Available online at www.sciencedirect.com

ScienceDirect

Procedia Manufacturing 42 (2020) 197–204

Procedia MANUFACTURING

www.elsevier.com/locate/procedia

International Conference on Industry 4.0 and Smart Manufacturing (ISM 2019)

# Bidirectional Transformation of MES Source Code and Ontologies

Badarinath Katti[a,*], Christiane Plociennik[a], Martin Ruskowski[a], Michael Schweitzer[b]

[a]*Innovative Factory Systems, German Research Center for Artificial Intelligence, Kaiserslautern, Germany*
[b]*SAP SE, Walldorf, Germany*

**Abstract**

Future production environments must be flexible and reconfigurable. To achieve this, the devices and services to fulfill the different steps of a production order (PO) should not be selected in the manufacturing execution system (MES), but in an edge component close to the shop floor. To enable this, abstract services in the PO and concrete services provided by the field devices on the shop floor need to refer to a production ontology. The creation of this ontology is a challenge of its own. This research proposes a pragmatic automation of an encoding of a primary and light weight production ontology based on the source code of MES. The transformation procedure of source code to resource, product and generic concepts of the manufacturing plant ontology is described. To this end, the knowledge of OPC UA collaborations are also exploited during the creation of resource ontologies. Due to a fundamental difference between source code implementation (imperative paradigm) and ontology representation (declarative paradigm), the problem of information loss is inevitable. This problem is overcome by formulation of production and business rules that encapsulate the logic of the MES. The foundation of ontology is exploited to formulate these rulesets using OWL based constructs and OWL based rule languages such as Semantic Web Rule Language(SWRL), Semantic Query-Enhanced Web Rule Language (SQWRL) and SPARQL Protocol and RDF Query Language (SPARQL) based on feasibility and requirements of specific rules. Further, these rulesets are either run on the automatically generated ontology at design time with an intention to enrich the knowledge base, or production runtime to validate the pre-defined business rules between the production steps. The generated ontology also acts as basis for automatically generating the OWL-S ontologies for the OPC UA application methods for the purpose of dynamic manufacturing service discovery and orchestration. The generated ontology and an abstract PO hooked with formulated rules are cached to the shop-floor network for consequent production control to enable smart edge production. An implementation is conducted on an industrial use-case demonstrator to evaluate the applicability of the proposed approach.



*Keywords:* Cloud based MES; Production Automation; Knowledge Representation; OPC-UA; Ontology; OWL; SWRL; SQWRL; SPARQL; Edge Computing;

## 1. Introduction

The research continues the pursuit of delegating the responsibility for decision-making process in manufacturing to the production network layer in a scenario where cloud based MES executes and controls the overall activities. Previous research proposed an architecture for such an edge component, called Generic Shop-Floor Connector (GeSCo) [10], to carry out the entire production processes and this work builds upon it. This research work considers the manufacturing activities ranging from production design to production planning and production control. Though these three stages of manufacturing are distinct in theory, in reality the boundaries of each of these activities come into contact with each other. To achieve higher efficiency, integration of these stages is essential, and ontologies make it possible by making the data interoperable across different stages. The development of ontologies which allow to configure the complete manufacturing system using a model-based engineering approach is the step in the right direction for the advancement of domain and contextual knowledge. At the same time, when they are designed in certain specific ontology construction languages such as OWL and Resource Description Framework (RDF), they also permit effective assimilation of such knowledge in software and agent-based automation systems. To accomplish that, the authors propose to model the manufacturing and internal logistics system structure into an ontology which is machine processable and subsequently enables integration of knowledge within the automated systems.

* This research work is supported by a doctoral grant from SAP SE. A non-provisional patent, with ID 15/926,856, on the research of CMES controlled edge manufacturing has been filed in USA on 20-Mar-2018 with the authors as the inventors.

*E-mail address:* katti@rhrk.uni-kl.de (Badarinath Katti).


10.1016/j.promfg.2020.02.070

To summarize, the authors employ the ontology in this work for the following purposes:

- Automation of data assimilation among different software units
- Formal representation of relationships between various concepts in the information model and creation of relevant rules that need to be adhered to in the production
- Gain contextual awareness
- Assist GeSCo in decision-making activities to generate an on-the-fly configuration of the production processes that consists of dynamic orchestration of production resources
- Decentralized manufacturing planning and control without an upfront knowledge of factory layout

The traditional approach has been to design the information model in a formal modeling language such as OWL for the purpose of correctness and subsequently, this information model is used to build the applications. To this end , the general assumption is that an ontology is predefined by a domain expert and can then be taken as a starting point for further software engineering processes. However, ontology encoding is a time-consuming task. This paper proposes an automation of reverse engineering process of translation of source code of MES to the corresponding ontology. This approach substantially decreases the ontology engineering effort at design time. Mere ontology encoding is also not the final exercise in formal representation of production information model. Additionally, the semantic rule languages need to be employed to fill in the knowledge gap as OWL is not fully capable to express all the aspects of information model due to a lack of constructs. The generated ontology is used as reference to create further rules in OWL-based rule languages. Excluding exceptions, the generation of conceptual ontology and the formulation of corresponding semantic rules is an one-time process. Thus, these ontology concepts and rules can be reused to design the PO and validate the conditions of the assembly, resources and shop-floor during intermediate steps of production to determine next courses of action in production execution and control. In the rare events of changes in base ontologies owing to the corresponding changes in the source code of MES, only the production rulesets have to be readjusted to reflect the ontology changes.

## 2. Related Work

### 2.1. Transformation of XML/UML artifacts

There is some literature that focuses on the reverse engineering process of translating software artifacts to ontologies. Among such works, a conversion of an XML document or an UML diagram to an ontology is the most widely adopted use case across the industries. [17] is one of the earlier investigations regarding the feasibility of automatic data translation between XML and ontology and subsequently, devices an approach at the cost of accuracy and efficiency of generated ontology. Such XML to ontology translation processes exploit XSD files that define the possible structure and contents of XML instances as support systems. However, the correctness of XSD files is presumed. The same argument holds good for UML to ontology translation processes using XSLT instances. Recently, the OntoUML language has been developed which is used as UML profile for modeling sound ontology models [5]. Another attempt to convert UML diagrams of varying complexities to RDF ontologies as a part of OSLO project to increase the interoperability between belgian government services is presented in [4].

### 2.2. Transformation using statistical models

There are numerous research works that use sophisticated statistical models that rely on linguistic processing techniques to construct automatic and semi-automatic ontology models. For example, [13] applies the K-Means Clustering algorithm on a website based on a shipping yard to build and expand an ontology automatically.

A semi-automatic approach to generate a reference ontology w.r.t. the wind energy domain by crawling through relevant wikipedia pages was presented in [11]. But, such an open ended attempt using a web crawling program eventually renders a sub-optimal ontology. There are also numerous efforts to generate the ontology using Natural Language Processing (NLP) techniques where mostly text is analyzed to extract the required concepts and relations between those concepts. These supervised or unsupervised methodologies generate subpar ontologies at the first attempt that must be subjected to ontology evaluation techniques [16]. However, after the intervention the precision of the generated ontology is only incrementally improved. [2] is also a similar work of ontology generation based on patterns.

[3] comes close to our methodology where a formal ontology is generated using source code and other information systems as a starting point. But this approach is based on finding keywords in the source code whose logic depends on term frequency in an unsupervised environment. The approach also lacks *relation learning*.

However, in the area of manufacturing there is no previous work to reverse engineer the source code data into ontology representation. Our work translates the cloud based MES source code to a corresponding OWL based reference ontology with a certain degree of loss of information. This loss is then compensated by addition of axioms based on basic OWL constructs, and rules using various semantic rule languages over the reference ontology. The rules are analyzed per case basis and suitable rule languages are chosen to better represent them. The reference ontology, rules and PO are then cached to the network layer to enable smart edge production.

## 3. Research Use Case

A demonstrator system that produces the smart key finders was employed in order to evaluate the applicability of the proposed approach. The demonstrator setup contains industrial modules from various vendors to constitute a production cell where three individual parts of the key finder, namely, housing cover, housing base and the circuit board, are assembled.

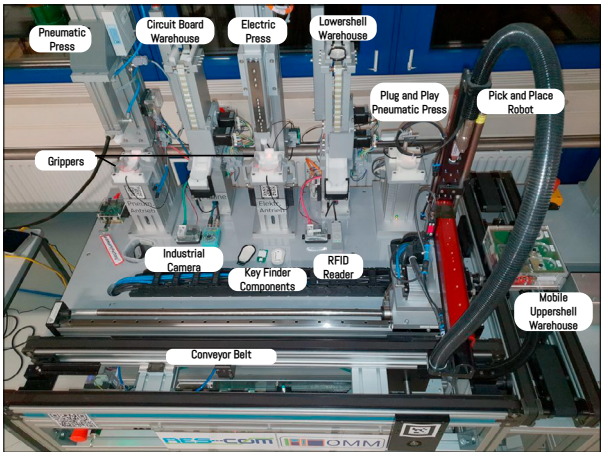


Fig. 1. OWL/Layout of the Key Finder Demonstrator.

The work station has multiple key finder assembly units and a general purpose pick-and-place robot which makes it convenient for experimenting with adaptability and reusability features of the manufacturing resources. Besides, the components of each key finder module also undergo quality assurance activities through visual identification and automatic data capture at industrial camera and RFID reader work stations, respectively (refer demonstrator layout in Figure 1).

## 4. Integration of Ontology to Cloud based MES

When the manufacturer opts for the cloud based MES (CMES), the GeSCo is also shipped as part of the manufacturing solution suite. In order to provide ontological support, the MES vendor must put considerable effort into the ontology development process also known as ontology engineering. It includes the following steps: Determination of scope, enumeration of terms of taxonomy, encoding of the ontology in terms of definitions of classes, definitions of properties and constraints, instantiation of individuals, and design of ABox rules, ontology update and ontology enrichment. In cases of heterogeneous ontologies, ontology alignment/learning is another supplementary task. This effort with regards to ontology development and maintenance is additional to the maintenance of the main codelines of the MES and the GeSCo.

In addition to the ontological engineering task, the ontology should also be subjected to adaption, extension and/or reconstruction owing to corresponding changes in the development codelines of the MES and the GeSCo. Moreover, ontology enrichment which does not change the concepts and relations, but only refines the existing constraints also needs to be handled as a result of fine tuning the source code in MES. On the other hand, the delegating the ontology engineering task to the manufacturer is not a feasible solution either. In practice, the domain experts on the manufacturer's side who are involved in PO creation might not have sufficient first hand ontological development experience. Furthermore, the construction of ontologies using techniques of ontology engineering is also a time-consuming task. The fact that MES software development and maintenance, and ontology development require a very different skillset also make complementing the MES with an ontology an incompatible task. In such a scenario, the solution to the problem of ontology creation and maintenance to keep it *current* is to automate the process of encoding of the ontology.

There are several methods to achieve this process: text-based and machine learning are notable approaches. The ontology of the manufacturing resource and the related production information can also be extracted from the so called OPC UA collaborations (also called companion specifications). These open machine models are offered in two flavors: text based portable document format (PDF) and embedded links inside these PDF files to XML documents known as *UANodeSet* file. In agreement with the UANodeSet metadata, the MES software already implements machine models using this XML metadata file. These software modules are further used to represent the *digital twin* model required for condition monitoring from the cloud. Another area of application of machine models in MES is the design of a static routing plan. In such a scenario, the production designer who sets out to create an ordered list of manufacturing services to transform the raw materials to end products has to choose corresponding manufacturing resources that provide these services. At this point, he maps the methods provided by these machine models against the required manufacturing services of the PO. The authors believe that the source code of the cloud based MES that implements these *digital twins* is the right information model for ontology modeling for the reasons explained in the following:

- It is not realistic to assume that every manufacturing resource in the shop-floor possesses an OPC UA companion specification. However, irrespective of the availability of UANodeSet file, the MES implements machine models for the reasons stated above.
- It is also possible that a companion specification compliant manufacturing resource might have additional functionality and hence, the resource can have an enhanced information model which is not identical to the information model of the companion specification.
- Furthermore, there are also additional utility and calculated fields in the resource information model of MES that play a vital role in formulating decision-making rules.
- For an effective production control, it is required that the generated ontology covers all the concepts that play a role in the shop-floor such as manufacturing resources, manufacturing operations, product, its variants and sub-assemblies. However, the UANodeSet only contains the node information about the server representing the manufacturing resource type. The constructed ontology employing UANodeSet which only describes the manufacturing resources and their offered services is of little utility in the context of holistic production control. However, there exists corresponding source code in the cloud based MES that incorporates all the above-mentioned concepts of materials, resources, workstations, manufacturing operations, BOM, routing plan and PO. Hence, the source code of the MES must be employed to

generate a common reference ontology model of the different entities.
- Any change to the information model and/or production control logic in MES is immediately reflected in the ontology when the ontology generation algorithm is triggered on updated source code. This in turn enables the creation and the formal reasoning of production logic based on the latest version of the ontology and thereby substantially reduction of the latency of propagation of source code changes to the real time production.

The authors propose to exploit the machine model data in the cloud based MES to encode the ontology to serve the following purposes:

- Assignment of ontology concepts or/and production process ruleset to PO constituents so that formal reasoning is possible during the selection of suitable services in the manufacturing shop-floor
- Formulation of the business ruleset using the generated ontology that needs to hold true before or/and after a production step during PO execution runtime

## 5. Automatic Encoding of the Ontology

The machine model classes and interfaces that were implemented in the cloud based MES are considered for the automatic ontology translation process. High-level programming languages such as Java and OWL modeling language belong to different spheres of software engineering and ontology engineering respectively. Hence, an appropriate translation mechanism has to be devised to find the analogous concepts. The annotations on MES source code provide additional information to the encoding algorithm to construct the Subject-Verb-Predicate model of the OWL object properties. The annotations on the MES source code present an additional, but small overhead activity to the developers in the form of documentation that states the purpose and context for smooth code modifications and extensions in the future.

The output of this process is the generation of a formalized ontology file that corresponds to the concepts of manufacturing operation, product, routing and resources. The development of a formal description of inter-relationships between software artifacts is not the end goal. They merely provide a medium to the rule-based applications for the formal analysis of various concepts. In this research use-case, the generated ontology is used as a reference to construct the ruleset for the defined concepts and also to deduce the inferences. Arguing on the same lines, even though the transformation rules provide guidance to transform a class object to an OWL individual, the authors only generate the TBox ontology in practice. Later, during the description of concrete manufacturing services in shop-floor, in OWL-S framework for example, ABox ontologies are also created as instantiations of earlier generated TBox ontologies.

It is to be noted that the automatically generated preliminary ontology does not describe the intricate details of all the entities of the production process. It only describes the necessary concepts that are required for manufacturing automation through

Table 1. Rules to transform source code to formal ontology.

| **Source Code Attribute (SCA)** | **OWL Entity** | **Comments** |
|---|---|---|
| Project source code files | Ontology File | The complete ontology corresponding to the various machine model source code is stored into a single ontology file |
| Class | Class | Both the source code and OWL ontology have identical class concepts. The fully qualified name of the class is used in naming the ontology class to prevent ambiguity. |
| Primitive class member fields | Data Property | The primitive types in source code are translated to the corresponding *Datatype* in OWL |
| Primitive member field name | Data Property Name | — |
| | Domain and Range of Data Property | The encircling class in the source code is made the domain of the encoded data property. The encircled data type of primitive field is made the range of the encoded object property |
| Complex member fields/Composition/ Composite Aggregation | Object Property | The class and complex fields are related by *whole-part* relationship. The *part* fields are related to the *whole* by an object property |
| Binary Association / Aggregation | Two Object Properties | The associations are annotated with different values. Therefore, two object properties are encoded, and they are related by means of *inverse* relations in OWL. |
| Annotations on Association/Aggregation/ Composition fields | Name of the Object Property | The complex member fields are annotated with the object property name. This value is extracted at runtime using Java reflection. |
| Association / Aggregation / Composition | Domain of Object Property | The encircling class in the source code is made the domain of the encoded object property. |
| | Range of Object Property | The encircled complex data type is made the range of the encoded object property |
| | Cardinality of the Object Property | There is no general pattern in the object-oriented paradigm to represent this aspect of ontology. Hence, it does not figure in the ontology transformation process. |

| SCA | OWL Entity | Comments |
|---|---|---|
| — | ABox restrictions | If this step is subjected to automation, it reduces the reusability and also increases the development effort in the long run. Hence, the necessary production process and business ruleset has to be encoded in SWRL/SPARQL by a human expert. |
| Inheritance /Generalization (Implementation and Extension) | subClassOf restriction | The interface and classes that a class inherits in source code are translated to OWL subClassOf restriction. |
| | DisjointWith restriction | In the absence of inheritance in source code, all the OWL classes are marked disjoint with other classes. However, for the sake of simplicity, this fact is ignored in ontology. |
| | EquivalentTo restriction | The classes in the source code are unique, and hence, the case of equivalent classes in OWL does not arise. |
| Class Instance | OWL Individual | The instantiated class object in source code is equivalent to the OWL type individual. |
| Class Methods | Equivalent SWRL rules | There is no behavioral concept of entity that performs data transformation in OWL. Instead, this shortcoming is overcome by the formulation of SWRL rules. |
| Suitable Annotation on source code classes and fields | Annotations on TBox and ABox entities | The OWL annotation properties such as comments, label, deprecated, versionInfo et cetera are represented in suitable in annotations in source code. |
| — | Class necessary restrictions | The object-oriented concepts only support IS-A, whole-part, has-a relationships between classes. There exist no further constructs in OOPS to express further formal restrictions that can be placed on OWL classes. |

automatic resource discovery based on the offered methods and their characteristics, and support the formal design of the business ruleset. In contrast, manually built ontologies are much larger and more complex which might not necessarily be an advantage. In practice, only a small portion of an ontology is usually reasoned on business ruleset and automatic resource discovery. The larger ontology also necessitates larger processing times that goes against the principles of high speed manufacturing. To that end, a relatively small ontology is generated containing around 750 ontology concepts that are relevant to only the key finder unit PO at hand. This generated ontology is distributed among all the actors of the manufacturing system namely CMES based applications and GeSCo. This ontology is the underpinning for the construction of manufacturing resource ontologies. These in turn formally describe the various methods and the corresponding manufacturing services offered by these methods employing either OWL-S or SAWSDL framework. These frameworks facilitate accurate discovery and usage of resources and their methods. The complete process of ontology generation to formulation of production and business rules in terms of SWRL, SQWRL and SPARQL specifications [14] is shown in Figure 2. Additionally, the terminological knowledge expressed by the generated ontology is exploited to construct SWRL/SPARQL rules so that the design and manufacturing knowledge can be combined to formulate complex business constraint rules and inference rules. These rules that are slotted in between the various production steps provide clear and event-based guidelines to the production orchestrator and hence, facilitates smooth process automation. Based on the classes and properties modeled with formal restriction that have been defined in the generated ontology, the production designers model the PO and write business rules. The business ruleset offers enhanced expressiveness to the PO. In contrast to centralized automation, where processes are explicitly specified, this approach specifies the processes and control of production logic through interrelated production and business rules. Another use case of SWRL rules can also be the automated validation of a designed ontology. The SWRL rules that are formulated with

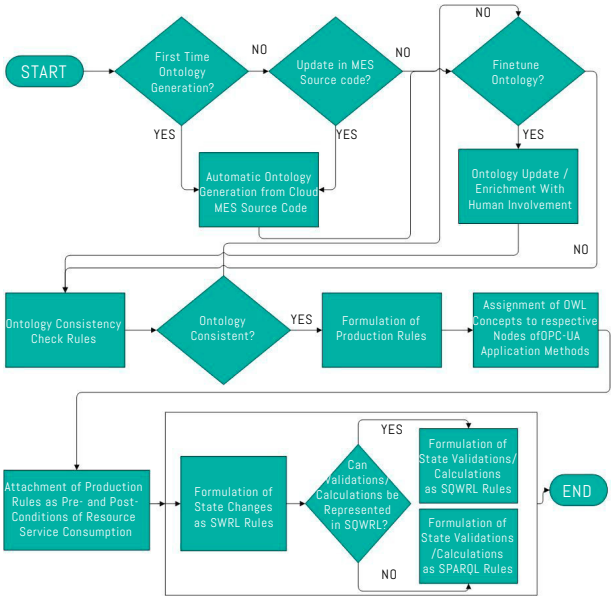


Fig. 2. Complete Cycle of Automatic Ontology Generation and Formulation of Rules with Human Involvement at Production Design Time.

such *assert* statements corroborate the consistency of the generated ontology.

## 6. Formulation of Ontology Resolution and Business Rules in SWRL, and Match-making Algorithms

When the common reference ontology model has been generated, the production designer can import the ontology file into Protégé editor. At this stage, the production designer, if familiar with ontology engineering, can enhance the ontology by adding or fine-tuning the automatically generated ontology. Such ontology resolution rules are shown in Figure 3. The traditional application of SWRL is the creation of data and object property assertions, and inference about the presence of individuals belonging to an OWL ontology class. Figure 4 provides examples of a few business rules involved in production of the key finder. SWRL rules can be attached to the reference ontology file in one of the many supported syntaxes such as RDF/XML format or they can be constructed and consumed programmatically on the fly in plain text form. It is recommended to consume the SWRL rules programmatically [1] and the argument becomes even more applicable if the parameters to be fed to the rules are generated at program run time.

The generated common ontology is pushed to the production network in general, and to the GeSCo in our research use case. The OPC UA servers of the manufacturing resources are then formally modeled using the SAWSDL or the OWL-S framework on top of the automatically encoded ontology. The various aspects, configurations and capabilities of the manufacturing resources are instantiated into OWL individuals using OWL constructs such as *equivalent, subClassOf, Instance, sameIndividual* and *DifferentIndividual*. The generation of this ABox ontology cannot be generated in an unsupervised environment and hence requires the involvement of a human expert. The human expert also formulates the SWRL rules that depict the

```
Rule 1:
rescom_resources_AutonomousTransporter(?at) ^
rescom_capability_abstracts_Picker(?m) ^
rescom_capability_abstracts_Placer(?m) ^ hasPicker(?m, ?p)
^ hasPlacer(?m, ?pl) -> rescom_concepts_Displacer(?m)

Rule 2:
rescom_capability_concrete_MaterialSupplier(?ms) ^ supplies(?ms, ?mat) ^
rescom_product_LowerShell(?mat) -> rescom_resources_LowerShellProvider(?ms)

Rule 3:
rescom_capability_concrete_MaterialSupplier(?ms) ^ supplies(?ms, ?mat) ^
rescom_product_UpperShell(?mat) -> rescom_resources_UpperShellProvider(?ms)

Rule 4:
rescom_capability_concrete_MaterialSupplier(?ms) ^ supplies(?ms, ?mat) ^
rescom_product_CircuitBoard(?mat) ->
rescom_resources_CircuitBoardProvider(?ms)
```

Fig. 3. Fragment of Ontology Resolution Rules for ResCom Demonstrator Ontology.

```
Rule 1:
rescom_product_LowerShell(?lowerShell) ^ hasColor(?lowerShell, White)
-> isIdentified(?lowerShell, true)

Rule 2:
rescom_product_LowerShell(?lowerShell) ^ isPlacedInAssembly(?lowerShell, true)^
rescom_product_CircuitBoard(?circuitBoard) ^ isPlacedInAssembly(?circuitBoard, true)
^ rescom_product_UpperShell(?upperShell) ^ isPlacedInAssembly(?upperShell, true)
-> sqwrl:select(true)
```

Fig. 4. Examples of Business Rules for the Key Finder PO.

pre- and post-conditions on the product, resource, transportation facilities, PO and the environment on account of provision of manufacturing service by the manufacturing resource. With this information at hand, we can make the paradigm shift from machine driven production definition to product driven control processes. A web application tool is hosted on the cloud based MES and was created as part of this research prototype to capture the OWL-S ontology of each of the methods of the manufacturing resources based on OPC UA servers. It allows to create an OPC UA client session to connect the HTTP WS-* SOAP-based resource servers of key finder demonstrator that are wrapped by a single OPC UA server for the sake of simplicity. The tool captures the information of various aspects of a manufacturing service provided by an OPC UA method in terms of the semantic concepts of the reference ontology. This information is exploited to create a corresponding OWL-S ontology which contains the profile, process model and grounding ontologies with regards to a specific method of the OPC UA server of a resource (refer [9] for details) using OWLAPI and cached at the production network for local manufacturing service discovery. The required manufacturing service for each of the production steps of the PO is compared to the offered services in these OWL-S ontologies and a best fit service is chosen to execute a manufacturing operation corresponding to the production step.

As an alternative to automatically generating an OWL-S ontology, the SAWSDL specification extended to OPC UA application methods can also be used as a substitute for the purpose of dynamic production orchestration [7]. Though the SAWSDL specification reduces the degree of match of a particular manufacturing service, it simplifies the process of finding the right manufacturing services. For the sake of completeness, this research evaluates the efficacy of automatically generated ontology and the subsequently constructed ruleset in orchestrating the key finder production workflow using an OWL-S ontology.

During production runtime, the generated ontology also serves the purpose of performing data collection and stores the PO specific data in PO individual entity of the ontology. GeSCo does not report the result of each of the production steps to the central cloud MES during production control, but instead it accumulates all the data in the cached ontology. At the end of PO execution, the entire ontology is uploaded to the cloud based MES. The MES performs the reverse engineering of transforming the ontology to the corresponding instance of the Java classes and invokes the appropriate transactions to store the collected data into CMES information systems. Instead of transmitting the raw data results after the completion of every production step, this technique provides a platform for GeSCo to filter and analyze the data at the factory network level, and hence, substantially reduces traffic to the cloud.

## 7. Implementation

A simulation cloud based MES is developed over the course of research which also contains the information models of the manufacturing resources of the shop-floor in the form of Java

source code. The demonstrator modules are designed as OPC UA servers which communicate with the Programmable Logic Controllers (PLC) to control the kinematics of the mechanical manufacturing resources. These resources do not conform to any of the OPC UA *collaboration types*. However, the use case is executed under the pretense that the information model of manufacturing resource recovered from the source code of the cloud MES is designed based on the resource *UANodeSet* file. The subsequent step is the automatic generation of manufacturing ontology of the demonstrator based on the simulated cloud MES source code with the aid of OWLAPI. The rules laid out in Table 1 are followed for this ontology transformation. The jar files corresponding to the source code of information model of demonstrator modules are fed to the automatic ontology generation application which also imports OWL-API. Using Java reflection concepts, the source code is analyzed and the corresponding ontology is generated in the form of OWL classes, object and data properties, and similar class expressions.

The next step is the formulation of the production and business rulesets. In addition to the SWRL rules, SQWRL queries which gather information and compute aggregations without writing back the results to ontology are also formulated. Such SQWRL queries are suitable to compute the rule validation result of the current production step and apply it directly in the application without the need to store it in a result variable. Both the SWRL rules and SQWRL queries were executed using the Java based SWRLAPI [12], version 2.0.5. SWRLAPI comes with two major advantages over the other popular open source reasoners such as Hermit [6]. SWRLAPI contains implementations to a number of built-in libraries such as temporal built-ins, mathematical built-ins, extensions built-ins, and string, boolean and date built-ins and thereby largely increases the expressivity of the rules. It also provides a powerful extension mechanism to define user-defined built-in libraries. Another advantage of SWRLAPI is the provision of built in libraries for TBox and RBox ontologies using SQWRL to query all the OWL axiom types of an ontology [15].

However, both SWRL and SQWRL do not support deep and nested querying features that are required to compare varying ontology entities and their corresponding property values. For instance, there are three different assembly resources that provide similar *press* manufacturing services in the demonstrator. A rule needs to be constructed that determines the most suitable press resource based on the least effort in terms of distance to fetch different raw materials to the press assembly taking into the account various intermediate quality assurance station visits in the production of intelligent key finder. This rule involves the creation of nested conditions to compare the efforts for the three assembly resources which cannot be achieved with a single SQWRL query. In such cases, the rules are written as SPARQL queries. The SPARQL query that computes the least effort in terms of distance involved among the three assembly resources is shown in Figure 5. This query only computes the distances along X-axis. Similar queries are run to find distances along Y- and Z-axes. In implementation, these three rules corresponding to the X, Y and Z axes are merged into a single query, and it is not shown owing to space constraints. At the end, the

```
PREFIX : <http://emea.global.corp.sap/rescom#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?PressInstance
    ((abs(?Press_XCoOrd_Value - ?LowerShellWarehouse_XCoOrd_Value))
    +(abs(?Press_XCoOrd_Value - ?UpperShellWarehouse_XCoOrd_Value))
    +(abs(?Press_XCoOrd_Value - ?CircuitBoardWarehouse_XCoOrd_Value))
    +(abs(?Press_XCoOrd_Value - ?Camera_XCoOrd_Value))
    +(abs(?Press_XCoOrd_Value - ?RFIDReader_XCoOrd_Value)) as ?CumulativeDistance_X_Direction)
    WHERE{
      ?LowerShellWarehouseInstance :hasPositionX ?XCoOrd_LowerShell_Inst;
                                   rdf:type :rescom_resources_LowerShellProvider.
      ?XCoOrd_LowerShell_Inst :Value ?LowerShellWarehouse_XCoOrd_Value.

      ?UpperShellWarehouseInstance :hasPositionX ?XCoOrd_UpperShell_Inst;
                                   rdf:type :rescom_resources_UpperShellProvider.
      ?XCoOrd_UpperShell_Inst :Value ?UpperShellWarehouse_XCoOrd_Value.

      ?CircuitBoardWarehouseInstance :hasPositionX ?XCoOrd_CircuitBoard_Inst;
                                     rdf:type :rescom_resources_CircuitBoardProvider.
      ?XCoOrd_CircuitBoard_Inst :Value ?CircuitBoardWarehouse_XCoOrd_Value.

      ?IndustrialCameraInstance :hasPositionX ?XCoOrd_Camera_Inst;
                                rdf:type :rescom_resources_IndustrialCamera.
      ?XCoOrd_Camera_Inst :Value ?Camera_XCoOrd_Value.

      ?RFIDReaderInstance :hasPositionX ?XCoOrd_RFIDReader_Inst;
                          rdf:type :rescom_resources_RFIDReader.
      ?XCoOrd_RFIDReader_Inst :Value ?RFIDReader_XCoOrd_Value.
      {
        SELECT ?PressInstance ?Press_XCoOrd_Value
        WHERE{  ?PressInstance :hasPositionX ?XCoOrd_Press_Inst;
                               rdf:type :rescom_capability_abstracts_Assembler.
                ?XCoOrd_Press_Inst :Value ?Press_XCoOrd_Value.}
  }} ORDER BY ASC(?CumulativeDistance_X_Direction) LIMIT 1
```

Fig. 5. SPARQL Query to find suitable Assembly Resource with Least Distance Coverage in X-Axis.

suitable assembly unit is chosen based on least distance coverage in all the 3 directions. The advantage of writing such simple to moderate sized rules is avoidance of writing complex source code in a high level language such as Java to realize the same objective. Furthermore, such rules neither presume any knowledge of presence of specific resources on the shop-floor nor require any hard-coded values for computations/validations. The dispatching of PO to the shop-floor network at GeSCo and PO execution at production runtime including resource orchestration using the OWL-S or the SAWSDL frameworks is explained in our previous research [8].

## 8. Lessons Learned

During experimental evaluation, it was observed that the creation of a new instances of an OWL ontology manager and an OWL reasoner, creation of SWRL, SQWRL and SPARQL query engine instances, and loading the ontology documents from the file system take an inordinate amount of time. In production runtime, this impacted the resource discovery and rule validations negatively in terms of processing times. Therefore, REST based web services that provide OWL reasoning and inference services, and execute SWRL/SQWRL/SPARQL rules/queries are implemented using a singleton software design pattern in a way it does not affect the concurrency of the whole system. This permitted in-memory computation of ontology processing, querying and rule validations, and consequently brought down the average response time of these web services from one second to under 100 ms. The research use case on key finder demonstrator does not contain a large number of parallel processes. Even for a production scenario involving parallel processes, the waiting times due to a small queue

of requests to these singleton OWL reasoners and rule engines outweigh the drawbacks of large response times owing to creation of these instances on *per service call* basis.

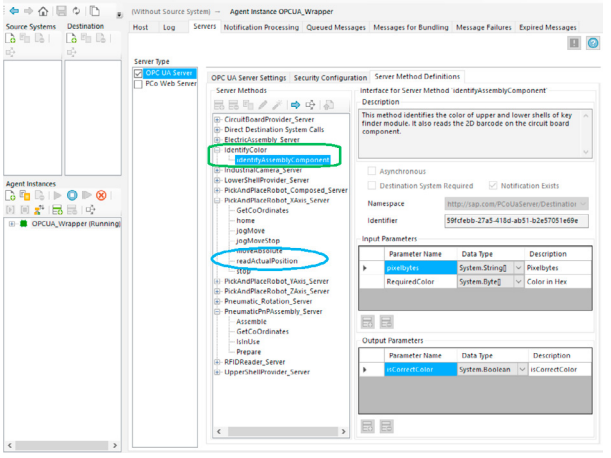

Fig. 6. OPC-UA Wrapper for HTTP based ResCom servers, and Caching of source code for validations involving large data processing (Highlighted in Green Color).

Though the presented rule-based approach covers most business scenarios, it was also discovered during evaluation that the semantic web languages fail in the processing of large amount of data and complex calculations. However, in retrospect, the semantic web languages were essentially conceptualized with only formal knowledge representation in mind. The research use-case has moderate data processing in the form of identification of colors of upper and lower shells of the key finder module and identifying a 2D barcode on the circuit board which is sandwiched between these upper and lower shells. The corresponding image processing source code was compiled and the resulting 6kb dynamic linked library (DLL) was cached along with the PO to GeSCo. The GeSCo is designed such that custom process logic that inherits its API classes can be attached to it as either OPC UA, SOAP, REST or ODATA based WS-* servers (refer Figure 6) via the design principle of *dependency injection*. Alternatively, when the source code size for the production step validation is too large to cache in the production network, it is inevitable to request the cloud-based business applications such as MES for rule validations and determination of the next step of production.

## 9. Conclusion and Future Work

In order to avoid burdening the manufacturer purchasing the MES with the task of ontology encoding, this paper describes a semi-automatic bidirectional transformation of MES source code to a corresponding reference ontology taking into account the OPC UA collaborations. With human involvement, various production and variant-based business rules are formulated on top of the reference ontology with an intent of reuse. These rules make up for the information loss suffered in the transformation process. These rules are attached to various production steps of PO and the entire PO is cached at the production network in GeSCo to facilitate decentralized production. The generated ontology is also cached to the shop-floor and it is only updated in the event of change of source code in the cloud based MES. The combination of local access to an ontology generated from a centralized system and rule-based behavior definition in an abstract PO empowers GeSCo with increased variability in production control and hence, enables to create a flexible production orchestration plan which is resistant to turbulences in the production.

## References


[1] Ameri, F., Urbanovsky, C., McArthur, C., 2012. A systematic approach to developing ontologies for manufacturing service modeling, in: Proc. 7th International Conference on Formal Ontology in Information Systems (FOIS 2012), Graz, Austria, Citeseer.

[2] Blomqvist, E., 2009. Semi-automatic ontology construction based on patterns. Ph.D. thesis. Linköping University Electronic Press.

[3] Bontcheva, K., Sabou, M., 2006. Learning ontologies from software artifacts: Exploring and combining multiple sources, in: Workshop on Semantic Web Enabled Software Engineering (SWESE), Athens, GA, USA.

[4] De Paepe, D., Thijs, G., Buyle, R., Verborgh, R., Mannens, E., 2017. Automated uml-based ontology generation in oslo 2, in: European Semantic Web Conference, Springer. pp. 93–97.

[5] Guerson, J., Sales, T.P., Guizzardi, G., Almeida, J.P.A., 2015. Ontouml lightweight editor: a model-based environment to build, evaluate and implement reference ontologies, in: Enterprise Distributed Object Computing Workshop (EDOCW), 2015 IEEE 19th International, IEEE. pp. 144–147.

[6] HermiT OWL Reasoner, 2018. HermiT OWL Reasoner. URL: http://www.hermit-reasoner.com/ [Date: 2018-11-28].

[7] Katti, B., Plociennik, C., Ruskowski, M., Schweitzer, M., 2018a. Sa-opc-ua: Introducing semantics to opc-ua application methods, in: 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE), IEEE. pp. 1189–1196.

[8] Katti, B., Plociennik, C., Schweitzer, M., . A jumpstart framework for semantically enhanced opc-ua. KI-Künstliche Intelligenz , 131–140.

[9] Katti, B., Plociennik, C., Schweitzer, M., 2018b. Semopc-ua: Introducing semantics to opc-ua application specific methods. IFAC-PapersOnLine 51, 1230–1236.

[10] Katti, B. and Plociennik, C. and Schweitzer, M., 2018. GeSCo: Exploring the Edge Beneath the Cloud in Decentralized Manufacturing. International Journal On Advances in Systems and Measurements v11,1&2, 183–195.

[11] Küçük, D., Arslan, Y., 2014. Semi-automatic construction of a domain ontology for wind energy using wikipedia articles. Renewable Energy 62, 484–489.

[12] O'Connor, M.J., Shankar, R.D., Musen, M.A., Das, A.K., Nyulas, C., 2008. The swrlapi: A development environment for working with swrl rules., in: OWLED.

[13] Song, Q., Liu, J., Wang, X., Wang, J., 2014. A novel automatic ontology construction method based on web data, in: Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP), 2014 Tenth International Conference on, IEEE. pp. 762–765.

[14] SPARQL Protocol and RDF Query Language, 2018. SPARQL Protocol and RDF Query Language. URL: https://www.w3.org/TR/rdf-sparql-query/ [Date: 2018-11-28].

[15] SWRL TBox Built In Library, 2018. SWRL TBox BuiltIn Library. URL: https://github.com/protegeproject/swrlapi/wiki/SWRLTBoxBuiltInLibrary/ [Date: 2018-11-28].

[16] Vrandečić, D., 2009. Ontology evaluation, in: Handbook on ontologies. Springer, pp. 293–313.

[17] Yang, K., Steele, R., Lo, A., 2007. An ontology for xml schema to ontology mapping representation. Information Integration and Web-based Applications and Services (2007) .