework is proposed to design CPS modules which makes the data of the complete system globally accessible by the virtue of HTTP URIs/IRIs. The SDI framework explained the steps used to decide the boundary between the central system and CPS modules, the performance requirements of hardware, describing modules with the help of information models and finally developing and merging ontologies. It also explains the ecosystem of ontologies consisting of upper, mid-level and module ontologies. Hardware and software implementation of an inventory module is completed. For the inventory module, the framework is implemented in Protégé using OWL-S. OWL API is used to simulate CPS behaviour and data exchange is demonstrated. Communication between a CPS module and the central system is also described. However, the proposed framework can be used to design CPS in general: the discussion in the paper was limited to designing a CPS for an assistance system for ease of both exposition and demonstration.

The work assumes that all vendors and third party development use SPARQL as the query language. Calbimonte et al. have discussed how such a problem of multi-vendor multi-querying language can be resolved [48]. It can be incorporated in the SDI framework to make it more robust. Knowledge mapped in ontologies may evolve over time due to modifications in conceptualisation and adaptation to incoming changes. Thus, in future, it is important to establish protocols for versioning of data on Semantic Web as well as understanding the missing data [49]. Another non-trivial task towards adoption of ontologies in real life is setting up committees which oversee the creation and maintenance of upper and mid-level ontologies [50].

The framework results in a repository of data from all modules of the system and interoperability between these modules, thus, laying the foundation of plug-and-play production systems. The next important step in the development of assistance systems is to develop a plug-and-play methodology for CPS modules, as alluded to in Section III.

Another important step to make the system deployable is 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 4.0 [51].

References

[1] A. Singh, F. Quint, P. Bertram, and M. Ruskowski, "Towards modular and adaptive assistance systems for manual assembly: A semantic description and interoperability framework," in *The Twelfth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2018)*, 2018.

[2] M. M. Tseng and S. J. Hu, "Mass customization," in *CIRP encyclopedia of production engineering*. Springer, 2014, pp. 836–843.

[3] F. Salvador and C. Forza, "Configuring products to address the customization-responsiveness squeeze: A survey of management issues and opportunities," *International journal of production economics*, vol. 91, no. 3, pp. 273–291, 2004.

[4] Y. Koren and M. Shpitalni, "Design of reconfigurable manufacturing systems," *Journal of manufacturing systems*, vol. 29, no. 4, pp. 130–141, 2010.

[5] D. Romero, O. Noran, J. Stahre, P. Bernus, and Å. Fast-Berglund, "Towards a human-centred reference architecture for next generation balanced automation systems: human-automation symbiosis," in *IFIP International Conference on Advances in Production Management Systems*. Springer, 2015, pp. 556–566.

[6] S. Tzafestas, "Concerning human-automation symbiosis in the society and the nature," *Intl. J. of Factory Automation, Robotics and Soft Computing*, vol. 1, no. 3, pp. 6–24, 2006.

[7] P. A. Hancock, R. J. Jagacinski, R. Parasuraman, C. D. Wickens, G. F. Wilson, and D. B. Kaber, "Human-automation interaction research: past, present, and future," *ergonomics in design*, vol. 21, no. 2, pp. 9–14, 2013.

[8] V. Villani, L. Sabattini, J. N. Czerniak, A. Mertens, B. Vogel-Heuser, and C. Fantuzzi, "Towards modern inclusive factories: A methodology for the development of smart adaptive human-machine interfaces," *22nd IEEE International Conference on Emerging Technologies and Factory Automation*, 2017.

[9] F. Quint, F. Loch, M. Orfgen, and D. Zuehlke, "A system architecture for assistance in manual tasks." in *Intelligent Environments (Workshops)*, 2016, pp. 43–52.

[10] E. Tantik and R. Anderl, "Integrated data model and structure for the asset administration shell in industrie 4.0," *Procedia CIRP*, vol. 60, pp. 86–91, 2017.

[11] 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 *Semantic Computing (ICSC), 2016 IEEE Tenth International Conference on*. IEEE, 2016, pp. 230–237.

[12] J. Gezer, Volkan Um and M. Ruskowski, "An extensible edge computing architecture: Definition, requirements and enablers," in *The Eleventh International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2017)*, 2017.

[13] D. Zuehlke, "Smartfactory–from vision to reality in factory technologies," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 14 101–14 108, 2008.

[14] M. Gaham, B. Bouzouia, and N. Achour, "Human-in-the-loop cyber-physical production systems control (hilcp 2 sc): A multi-objective interactive framework proposal," in *Service orientation in holonic and multi-agent manufacturing*. Springer, 2015, pp. 315–325.

[15] E. A. Lee, "Cyber physical systems: Design challenges," in *11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*. IEEE, 2008, pp. 363–369.

[16] N. Jazdi, "Cyber physical systems in the context of industry 4.0," in *Automation, Quality and Testing, Robotics, 2014 IEEE International Conference on*. IEEE, 2014, pp. 1–4.

[17] P. Leitão, "Agent-based distributed manufacturing control: A state-of-the-art survey," *Engineering Applications of Artificial Intelligence*, vol. 22, no. 7, pp. 979–991, 2009.

[18] 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 *Industrial Technology (ICIT), 2016 IEEE International Conference on*. IEEE, 2016, pp. 2099–2104.

[19] D. Gorecky, S. F. Worgan, and G. Meixner, "Cognito: a cognitive assistance and training system for manual tasks in industry." in *ECCE*, 2011, pp. 53–56.

[20] 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*, vol. 15, pp. 90–98, 2014.

[21] B.-C. Pirvu, C.-B. Zamfirescu, and D. Gorecky, "Engineering insights from an anthropocentric cyber-physical system: A case study for an assembly station," *Mechatronics*, vol. 34, pp. 147–159, 2016.

[22] D. Kolberg, C. Berger, B.-C. Pirvu, M. Franke, and J. Michniewicz, "Cyprof–insights from a framework for designing cyber-physical systems in production environments," *Procedia CIRP*, 2016.

[23] T. Arai, Y. Aiyama, M. Sugi, and J. Ota, "Holonic assembly system with plug and produce," *Computers in Industry*, vol. 46, no. 3, pp. 289–299, 2001.

[24] W. Wahlster, "Semantic technologies for mass customization," in *Towards the Internet of Services: The THESEUS Research Program*. Springer, 2014, pp. 3–13.

[25] M. Graube, J. Pfeffer, J. Ziegler, and L. Urbas, "Linked data as integrating technology for industrial data," *International Journal of Distributed Systems and Technologies (IJDST)*, vol. 3, no. 3, pp. 40–52, 2012.

[26] S. K. Semy, M. K. Pulvermacher, and L. J. Obrst. (2004) Toward the use of an upper ontology for us government and us military domains: An evaluation. Retrieved on 2018-09-20.

[27] C.-H. Cheng, T. Guelfirat, C. Messinger, J. O. Schmitt, M. Schnelte, and P. Weber, "Semantic degrees for industrie 4.0 engineering: Deciding on the degree of semantic formalization to select appropriate technologies," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 2015, pp. 1010–1013.

[28] M. Obitko and V. Jirkovskỳ, "Big data semantics in industry 4.0," in *International conference on industrial applications of holonic and multi-agent systems*. Springer, 2015, pp. 217–229.

[29] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data: The story so far," in *Semantic services, interoperability and web applications: emerging concepts*. IGI Global, 2011, pp. 205–227.

[30] A. Schultz, A. Matteini, R. Isele, P. N. Mendes, C. Bizer, and C. Becker, "Ldif-a framework for large-scale linked data integration," in *21st International World Wide Web Conference (WWW 2012), Developers Track, Lyon, France*, 2012.

[31] V. Gezer and S. Bergweiler, "Cloud-based infrastructure for workflow and service engineering using semantic web technologies," *International Journal on Advances on Internet Technology*, pp. 36–45, 2017.

[32] S. Bechhofer, "Owl: Web ontology language," in *Encyclopedia of database systems*. Springer, 2009, pp. 2008–2009.

[33] 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.

[34] N. Guarino, *Formal ontology in information systems: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy*. IOS press, 1998, vol. 46.

[35] I. Niles and A. Pease, "Origins of the ieee standard upper ontology," in *Working notes of the IJCAI-2001 workshop on the IEEE standard upper ontology*. Citeseer, 2001, pp. 37–42.

[36] J. F. Sowa *et al.* Building, sharing, and merging ontologies. Retrieved on 2018-09-20. [Online]. Available: http://www.jfsowa.com/ontology/ontoshar.htm

[37] R. Hoehndorf, "What is an upper level ontology?" *Ontogenesis*, 2010.

[38] E. Beisswanger, S. Schulz, H. Stenzhorn, and U. Hahn, "Biotop: An upper domain ontology for the life sciences," *Applied Ontology*, vol. 3, no. 4, pp. 205–212, 2008.

[39] J. F. Sowa *et al.*, *Knowledge representation: logical, philosophical, and computational foundations*. Brooks/Cole Pacific Grove, CA, 2000, vol. 13.

[40] H. S. Pinto, A. Gómez-Pérez, and J. P. Martins, "Some issues on ontology integration." IJCAI and the Scandinavian AI Societies. CEUR Workshop Proceedings, 1999.

[41] E. Prud *et al.* Sparql query language for rdf. Retrieved on 2018-09-20.

[42] R. Katz and T. J. Allen, "Investigating the not invented here (nih) syndrome: A look at the performance, tenure, and communication patterns of 50 r & d project groups," *R&d Management*, vol. 12, no. 1, pp. 7–20, 1982.

[43] R. E. Smith, *Authentication: from passwords to public keys*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[44] A. Singh. Example implementation of the SDI framework. Retrieved on 2018-11-16. [Online]. Available: https://github.com/AmitaChauhan/SDI-Framework

[45] P. Hintjens, *ZeroMQ: messaging for many applications*. " O'Reilly Media, Inc.", 2013.

[46] G. R. Andrews, *Concurrent programming: principles and practice*. Benjamin/Cummings Publishing Company San Francisco, 1991.

[47] M. Masmano, I. Ripoll, A. Crespo, and J. Metge, "Xtratum: a hypervisor for safety critical embedded systems," in *11th Real-Time Linux Workshop*. Citeseer, 2009, pp. 263–272.

[48] J.-P. Calbimonte, H. Jeung, O. Corcho, and K. Aberer, "Enabling query technologies for the semantic sensor web," *International Journal On Semantic Web and Information Systems (IJSWIS)*, vol. 8, no. 1, pp. 43–63, 2012.

[49] M. C. Klein and D. Fensel, "Ontology versioning on the semantic web." in *SWWS*, 2001, pp. 75–91.

[50] I. Jacobs. World wide web consortium process document. Retrieved on 2018-09-20. [Online]. Available: https://www.w3.org/2018/Process-20180201/

[51] M. Weyrich and C. Ebert, "Reference architectures for the Internet of things," *IEEE Software*, vol. 33, no. 1, pp. 112–116, 2016.