Case-based Formalization of Integration Knowledge supporting System Integrators to enable Automated Component Coupling[1]

Fabian Burzaff
University of Mannheim
Institute for Enterprise Systems
Mannheim, Germany
burzlaff@es.uni-mannheim.de

Christian Bartelt
University of Mannheim
Institute for Enterprise Systems
Mannheim, Germany
bartelt@es.uni-mannheim.de

***Abstract* — Using languages with formalized semantics for automating component integration is a well-established research area. As a consequence, independently developed software systems can interact without the need for manual integration effort in a "plug-and-play" manner. However, such dynamic adaptive system architectures are not widely used in industry automation scenarios. Practitioners mostly rely on informal, domain-specific standards as formal interface specifications tend to become highly complex quickly. Nonetheless, this results in high manual integration efforts as integration knowledge cannot be reused. Thus, interface specifications should be tailored towards its case-based requirements. Interface specifications should only be created evolutionary after specific integration tasks and persisted with knowledge management techniques. This makes integration knowledge available to be reusable by system integrators and can ultimately automate component integration.**

Keywords — Syntax, Semantic, Industry, Interface Specification, Interactive Semantic Integration

I. INTRODUCTION

The digitalization of industry (e.g. Industry 4.0/Industrial Internet of Things) requires dynamic and adaptive system architectures for meeting its vertical and horizontal interaction promises. Thus, semantic interface specifications have been applied in various contexts for automating service discovery, composition and executing [10][17][18]. However, this approach requires component suppliers to specify service specification for all use cases before automated service composition can take place. The strive for completeness often results in highly complex specification efforts in order to fully exploit automatic component interaction. Hence, practitioners rely on informal industry standards [1]. Although component suppliers do not need to formally specify every detail of the respective service interfaces upfront, this approach often results in individual point-to-point connections which must be maintained by systems integrators. Worse, integration knowledge is not persisted and consequently cannot be reused in an automated manner. Thus, a new approach is suggested that positioned itself between pure practical standardization initiatives and complete formal specifications for using interface specification languages:

1. A knowledge base should be extended step-by-step with formal semantic integration knowledge resulting from one specific manual integration task
2. The combination of several distinct formalized manual integration tasks serves as a basis for knowledge reuse automating future component integration
3. The completeness criteria for interface specifications is relaxed to moderate specification complexity

Integrating different systems may become a dominant task in the application area of Industrial Internet/Industry 4.0. Especially in the domain automation of production process, various physical machines equipped with software interfaces (e.g. Cyber-Physical Systems [2]) must be connected in a syntactically and semantically correct way. Especially, the semantics of services and information are expected to play a key role in future industry architecture [4]. This is commonly evaluated in research addressing system interoperability issues [3]. For example, system integrators ensure syntactic interoperability by choosing a suitable standard for connecting distinct systems (e.g. OPC UA [5]) whereas semantic interoperability is ensured by specifying interface characteristics as unambiguously as possible.

In contrast, formal languages for capturing semantics are often based on ontologies (e.g. vocabulary) or logic-based languages (e.g. description logics) [8][9]. Although formal service descriptions may allow for automated inference mechanism (e.g. using a reasoner), all "relevant" information must be provided upfront in a complete manner [10].

As a consequence, practitioners often rely on several, informal standards and techniques that integrate them [6][7]. However, using standards often result in concrete interface implementation (e.g. REST) that do not have an unambiguous

[1] F. Burzlaff and C. Bartelt, "Knowledge-Driven Architecture Composition: Case-Based Formalization of Integration Knowledge to Enable Automated Component Coupling," 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), 2017, pp. 108-111, doi: 10.1109/ICSAW.2017.54.

meaning. Hence, an integrator is needed to test interoperability and implement adapters between system interfaces if necessary. Thus, dynamic adaptive systems preventing any integration effort at all must be well specified which may hinder its application in broad industry context. In addition, informal integration knowledge cannot be reused.

Ensuring service interoperability should ultimately shift from engineering language- and context-specific adapters towards reusing integration knowledge and thus maintaining knowledge-base.

The rest of this paper is structured as follows: Part II introduces a semi-formal specification based on general languages characteristics. Part III presents conceptual architectural principles based on interactive semantic interface integration techniques and Part IV summarizes this position paper.

II. RESEARCH CHALLENGE

Various researchers from different communities have classified integration tasks based on information, processes and functions [3][11][12].

Syntactic and semantic information integration is already a mature research topic and has been examined in database integration scenarios or by ontology matching techniques for knowledge-based integration [13]. Furthermore, services can be semantically integrated by using formal service specification languages or by using informal standards [8][5]. Nevertheless, dynamic adaptive systems purely defined by formal service specifications are not common in practice. In order to analyze the reasons for this circumstance more deeply, a clear separation between syntax and semantics of interfaces has to be made.

D. Harel and B. Rumpe [14] define syntax and semantics (i.e. the meaning of the syntax) of one language as follows:

- **Syntax**: Syntax is defined as the basic language expressions constrained by a grammar which are allowed to be use in a language in a particular way
- **Semantic Domain**: Syntactic expressions must correspond to one element in one semantic domain (i.e. universe of discourse) to reveal its meaning
- **Semantic Mapping**: A function maps every syntactic language expression to its semantic domain element

In addition, Karsai et al. [15] further state that syntax itself can be defined in various concrete syntax elements all referring to only one abstract, syntactic element (e.g. a graphical and a textual expression for the abstract, syntactic element “class”). Consequently, the semantic mapping takes place between the abstract syntax and the semantic domain. For the rest of the paper, definitions (I) - (VIII) will be used.

(I) SYN, DOM, IF

(II) $LANG = p(SYN) \times p(DOM) \times SEM$ where $p(X)$ indicates a power set of X

(III) $SEM = \{\overrightarrow{sem}: SYN \rightarrow DOM\}$

(IV) $L = (Syn, Dom, \overrightarrow{sem})$ where $Syn \subseteq SYN, Dom \subseteq DOM$ and $\overrightarrow{sem} \subseteq SEM$

(V) $\overrightarrow{Lang}: IF \rightarrow LANG$

(VI) $\overrightarrow{syntax}: IF \rightarrow SYN: \forall if \in IF: \overrightarrow{syntax}(if) \in Syn \Leftrightarrow \overrightarrow{lang}(if) = (Syn, Dom, \overrightarrow{sem})$

(VII) $\overrightarrow{semantics}: IF \rightarrow SEM$

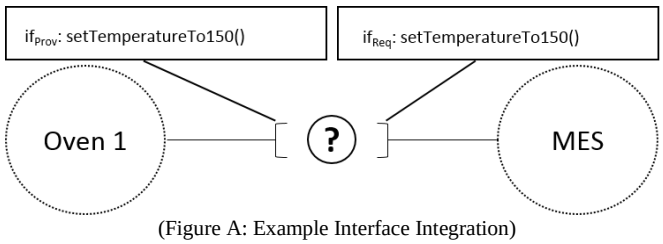
$\forall if \in IF, \overrightarrow{sem} \in SEM: \overrightarrow{semantics}(if) = sem \wedge \overrightarrow{lang}(if) = (Syn, Dom, \overrightarrow{sem})$

(VIII) $\overrightarrow{sem_{if}} = \overrightarrow{semantics}(if)$ (i.e. returns a functions that maps syntactic interface elements to their semantic element)

As an illustration, the semantic domain DOM could be “setting oven temperature to 150 °C” where the syntactic expression of one interface “setTemperatureTo150” is assigned it´s meaning by the semantic mapping function $\overrightarrow{sem}$ to the semantic domain. Following this classification, four interesting cases can be identified when a requestor should invoke a service on a provider. These are described as follows:

1. $\overrightarrow{syntax}(if_{REQ}) = \overrightarrow{syntax}(if_{PROV}) \wedge \overrightarrow{sem}_{if_{REQ}}\{\overrightarrow{syntax}(if_{REQ})\} = \overrightarrow{sem}_{if_{PROV}}\{\overrightarrow{syntax}(if_{PROV})\}$
2. $\overrightarrow{syntax}(if_{REQ}) = \overrightarrow{syntax}(if_{PROV}) \wedge \overrightarrow{sem}_{if_{REQ}}\{\overrightarrow{syntax}(if_{REQ})\} \neq \overrightarrow{sem}_{if_{PROV}}\{\overrightarrow{syntax}(if_{PROV})\}$
3. $\overrightarrow{syntax}(if_{REQ}) \neq \overrightarrow{syntax}(if_{PROV}) \wedge \overrightarrow{sem}_{if_{REQ}}\{\overrightarrow{syntax}(if_{REQ})\} = \overrightarrow{sem}_{if_{PROV}}\{\overrightarrow{syntax}(if_{PROV})\}$
4. $\overrightarrow{syntax}(if_{REQ}) \neq \overrightarrow{syntax}(if_{PROV}) \wedge \overrightarrow{sem}_{if_{REQ}}\{\overrightarrow{syntax}(if_{REQ})\} \neq \overrightarrow{sem}_{if_{PROV}}\{\overrightarrow{syntax}(if_{PROV})\}$

As an example, let´s assume that a Manufacturing Execution System (MES) in the role of a requestor requires an interface if_{Req} *setTemperatureTo150* on a heating oven. Furthermore, an oven provides an interface if_{Prov} *setTemperatureTo150*. This situation is depicted in Figure A.



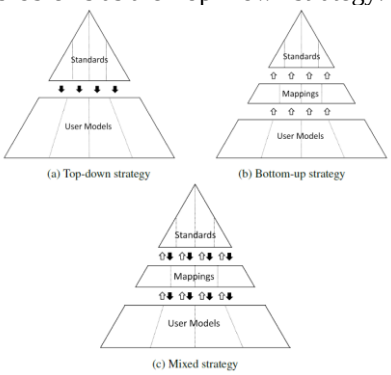
(Figure A: Example Interface Integration)

Case 1 and 2 are typically present if both interfaces are specified according to a prior agreed standard and language. The function $\overrightarrow{syn}(if_{Req})$ and $\overrightarrow{syn}(if_{Prov})$ would result in identical syntactical elements in SYN. As both interfaces are syntactically identical, it must be ensured that this syntactic element is mapped to the same semantic domain element

in DOM. Thus, Case 1 can be interpreted as using the same language in the same way for both interfaces. Please note that Case 1 only holds for both specific interface syntax, but may not hold for all possible language expression. Case 2 describes the circumstance where a single, identical syntactic element is mapped to different semantic domain elements. This ambiguous semantic mapping is created when prior defined guidelines are interpreted differently by interface designers.
Both cases are representative for interfaces based on prior defined formal or informal standards.
Henßen and Schleipen introduced a tool called AutomationML that is used during the design phase of software systems to syntactically and semantically describe various engineering concerns (e.g. physics and motion of an object) [6]. Furthermore, a companion standard ensuring interoperability with the widely used OPC UA Framework [6] as a vendor independent industry interface specification is provided.
However, the formal semantic mapping is not explicitly defined for both, OPC UA and AutomationML [5][6] that would unambiguously map syntactic elements to their semantic domain element. Thus, testing of sematic domain interoperability must be done to uncover interfaces representing Case 2.
At best, no integration effort (i.e. only testing effort) at all is necessary as every possible user model is created with the same syntactic and semantic domain element in mind. This is realized as a Top-down strategy (see Figure B) by creating detailed interface specifications and their syntactic mapping (e.g. by using a companion standard between AutomationML and OPC UA [6]).
Case 3 and 4 describe situations where system interfaces are developed independently without prior defining common standards. Consequently, interfaces may differ in their syntactic expression. For instance, assume that Oven 1 provides the interface *activateHeating()* as if_{PROV}. Thus, the function $\overrightarrow{syn}(if_{Prov})$ would return another element from SYN compared to $\overrightarrow{syn}(if_{Req})$. Nevertheless, both syntactic elements may still be mapped to the same semantic domain element (i.e. "setting oven temperature to 150 ℃") if *activateHeating()* is setting the temperature to exactly 150 ℃. This is described in Case 3. However, this is a special case where the system integrator explicitly knows about this circumstance before the concrete integration case. Case 4 describes the probable situation as the semantic function $\overrightarrow{sem}$ of a language cannot be explicitly evaluated or the system integrator simply does not know how to couple both interfaces. Thus, the integrator must find new relations between both components to be able to couple both interfaces. This "search" for new relations can by described as follows:

$$
\begin{aligned}
&(IX)\ \mathrm{REL} = \mathrm{LANG} \times \mathrm{LANG} \times \mathrm{p(SYN)} \times \{\colon \mathrm{SYN} \to \mathrm{SYN}\} \\
&\quad \forall\, \mathrm{r} \in \mathrm{REL}, \forall\, \mathrm{l}_1, \mathrm{l}_2 \in \mathrm{LANG}\colon \mathrm{r} \in \mathrm{Kb}^{\mathrm{l}_1,\mathrm{l}_2} \leftrightarrow \\
&\quad \mathrm{r} = (\mathrm{l}_1, \mathrm{l}_2, \mathrm{s}, \vec{\mathrm{t}}) \ \text{ where } \vec{\mathrm{t}} \text{ is a transformation} \\
&\quad \mathrm{r} \in \mathrm{Kb}^{\mathrm{l}_1,\mathrm{l}_2}\colon \mathrm{r} = \left(\mathrm{l}_1, \mathrm{l}_2, \mathrm{s}, \vec{\mathrm{t}}\right) \wedge \mathrm{l}_1 = (\mathrm{Syn}_1, \mathrm{Dom}, \overrightarrow{\mathrm{sem}}_1) \wedge \\
&\quad \mathrm{l}_2 = (\mathrm{Syn}_2, \mathrm{Dom}, \overrightarrow{\mathrm{sem}}_2) \Rightarrow \{\mathrm{S} \subseteq \mathrm{Syn}_1 \cup \mathrm{Syn}_2 \wedge \\
&\quad \forall\, \mathrm{s} \in \mathrm{S}\colon \overrightarrow{\mathrm{sem}}_1(\mathrm{s}) = \overrightarrow{\mathrm{sem}}_2(\vec{\mathrm{t}}(\mathrm{s}))\}
\end{aligned}
$$

This means, that this formalized search results in a relation that maps a syntactic element to another syntactic element both being mapped to the same semantic domain element.
Suppose the integration knowledge from Case 3 is not present. After several testing steps, this new relation can be formalized and inserted in a knowledge base Kb. In the future another system integrator not knowing about this connection can reuse it and the interfaces can be coupled.
Case 3 and 4 typically require system integrators to code individual adapters including integration knowledge. The required knowledge is often not reusable in an automated manner (Bottom-up strategy in Figure B). Parts of these mappings (i.e. relations between syntactic and semantic domain elements) can then be summarized as standards that may deal with similar problems as the Top-Down strategy.



(Figure B: Integration strategies)

Despite the circumstance that system designers can choose from a variety of suitable standards [1], they are often subject to interpretation issues and are only implemented for a specific use case. This circumstance is enforced if informal standards (e.g. documentation in natural language) or no standards at all are used. Furthermore, interfaces are often described in different languages.
To overcome vendor-specific interface interoperability problems, semantic service-oriented architecture can be used. Based on semantic interface specification languages, they offer mechanisms for searching, composing and executing services as an architecture principle [17][18]. Therefore, vendor-specific interfaces are equipped with formal descriptions in a suitable language (i.e. Input and Output parameters or Pre- and Postconditions). However, such approaches still require integrators to specify interfaces that are not suitable for reusing integration knowledge.
Thus, a mixed strategy approach is proposed that formalizes integration knowledge evolutionary (see Figure B). Each step is one specific integration task that is being formalized in a knowledge-base to facilitate knowledge reusability. Complete, formal described interfaces sare not necessary from the beginning on but can be eventually achieved over time. The next chapter will introduce initial architectural principles towards realizing the proposed strategy.

III. KNOWLEDGE-BASED ARCHITECTURE PRINCIPLES

Typically, a system integrator would connect distinct interfaces by writing suitable adapters (e.g. assume Case 3 applied to the situation depicted in Figure A). In order to make specific integration knowledge reusable, architecture principles are needed that incorporates interactive mechanisms to modify a knowledge-base as a supporting tool for system integrators. Furthermore, a possible engineering process is outlined (see Figure C):

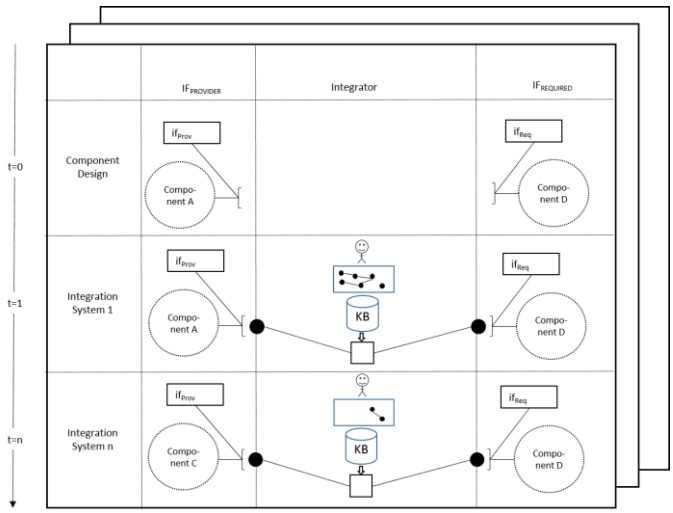


Figure C: Architecture Principles

In the component design phase at time $t = 0$, component provider as well as requester define their service interface. This can be done either based on prior defined standards (i.e. Top-Down strategy) or by vendor-specific implementation (i.e. Bottom-Up strategy). Assume at time $t = 1$, an integration task is necessary. Instead of writing individual adapter to connect component A and B, the system integrator adds syntactic mappings or relations between semantic domain elements including their language to a knowledge-base. This knowledge-base is then used to create an adapter and a connection can be established.

For example, the integrator would insert the new relation containing the syntactic mapping between if_{Req} *setTemperaturTo150()* and if_{Prov} *activateHeating()*. Thus, an adapter is constructed that actually transforms both syntactic interface expression in a correct way and enables Component A and Component B to communicate semantically without changing their initial interfaces.

Over time, various other components are also integrated (i.e. indicated by the other frames in Figure C) and new relations are added depending on the specific integration task. At time $t = n$, only few new relations are required which could ultimately result in fully automated component coupling.

The novelty of this approach is the evolutionary process of formalizing integration knowledge without requiring complete interface specification and how they should be integrated upfront. In addition, once an integration task has taken place the integration knowledge can be reused.

IV. CONCLUSION & FUTURE WORK

Both, the usage of complete interface specifications and context-specific interfaces expose weaknesses depending on their usage strategy. Strictly separating between syntax and semantics of an interface, interactive semantic interface integration mechanisms have been introduced that offer reusability of integration knowledge relying only on relevant interfaces.

In the future, a concrete example for working with a knowledge-base is implemented and evaluate which meta-constructs a suitable language for describing integration knowledge must offer. Furthermore, more complex functions including several parameters should be included in our formal definition and a prototype for evaluation should be created.

REFERENCES

[1] Meddeb, Aref. "Internet of things standards: who stands out from the crowd?." IEEE Communications Magazine 54, no. 7 (2016): 40-47.

[2] Lee, Edward A. "CPS foundations." In Design Automation Conference (DAC), 2010 47th ACM/IEEE, pp. 737-742. IEEE, 2010.

[3] Sheth, Amit P. "Changing focus on interoperability in information systems: from system, syntax, structure to semantics." In Interoperating geographic information systems, pp. 5-29. Springer US, 1999.

[4] Thoma, Matthias, Torsten Braun, Carsten Magerkurth, and Alexandru-Florian Antonescu. "Managing things and services with semantics: A survey." In Network Operations and Management Symposium (NOMS), 2014 IEEE, pp. 1-5. IEEE, 2014.

[5] Schleipen, Miriam, Syed-Shiraz Gilani, Tino Bischoff, and Julius Pfrommer. "OPC UA & Industrie 4.0-Enabling Technology with High Diversity and Variability." Procedia CIRP 57 (2016): 315-320.

[6] Henssen, Robert, and Miriam Schleipen. "Interoperability between OPC UA and AutomationML." Procedia CIRP 25 (2014): 297-304.

[7] Grangel-González, Irlán, Diego Collarana, Lavdim Halilaj, Steffen Lohmann, Christoph Lange, María-Esther Vidal, and Sören Auer. "Alligator: A Deductive Approach for the Integration of Industry 4.0 Standards." In Knowledge Engineering and Knowledge Management: 20th International Conference, EKAW 2016, Bologna, Italy, November 19-23, 2016, Proceedings 20, pp. 272-287. Springer International Publishing, 2016.

[8] Uschold, Michael, and Michael Gruninger. "Ontologies and semantics for seamless connectivity." ACM SIGMod Record 33, no. 4 (2004): 58-64.

[9] Grangel-González, Irlán, Lavdim Halilaj, Gökhan Coskun, Sören Auer, Diego Collarana, and Michael Hoffmeister. "Towards a semantic administrative shell for industry 4.0 components." In Semantic Computing (ICSC), 2016 IEEE Tenth International Conference on, pp. 230-237. IEEE, 2016.

[10] Grimm, Stephan, Boris Motik, and Chris Preist. "Matching semantic service descriptions with local closed-world reasoning." In European Semantic Web Conference, pp. 575-589. Springer Berlin Heidelberg, 2006.

[11] Rowley, Jennifer. "The wisdom hierarchy: representations of the DIKW hierarchy." Journal of information science 33, no. 2 (2007): 163-180.

[12] Weyrich, Michael, and Christof Ebert. "Reference architectures for the internet of things." IEEE Software 33, no. 1 (2016): 112-116.

[13] Wache, Holger, Thomas Voegele, Ubbo Visser, Heiner Stuckenschmidt, Gerhard Schuster, Holger Neumann, and Sebastian Hübner. "Ontology-based integration of information-a survey of existing approaches." In IJCAI-01 workshop: ontologies and information sharing, vol. 2001, pp. 108-117. 2001.

[14] Harel, David, and Bernhard Rumpe. "Meaningful modeling: what's the semantics of" semantics"?." Computer 37, no. 10 (2004): 64-72.

[15] Karsai, Gabor, Janos Sztipanovits, Akos Ledeczi, and Ted Bapty. "Model-integrated development of embedded software." Proceedings of the IEEE 91, no. 1 (2003): 145-164.

[17] Vitvar, Tomas, Adrian Mocan, Mick Kerrigan, Michal Zaremba, Maciej Zaremba, Matthew Moran, Emilia Cimpian, Thomas Haselwanter, and Dieter Fensel. "Semantically-enabled service oriented architecture: concepts,

technology and application." Service Oriented Computing and Applications 1, no. 2 (2007): 129-154.

[18] Haller, Armin, Emilia Cimpian, Adrian Mocan, Eyal Oren, and Christoph Bussler. "Wsmx-a semantic service-oriented architecture." In Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on, pp. 321-328. IEEE, 2005.