Towards Modular and Adaptive Assistance Systems for Manual Assembly: A Semantic Description and Interoperability Framework

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. 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 using Protégé is provided 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, 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 [1]–[3]. 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 [4][5]. 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 [6][7].

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 [8]. 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 [9][10]), and edge-computing [11] 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.

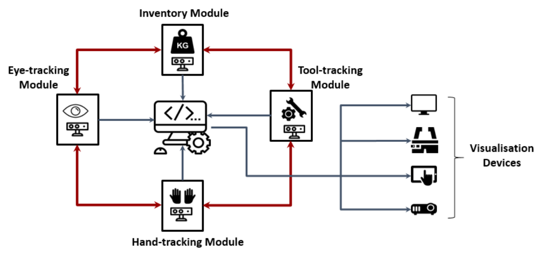


Figure 1. Schematic description of an assistance system.

The central system, in turn, processes this information to assess the state of the process and the worker's needs.

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 related work is presented in Section II and then the concepts of modularity and interoperability are discussed in detail in Section III. In Section IV, the SDI framework for design of modules is presented. Next, the development of such modules is discussed in Section V. Finally, the implementation of an assistance system is simulated using the proposed framework in Section VI, followed by the conclusion and potential future work.

II. Related Work

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.

Assistance Systems. There is significant contemporary research interest in using sensor technology for developing context-aware CPS [8][12]–[14]. Nelles et al. have looked into assistance systems for planning and control in production environment [12]. Gorecky et al. have explored cognitive assistance and training systems for workers during production [13]. Zamfiresu et al. have also integrated virtual reality and a hand-tracking module to help workers during assembly processes [14]. However, they do not consider the modular design of these modules and interoperability between such modules. 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 [8]. 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.

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. [15][16] use Semantic Web technologies to represent and integrate industrial data in a generic way. Grangel et al. [10] discuss Semantic Web technologies in handling heterogeneous data from distributed sources using light-weight vocabulary. Semy et al. [17] 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. 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 [18]. 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.

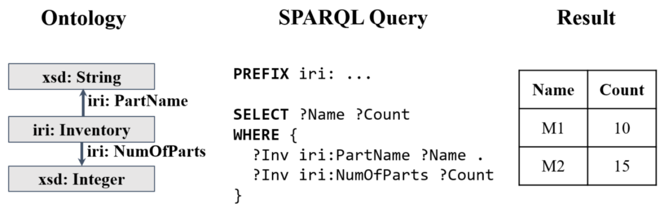
Interoperability. Interoperability is the ability to communicate and interconnect CPS from different vendors. It is vital in order to have cost effective rapid development. According to domain experts [10][16][19], RDF and Linked Data are proven Semantic Web technologies for integrating different types of data. Gezer et al. [20] 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 [10]. Further, data can be made available through a standard interface using SPARQL, a W3C recommendation for RDF query language [21].

Recently, Negri et al. [22] 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 [23]. 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. [24] 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. [17] 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. [25] 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 [21] are followed.

Ontologies. 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.



```
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.

To know more about ontologies, the reader is encouraged to visit the W3C standards [21]. 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 [26]. 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 V and in Section VI.

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 3. 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. The framework, and its implementation, are explained with the help of a use case of an inventory module which is shown in Figure 4.

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”.

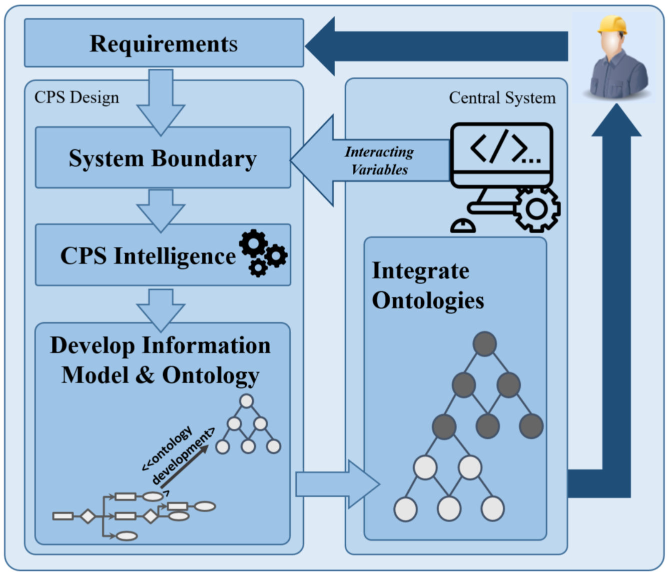


Figure 3. SDI framework for designing a CPS module.

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.

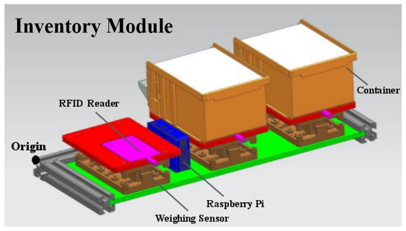


Figure 4. Schematic description of an inventory module

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. 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. 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.

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 [10].

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 5). 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 5 also shows the structural and description models of an inventory module and the process model contained by the central system.

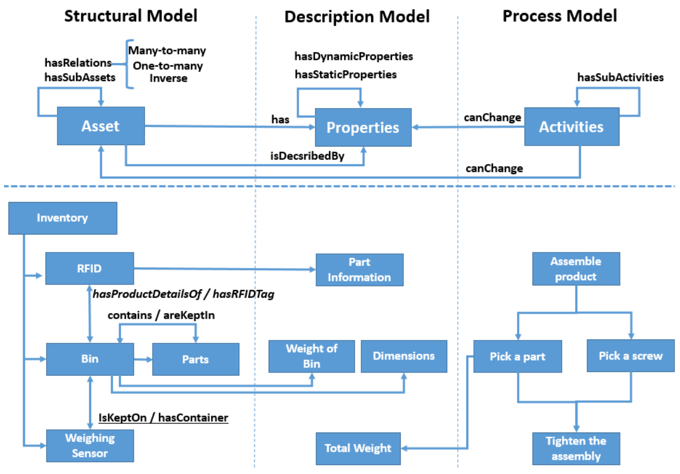


Figure 5. Information model contains structural, description and process models.

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.

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 [21] which consists of all stakeholders is favoured.

V. 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.

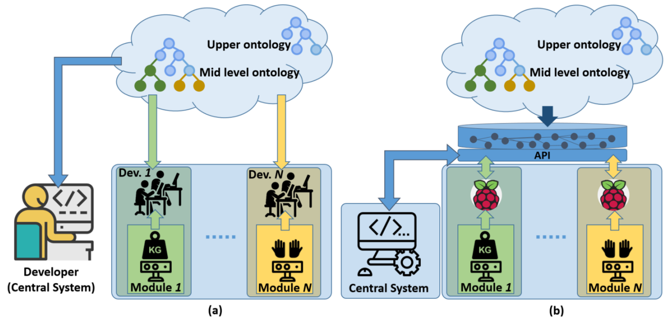


Figure 6. Ontology development of CPS modules.

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 6(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 API and SPARQL queries to update the central system database (as shown in Figure 6(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 (Application Programmable Interface), 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. In the next section, simulation of the central system and an inventory CPS module using Protégé is discussed.

VI. Implementation

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. Assistance system ontology is developed in Protégé, a free, open source ontology editor. The code generated using Protégé (as shown in Figure 7) is used to simulate the behaviour of CPS module, via OWL API. This implementation is written in Java. For the ease of exposition, it is assumed in the text that the central system can answer queries sent to it in SPARQL. These queries can be written in a different language or may be provided 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.

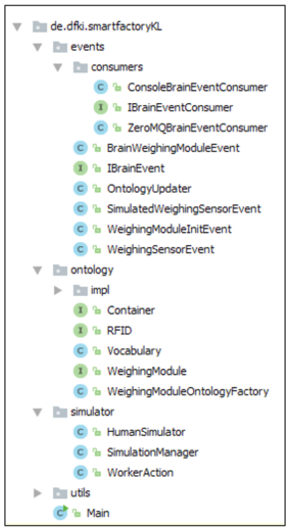

```
▼ de.dfki.smartfactoryKL
  ▼ events
    ▼ consumers
        C ConsoleBrainEventConsumer
        I IBrainEventConsumer
        C ZeroMQBrainEventConsumer
      C BrainWeighingModuleEvent
      I IBrainEvent
      C OntologyUpdater
      C SimulatedWeighingSensorEvent
      C WeighingModuleInitEvent
      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 7. Classes generated by Protégé, based on OWL API. Central system discussed in the paper is referred to as `Brain`.

It is assumed that ontologies of an assistance system, i.e the central system and CPS modules, are developed using the proposed framework. During execution, the ontologies are *populated* by creating *individuals* locally on all modules. During execution, the system goes through three primary stages: (i) intialization, (ii) trigger, and (iii) update, which are shown in Figure 8, 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 [8]. 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 8, 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()* reqeust 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.

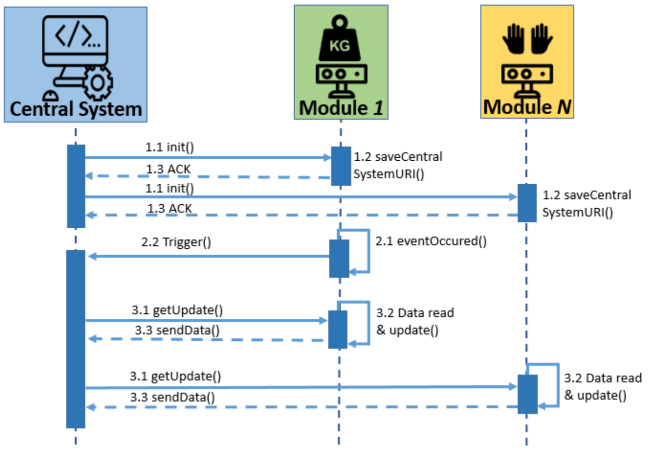


Figure 8. Communication between the central system and CPS modules.

An example implementation is available for download on GitHub [27]. 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.

VII. 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 briefly the ecosystem of ontologies consisting of upper, mid-level and module ontologies. The framework is implemented in Protégé using OWL-S. OWL API is used to simulate CPS behaviour and data exchange is demonstrated. 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 [28]. 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 [29]. 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 [30].

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 [31].

ACKNOWLEDGEMENTS

This work was done as a Master thesis at German Research Center for Artificial Intelligence (DFKI) and TU Kaiserslautern Germany.

REFERENCES

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

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] J. Gezer, Volkan Um and M. Ruskowski, "An extensible edge computing architecture: Definition, requirements and enablers," in *UBICOMM*, 2017.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

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

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

[27] A. Singh. Example implementation of the SDI framework. Retrieved on 2018-09-20. [Online]. Available: https://github.com/AmitaChauhan/SDI-Framework

[28] 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.

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

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

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