Semantic Degrees for Industrie 4.0 Engineering

Deciding on the Degree of Semantic Formalization to Select Appropriate Technologies

Chih-Hong Cheng, Tuncay Guelfirat, Christian Messinger,
Johannes O. Schmitt, Matthias Schnelte, Peter Weber
ABB Corporate Research Center, Germany
{firstname.lastname}@de.abb.com

ABSTRACT

Under the context of Industrie 4.0 (I4.0), future production systems provide balanced operations between manufacturing flexibility and efficiency, realized in an autonomous, horizontal, and decentralized item-level production control framework. Structured interoperability via precise formulations on an appropriate degree is crucial to achieve software engineering efficiency in the system life cycle. However, selecting the degree of formalization can be challenging, as it crucially depends on the desired common understanding (semantic degree) between multiple parties. In this paper, we categorize different semantic degrees and map a set of technologies in industrial automation to their associated degrees. Furthermore, we created guidelines to assist engineers selecting appropriate semantic degrees in their design. We applied these guidelines on publicly available scenarios to examine the validity of the approach, and identified semantic elements over internally developed use cases concerning plug-and-produce.

Categories and Subject Descriptors

D.2.12 [**Software Engineering**]: Interoperability; I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems – *industrial automation*

Keywords

Embedded software, design space exploration and evaluation guidelines, Industry 4.0 engineering, semantic degree

1. INTRODUCTION

Industrial manufacturing companies are facing strong demands to improve their production process, not only on the shop-floor but throughout the complete value chain. These demands arise from requests such as growing productivity expectations, increasing number of product variants, reducing lot sizes, etc. It is widely perceived that new information technologies will reshape production processes via an integration into existing industrial automation and communication technologies, from engineering to commissioning to operation. The activity of Industrie 4.0 (I4.0, or Industrial Internet of Things) is such an initiative to apply internet of things (IoT) technologies to the manufacturing context.

Nevertheless, to enable automated interpretation and processing of the interchanged information throughout the enterprise (from machines to services), apart from basic physical communication and data transmission, having a *commonly understood information for exchange* is a premise. We refer the degree of common understanding as the *semantic degree*. As realization of I4.0 is largely via software technologies, a predefined semantic-degree governs the underlying software engineering and commissioning (e.g., implementation or configuration) efforts, as it is highly associated with the amount of information to be revealed on the functional interface. Therefore, to agree on a proper semantic degree is crucial in early stage system design. The risk of improper "selection of semantic degrees" can be observed by two extremes: On one side, choosing a degree of no common understanding implies that each component needs to implement its own parser to understand the meaning of other components. On the other extreme, having a degree where each component understands the real world (in terms of physical equations) simply contains unnecessary details and incurs huge reasoning effort.

In this paper, we categorize the exposed semantic degree (Section 2) by first considering whether the exchanged information is structural or behavioral (i.e., the receiver needs to know the system configuration which evolves over time). For structural information, semantic degrees range from simple document repositories to ontologies, while for dynamic information, finite automata, Petri nets, or genetic programming language are used. With predefined degrees, we further identify a set of guidelines that can be used as a filtering procedure (Section 3), enabling engineers to select appropriate semantic degrees in their system design.

In our evaluation (Section 4), we first review commonly seen technologies used in I4.0 and associate these technologies with concrete semantic levels. We then present internally-developed use cases from different industrial segments which are transformed into future I4.0 scenarios, and depict the underlying rationale of selecting appropriate semantic degrees in the underlying software development process. Lastly, we present our examination over 20 publicly available "I4.0 demonstrators" and identify the required semantic degrees in order to fulfill the implemented features.

2. CATEGORIZATION CONCEPT

In order to categorize semantic degrees, we first distinguish between structural and behavior modelling. While structural modelling describes how (in which detail) information is exposed, behavioral modelling describes what kind of information is contained and knowledge about the information processing. These two aspects can be seen as two dimensions with certain degrees.

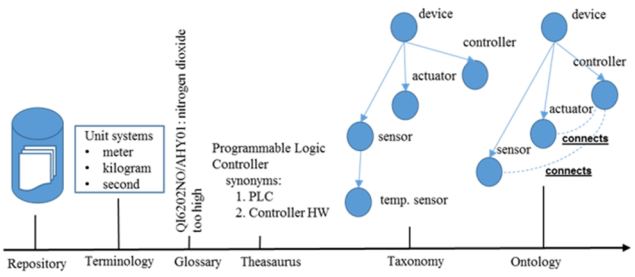

Figure 1: Degrees of structural formalization, with industrial automation examples

ESEC/FSE'15, August 30 – September 4, 2015, Bergamo, Italy
ACM. 978-1-4503-3675-8/15/08...$15.00
http://dx.doi.org/10.1145/2786805.2804434

Based on the demands of transparency and collaboration that one wants to achieve in the system, these degrees help engineers to select an appropriate modelling method and technology.

Structural formalization: Our categorization of structural information (Figure 1) is based on a refined set of criteria from the work of Navigli and Velardi [12], ranging from document repository to ontology. Table I provides their exposed information and some applied industrial contexts. Notice that our definition of the degree elements (e.g., glossary) are slightly altered to fit into the industrial automation setup.

Starting from the lowest degree of formalization, a **Repository** can be seen as a source for raw data (either structured or unstructured) without additional semantics. This means that additional knowledge is needed for interpreting the information. A **Terminology** provides a set of vocabularies that is implicitly controlled, agreed and specialized for the system under investigation – which means that a term is well defined and unique. This makes it possible to annotate elements with tags in order to make information out of data. A data point for example can hold the value: [33.3] and provide an additional tag: [Celsius] – a controlled application of this tag allows for comparison with other values of the same kind. A **Glossary** is an explicit, specialized list of words and their associated definitions. Within industrial contexts, it can be used for extending elements (tags) with a human readable description, such as associate an alarm tag with its actual meaning that can be shown in an HMI (e.g., associate tag QI6202NO/AHY01 with meaning "nitrogen dioxide too high,,). Importantly, we assume that the use of basic words such as measurement units (e.g., Kilogram, Meter), are considered as a common knowledge in terminology. However, these words do not need to be listed in a glossary, because of their roots in the common-sense.

TABLE I. FEATURES PROVIDED BY STRUCTURAL DEGREE

Structural Degree	*Exposed Information*	*Industrial Context*
S0: Repository	Unstructured/structured data	Devices catalogues, file structure,
S1: Terminology	Controlled vocabulary	Tags, annotations
S2: Glossary	Description over vocabularies	Human readable documentation, HMI
S3: Thesaurus	Basic relationships (association), similarities	Profile mapping, technology Integration
S4: Taxonomy	Tree Structure, parent-child relations, classifications	Abstraction (typing), device classification
S5: Ontology	Typed elements and relations	Interfaces, inheritance, topologies

A **Thesaurus (**or a Topic Map**)** is used to describe similarities between tags and to assign a single, well-defined term to all occurrences (e.g. from [Temperature Sensor] to [Bluetooth-Device_000A3A58F310]). It can also be used to map different technology terms from different vendors to avoid misunderstandings. **Taxonomies** are used to define parent-child relations and to build up tree structures, which are used in structuring device classes (e.g. a dimmer is a switch, a switch is an actuator). In contrast to taxonomies, an **Ontology** can have a full mashed topology. An ontology provides typed elements, references and enables the definition of type structures – which can be applied for the definition of interfaces as known from object models. Objects or object models (e.g. as used for the Common Information Model – CIM [3]) can be also represented by an ontology (however an object model typically has a fixed set of reference types).

Behavioral formalization: The entries in TABLE II. describe behavioral aspects which may be required to be exposed by a certain scenario in the industrial context[1]. Contents from B0 to B2 are more static – they are basic ingredients used to record the snapshot of the system and to specify system invariants (conditions where the system should hold in certain stages), while contents from B3 to B6 target to expose dynamic information, i.e., they are used when understanding how system evolves over time is needed.

The basic behavioral degrees are provisioning of **Data** and **Information**. While the first degree provides only values, the second extends the first with a meaning, which is needed to make information out of data. Data can also include so called BLOBS (binary large objects) - where also complex information can be contained but however without any (machine interpretable) relation to other parts of the model. **Constraints** can be used to formalize logical requirements over the information such that they be evaluated (to true or false) during commissioning or operation in finite amount of time[2]. E.g., ConNO > THREA is a predicate checking if the concentration of Nitro-Dioxide (ConNO) exceeds a pre-defined threshold value (THREA). In an industrial context, a system can use constraints to specify logical policies for their installation (e.g. size, energy consumption) or structural policies for their integration (e.g. allowed sub-modules or connectivity to other systems). By exposing constraints, one is able to perform automatic reasoning using state-of-the-art tools like SMT [5], SAT [2], or DL[11] reasoning solvers.

TABLE II. FEATURES PROVIDED BY BEHAVIORAL DEGREE

Behavioral Degree	*Exposed Information*	*Industrial Context*
B0: Data	Values	Data Points
B1: Information	Tags	Self description, Data types
B2: Constraints (predicates over information) as logic formula	Requirements, properties	Structural, logical policies, conditions to trigger alarms, simple logic (if-then-else rules)
B3: Finite automata	States, events	State machines
B4: Petri Nets	Signals, concurrencies	Concurrent/distributed processes, queues
B5: Programming Language	Flexible function set	Scripting, coding
B6: Integrated simulation model	Physical and logical behavior combined	Virtual commissioning, run-time optimization (e.g., MPC)

In some cases, exposing the information as a **Finite automaton (FA)** can become relevant – e.g. if a machine is supposed to expose its current state and potential state-changes to preceding or subsequent machines in order to support them to react accordingly. The state of a finite automaton, as presented in the Kripke structure, can actually be evaluations over atomic propositions where each atomic proposition is a constraint that can be algorithmically evaluated. The degree of **Petri Nets (PN)** becomes relevant if distributed or parallel processes have to be coordinated through the information model because of shared resources. The almost highest degree for the description of behavior is the use of a (general purpose) **Programming Language (PL)**, whose underlying model of computation can be viewed as Turing machines.

Still, for modeling and correctly interacting with physical environments, it is well known that models such as FA or PN lack the explicit notion of time [10]. An **Integrated Simulation Model**

[1] The selection of elements in the behavioral degree is based on whether one can easily find appropriate industrial contexts. E.g., one can also insert pushdown automata in the behavioral degree, but as pushdown systems are rarely used, it is thus not listed.

[2] Precisely, we only consider fragments of logic having decidability results. Thus, generic first-order logic (FOL) is considered as inappropriate, while decidable fragments such as description logic [11]; used in OWL reasoning) is allowed.

(ISM) combines the logic of a component like the programmable behavior and the surrounding physical environment. Examples include Modelica [8] or Ptolemy II [6]. With this degree, a system can derive the behavior of the physical process and use this knowledge in automatic decision support, such as the run-time optimization via model-predictive control (MPC) [4].

Integration of structural and behavioral modelling: Structural and behavioral modelling have to be integrated in order to allow for the combination of their features. For example *constraints* from the behavior model can make use of *tags* defined in the structural model – a climate control can restrict its input for temperature sensor values to [Celsius]. As another example a*utomata* in the behavior model can be linked with sensor values of a specific data type considering inheritance described by *ontologies* in the structural model.

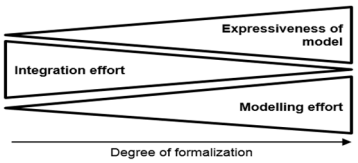

Figure 2: Tradeoff between modelling and integration efforts

TABLE III. EXAMPLARY GUIDELINES FOR SELECTING THE APPROPRIATE DEGREES

Rule	*Minimal degree when the answer is positive*	*Argument*
R0: Is the scope of the system very limited? E.g. one vendor, only few entitities, static setup	S0: Repository B0: Data	Hard coding with less effort than modelling
R1: Have multiple parties the need to exchange standardized knowledge - which can be intuitevly understood (such as units) ?	S1: Terminology B1: Information	Necessity of well defined terms.
R2: Have multiple parties the need to coordinate the use of terms.	S2: Glossary B1: Information	Human readable description needed for a common understanding
R3: Is it necessary to integrate definition of terms of other parties	S3: Thesaurus B1: Information	Mapping of different definitions using a Thesaurus
R4: Should the system provide a basic type system and be extensible in terms of lately added types?	S4: Taxonomy B1: Information	Parent-Child relations needed to classify types.
R5: Should the system be dynamic and extensible – e.g. allow for modelling of new elements during runtime?	S5: Ontology B1: Information	Even the meaning of a relationship can be modelled
R6: Is it required to validate evolving configurations during runtime?	S1: Terminology B2: Constraints	Modelling of requirements and a controlled vocabulary needed
R7: Shall a reasoning process involving multiple parties be supported?	S5: Ontology B2: Constraints	Necessity to describe complex and evaluable relationships
R8: Is it necessary to understand/modify the functionality (e.g. logic) of another system?	S1: Terminology B3:Automata (and can up to B5)	Machine interpretable description of logic required

3. SELECTING APPROPRIATE DEGREES

In our semantic degree formulation, usually a model can be mapped into a higher degree without the loss of information. The question now could be – why not simply take the highest degree? As shown in Figure 2, information modelling is a tradeoff between modelling and integration efforts. One would like to reduce the cost of one-time modeling efforts, while the result of modeling is still sufficient to allow efficient integration and commissioning, which can appear multiple times.

The decision process for the appropriate structural and behavior modelling starts with the analysis of the requirements of the application scenario. We provide some generic and exemplary guidelines in TABLE III, such that an engineer can use it to analyze its application scenario and derive the appropriate degree quickly. Notice that when all elements that are unique or rarely used, creating a detailed modeling for information exchange turns inefficient (see also Rule R0).

4. EVALUATION

In this section, we first give a summary of technologies in industrial automation and their corresponding semantic degrees. Then we give an example how . Lastly, we present a summary of 20 publicly available demonstrators and required semantic degrees. Due to space limits, we refer readers to the extended version [1] for complete details.

Evaluating technologies in industrial automation For our proposed semantic degrees, we have examined existing (software-related) technologies in industrial automation and associate each technology with the corresponding degree. The result is shown in Table IV; it can be used by software engineers in industrial automation to quickly filter technologies to be used in their I4.0 projects. E.g., if CAEX is used during the communication of I4.0 entities, an ontology system is communicated underneath.

TABLE IV. TECHNOLOGIES IN AUTOMATION AND THEIR CORRESPONDING SEMANTIC DEGREES

Automation Technologies		*Structural Degree*	*Behavioral Degree*
GSD/GSDML/CFF/IODD/ESI/SCL		S1: Terminology	B1: Information
MQTT		S0: Repository	B0: Data
AutomationML	Collada	S1: Terminology	B1: Information (3D) B6: ISM (due to kinematic definition)
	PLCOpen	S5: Ontology	B3: finite automata
	CAEX	S5: Ontology	B2: Information
STEP		S1: Terminology	B1: Information
RFID		S0: Respository	B0: Data
ecl@ss		S2: Glossary	B1: Information
EDD		S2: Glossary	B1: Information
KNX		S3: Thesaurus	B1: Information
FDI		S5: Ontology	B1: Information
BACNet, Enocean / BT/ Zigbee Profiles		S3: Thesaurus	B1: Information
Hart / PNO / FF / SCL Profiles		S3: Thesaurus	B1: Information
RDF		S4: Ontology	B1: Information
RDFS		S5: Ontology	B2: Constraints (structural)
CIM		S5: Ontology	B1: Information
OPC UA		S5: Ontology	B1: Information B3: Finite automata (partly due to state machine/ filter)
OWL		S5: Ontology	B2: Constraints (structural+ logical)
Domain Specific Language (DSL)		S1: Terminology	B3:Finite automata (can go up to B6)
PackML		S1: Terminology	B1:Information, B3:Automata

Applying semantic degrees in designing intelligent manufacturing systems. We present an I4.0 internal case study where we describe the targeted problem, the underlying mechanism design, and summarize the corresponding rationale of choosing appropriate semantic degrees using our guidelines. We have further analyzed three internally developed examples in engineering automation software, following the similar pattern. These examples can also be found in the extended version [1].

Scenario: Semantically-enabled Plug-and-Sense In state-of-the-art PLC programming, a PLC program is able to access the data of the sensor via a global variable mapping process. However, the engineer must know how to fetch data from a sensor connected to the field bus, and later encode such information in the I/O mapping file. All these activities are tedious and error prone. Whenever an error appears in the I/O mapping process (e.g., one can accidentally misplace a pressure sensor with the adjacent temperature sensor), it is difficult to be detected. Furthermore, sensors need be configured

differently (e.g., units) in order to support different project setups. A solution to above problems is not only useful in PLC programming but also for general I4.0 big data scenarios, as companies now search for innovative methods to shorten the integration time to bring data from edge devices to the cloud.

Targeting above problems, we have designed a demonstrator system comprised of (1) web-based software services and (2) component libraries for embedded software to be implanted on existing sensors, actuators, and PLCs. The high-level setup is shown in Figure 3 with entities *master (PLCs)*, *slave (smart sensors or actuators)*, and *I4.0-service* as *logical* elements. For ease of explanation, we explain the use-case scenario of inserting a temperature sensor using the sequence diagram in Figure 4, where the sensor is only able to read the temperature in Fahrenheit.

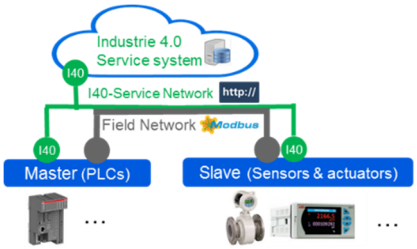

Figure 3: Intelligent devices supporting plug-and-sense, communicated under industrial (Modbus TCP) communication protocol

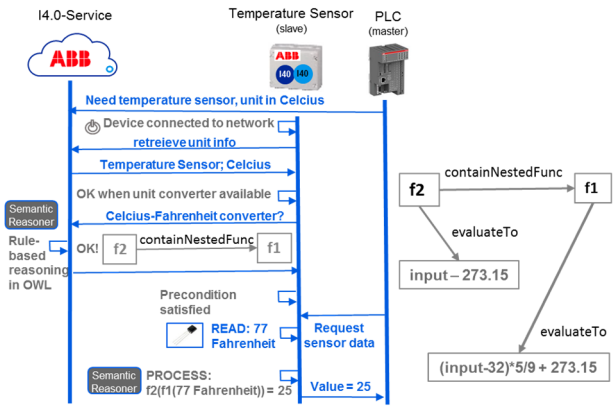

Figure 4: Sequence diagram when temperature sensor is inserted (left), and the returned ontological structure from as function composition (right)

Initially, by the time the engineer creates the I/O mapping, he also specifies what the connected device is, together with measurement units. Then the information is uploaded and stored to the I4.0-service. When a sensor is plugged to the network, it first queries the I4.0-service and examines whether itself is the desired device. For the sensor in Figure 4, it enters a conditional acceptance mode and queries the I4.0-service in order to obtain a unit converter.

The I4.0-service performs a *reasoning based on transitive closures* in order to fulfill the request. This is because I4.0-service only stores the three converters: (**f1**) from Fahrenheit to Kelvin, (**f2**) from Kelvin to Celsius, and (**f3**) from Celsius to Fahrenheit. The logic reasoned establishes the knowledge that to convert from Fahrenheit to Celsius, one can first apply **f1** on the sensed value, then apply **f2** on the previously computed value. The sensor thus receives two converters **f1** and **f2**, represented by the ontological structure in Figure 4 (right). By interpreting the ontology, the sensor converts 77 Fahrenheit to 25 Celsius via functional composition **f2**(**f1**(77 Fahrenheit)), before sending it to the PLC.

In the actual implementation, we applied our guidelines to select required semantic levels and corresponding technology ingredients.

Mininal Degree	***Applied Rule***
S5: Ontologies	R5: The system should be dynamic and extensible by new devices and converters R7: Reasoning should be supported (the returned two functions f2 and f1 should be connected by a ontological structure such that the sensor knows how to interpret it)
B2: Constraints	R7: Reasoning should be supported (the sensor sends a message: "find Celcius-Fahrenheit converter", which can be viewed as a logical constraint with existential quantification)

Evaluating publicly available I4.0 demonstrators Based on above mentioned guidelines, we examine 20 publicly available "I4.0 demonstrators" and identify the required semantic degrees to fulfill the implemented features Due to space limits, we refer readers to extended report [1] for details.

From our analysis, we observed that most demonstrators do not demonstrate intelligence and reasoning on the device or system level. Thus, scenarios using high semantic-degrees either structurally (e.g., ontology) or behaviorally (e.g., integrated simulation model) are limited, i.e., it is largely sufficient to use glossary (controlled vocabulary) in the implementation.

5. OUTLOOK

In this paper, we identified semantic degrees and provided guidelines for engineers to select appropriate semantic degrees in their Industrie 4.0 projects. Our definition is accompanied by concrete examples in industrial contexts, enabling automation software engineers to grasp the underlying concept. Our created guidelines allow engineers to quickly identify required semantic degrees for their projects, while our evaluation over existing technologies offers a single-stop for engineer to quickly understand semantic degrees behind commonly used technologies. Our investigation over publicly available demonstrators shows that most demonstrators are still largely far from intelligent, so that complex semantic degrees are not needed.

For future work, we will refine the rules and periodically update the table with new technologies and their associated semantic degrees. A decision support tool by utilizing the rules for engineering software systems in building automation is also under planning.

6. REFERENCES

[1] Full version available at: http://arxiv.org/abs/1505.05625

[2] A. Biere, M. Heule, and H. van Maaren, eds. *Handbook of satisfiability*. Vol. 185. IOS press, 2009.

[3] Common Object Model (CIM) http://www.dmtf.org/standards/cim

[4] E. F. Camacho and C. Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013.

[5] L. De Moura, and N. Bjørner. *Z3: An efficient SMT solver*. In: TACAS, pp. 337-340. Springer, 2008.

[6] J. Eker, J. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Sachs, Y. Xiong. *Taming heterogeneity - the Ptolemy approach*, Proceedings of the IEEE, 91(1):127-144, January 2003.

[7] A. Fisher, C. Jacobson, E. A. Lee, R. Murray, A. Sangiovanni-Vincentelli, E. Scholte. *Industrial Cyber-Physical Systems-iCyPhy*. In: CSD&M, Springer, pp. 21-37, 2013.

[8] P. Fritzson. *Principles of object-oriented modeling and simulation with Modelica 2.1*. John Wiley & Sons, 2010.

[9] International Journal of Knowledge and Learning 4.1 (2008): 93-108.

[10] E. A. Lee. *Computing needs time*. Communications of ACM 52(5): 70-79 (2009)

[11] D. Nardi, and R. J. Brachman. *An Introduction to Description Logics*. Description logic handbook. 2003.

[12] R. Navigli, and P. Velardi. *From glossaries to ontologies: Extracting semantic structure from textual definitions*. Ontology Learning and Population Knowledge (2008): 71-87 .