Next Steps in Knowledge-Driven Architecture Composition

Fabian Burzlaff[1], Christian Bartelt[2] and Heiner Stuckenschmidt[3]

1 University of Mannheim, 68131 Mannheim, Germany
burzlaff@es.uni-mannheim.de
[2] University of Mannheim, 68131 Mannheim, Germany
bartelt@es.uni-mannheim.de
[3] University of Mannheim, 68131 Mannheim, Germany
heiner@informatik.uni-mannheim.de

Abstract. Software architecture knowledge management has itself positioned as a mature research stream over the last years. Superficially, architectural knowledge management is about documenting design and design decisions. In software-intensive systems, a concrete application scenario of architectural knowledge management deals with the question whether a provided functionality fits a required functionality. To automate the underlying integration process, various research communities came up with, for example, interface definition languages and service matchers. However, formalizing the semantics of a software interface is in practice currently regarded as a price too high to pay. In this paper, we provide the status of our incremental case-based integration method that aims at reducing the effort for formalizing integration knowledge without losing the ability to compose software components based on interface semantics automatically.

Keywords: Knowledge-driven Architecture Composition, Interface Definition Language, Internet-of-Things, Integration Knowledge, Integration Methods

1 Introduction

Agile development methods and micro-services can be regarded as one of the state-of-practice artefacts when software solutions must scale dynamically. Two-Pizza-Teams are optimized towards implementing a minimal-viable-product while keeping the communication quality high. The agile manifesto “working software over comprehensive documentation” is present. Although this is a rather extreme example, software engineers tend to minimize their software documentation effort and ignore the needs for architectural knowledge management such as information discovery, sharing or traceability [1], [2].

A concrete example for implicit architectural knowledge can be seen when a system integrator examines whether a provided service fulfills the needs of a required functionality. Due to current trends such as the “Internet-of-Things” or “Industry 4.0”,

more and more physical objects are equipped with software interfaces to make them "smart". When connecting such devices with a communication platform (e.g. a communication bus or a client-server architecture), a system integrator must configure or even implement suitable translation adapters to establish a meaningful connection. This is needed to connect device services with other software services. The reason for creating such adapters are twofold: On the one hand, (domain) standards, if available and applicable, only describe the interface semantics in an informal way (e.g. OPC UA[1] or Swagger[2] and [2]). Hence, a human is needed to interpret the interface name and parameter and then program a software adapter between both endpoints. On the other hand, semantic interface description for web services (e.g. SAWSDL[3] over SOAP) could be used for formalizing the offered functionality based on ontologies. However, the creation of such descriptions is perceived as additional documentation effort and programmers do not know whether there will be a client who requires exactly this service.

As a consequence we have introduced an incremental and use-case specific integration method that tries to reuse prior formalized integration knowledge [3], [4]. In this context, formalizing integration knowledge means to make it machine-readable and does not mean to create a new formal standard incrementally. Hence, our research question (PhD) can be formulated as follows:

- **RQ**: How can software components be semantically coupled in an automated way based on partially incomplete integration knowledge?

2 Knowledge-Driven Architecture Composition

To answer our research question, we outlined a novel composition approach labeled "Knowledge-Driven Architecture Composition" (see Figure 1). One essential part of this approach is the usage of knowledge-base(s) for capturing integration knowledge between two interfaces (see "KB" in Figure 1). Within our method, a knowledge base contains information about the semantic relationship between two endpoints and their respective functionality. Overall, the process for capturing integration knowledge incrementally is as follows:

1. At time t=0, component D requires the provided functionality of component A. As the knowledge base is empty in the beginning, the system integrator must configure or implement an adapter (e.g. in an imperative programming language). In addition, he must capture the semantic transformation in a declarative language.
2. At time t=1, another functionality required by component D should be coupled with the provided functionality of component A. Again, the system integrator must perform both actions, formalizing the additional integration knowledge needed for

[1] https://jp.opcfoundation.org/wp-content/uploads/2014/05/OPC-UA_CollaborationOverview_DE.pdf
[2] https://swagger.io/
[3] https://www.w3.org/TR/sawsdl/

this case and configure/implement the adapter. However, existing declarative integration knowledge can now be reused. This could be either performed in the sense of a recommender systems that provides a suitable solution based on previous integration cases or by generating an adapter template that requires the system integrator only to insert the missing integration knowledge.

3. At time t=n, component A should be replaced by another component C (e.g. due to efficiency reasons). As both component offer the same semantic interface functionality, now previously formalized integration knowledge can be reused. Furthermore, if enough integration knowledge from other integration cases is present, a reasoner can derive missing integration knowledge automatically. Thus, previously unknown components can be integrated in an automated way and an executable adapter could be generated.

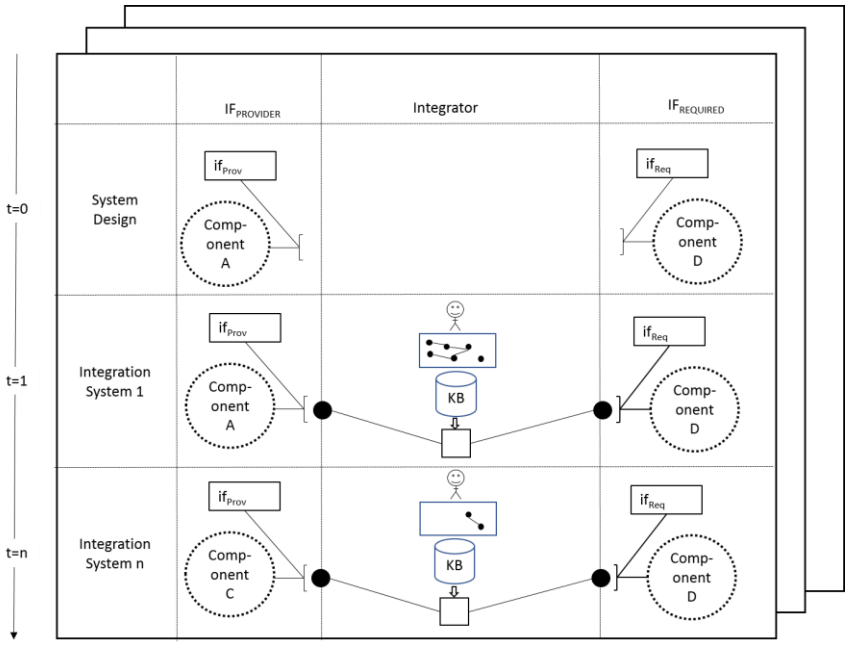

Fig. 1. Knowledge-Driven Architecture Composition Method

For supporting the system integrator to formalize the integration knowledge when executing this method over time, we are currently using the following technologies: 1) OWL-DL for storing integration knowledge 2) An Eclipse-based Editor for defining the entities, the object as well as data properties and the use-case specific individuals and 3) the reasoner Hermit (Version 1.3) as an inference mechanism.

2.1 Capturing Integration Knowledge with OWL-DL Ontologies

At the moment, our knowledge-base consists of three connected ontologies. One ontology for describing the data points as well as methods for each endpoint and one

abstract integration ontology that imports endpoint ontologies. The integration ontology with selected individuals is illustrated in Figure 2.

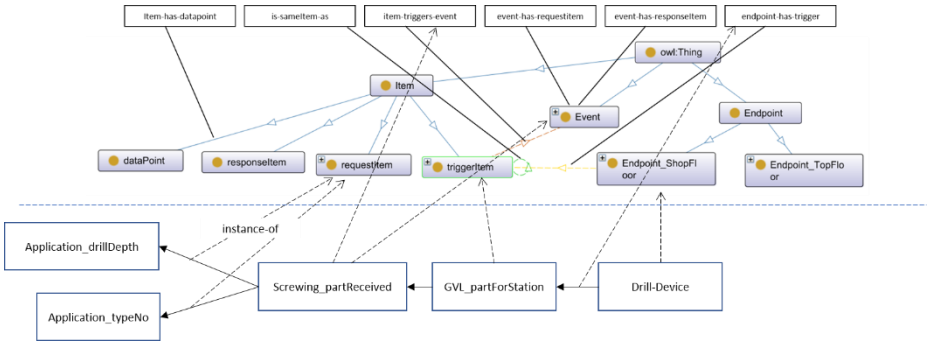

Fig. 2. Simple Integration Example based on trigger-request communication pattern

As this example originated out of the Industrial Automation context, a few details about the use-case may be useful: A communication between a drill device (i.e. Endpoint_ShopFloor) and a Manufacturing Execution System (i.e. Endpoint_TopFloor) should be established. Based on a simple trigger-request communication style, a message from the device is being sent to the MES-System. This message is sent as soon as the workpiece is detected by a physical sensor (e.g. *Boolean GVL_partForStation == true*). To map the information residing in the drill information model, the variables *Application_drillDepth* is read and sent via the MES-specific XML template *Screwing_partRecevied* to the TopFloor system. Due to the circumstance that in this use-case only one MES system as available, the correct individual for the entity *Endpoint_ShopFloor* can be directly selected. Concerning the automatic deduction process of new integration knowledge, a simple example could be the transformation of units of *Application_drillDepth* (e.g. transforming centimeter and inches).

As a consequence, if the drill device is now being replaced by another drill with extended capabilities and no changes in the communication style, the drill can be theoretically integrated in a plug-and-play manner based on formalized integration knowledge. Therefore, the system integrator only has to provide a mapping between concrete device interface and the endpoint entity by creating a new *Endpoint_ShopFloor* individual.

3 Related Work

Automatic interface coupling approaches are not new. Such approaches have been around within the component-based software engineering community for quite a long time [5]. In fact, the underlying ideas of information hiding, and modularization have already been tackled by early programming languages. For example, Modula 2 could already automatically check for interface compatibility in the year of 1978 [6].

The novelty of our approach lies in the idea of focusing on a method for formalizing integration knowledge that keeps the formalization effort low. This does not mean, that we ignore standards or build up standards incrementally but to provide a method and tool for capturing integration knowledge explicitly in a systematic way. Still, a uniform ontological schema per communication pattern must be used across all integration cases.

Regarding supporting research streams, case-based reasoning methods as well as interface matching approaches are relevant.

3.1 Case-based Reasoning

Case-based reasoning methods are based on a problem-based learning method that derives solutions for unknown problems from existing and known problem-solution pairs [7]. Their advantages lie in the sophisticated deduction process based in previous cases and have been successfully applied in the healthcare domain. Hence, reusability of existing solutions is achieved. However, case-based reasoning methods mostly rely on a continuous similarity function meaning that they may adapt a solution in an incorrect way. This can become a problem regarding interface coupling scenarios as the proposed solution may be partially incorrect.

3.2 Interface Matching Approaches

Interface Matching approaches exist for both, web services as well as software component interfaces. For matching SAWSDL descriptions, Klusch et al. [8] developed the SAWSDL-MX matcher that evaluates if two interfaces can be coupled based on their semantic interface description. For software component interfaces, hybrid matching approaches were developed for "on-the-fly" service matching [9]. These matching approaches are interesting for our approach, as they might provide reasonable results when not enough integration knowledge is present. In other words, they can be used as a recommendation system. However, a drawback of these matchers is that they either are too restrictive regarding their description capabilities [8] or that they only produce a probability score [9] which means that the coupling mechanism may produce wrong results.

4 Open Questions and Next Steps

In the last year, we have worked on a technical prototype for evaluation purposes of our integration method. During a first evaluation in the context of Industrial Automation [10][11], we identified, among others, further questions regarding the underlying knowledge acquisition, knowledge storing as well as the reasoning process.

- How can a system automatically detect if two interfaces are semantically identical if they are syntactically different?
- How can we "tell" the system integrator which integration knowledge is missing during an unknown integration case (e.g. *whynot* queries)?

- How can knowledge bases be maintained if integration knowledge changes over time (i.e. validation of integration knowledge across context-independent use-cases)?
- What are other suitable declarative languages and reasoner for capturing component composition knowledge besides OWL-DL and HermiT (i.e. functional composition in a mathematical sense like **λ**-Prolog)?

In the future, we will focus on how to support other communication patterns. Furthermore, we will investigate how new integration knowledge can be practically deduced and which assumptions must hold for such deduction approaches to work in practice.

References

[1] R. Capilla, A. Jansen, A. Tang, P. Avgeriou, and M. A. Babar, "10 years of software architecture knowledge management: Practice and future," *J. Syst. Softw.*, vol. 116, pp. 191–205, Jun. 2016.

[2] "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems," *IEEE Std 1471-2000*, pp. i–23, 2000.

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

[4] F. Burzlaff, *Knowledge-driven Architecture Composition*. Gesellschaft für Informatik, Bonn, 2017.

[5] T. Vale, I. Crnkovic, E. S. de Almeida, P. A. da M. Silveira Neto, Y. C. Cavalcanti, and S. R. de L. Meira, "Twenty-eight years of component-based software engineering," *J. Syst. Softw.*, vol. 111, pp. 128–148, Jan. 2016.

[6] N. Wirth, "Modula-2 and Oberon," in *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages*, New York, NY, USA, 2007, pp. 3–1–3–10.

[7] A. Aamodt and E. Plaza, "Case-based Reasoning: Foundational Issues, Methodological Variations, and System Approaches," *AI Commun*, vol. 7, no. 1, pp. 39–59, Mar. 1994.

[8] M. Klusch, P. Kapahnke, and I. Zinnikus, "SAWSDL-MX2: A Machine-Learning Approach for Integrating Semantic Web Service Matchmaking Variants," in *2009 IEEE International Conference on Web Services*, 2009, pp. 335–342.

[9] M. C. Platenius, W. Schäfer, and S. Arifulina, "MatchBox: A Framework for Dynamic Configuration of Service Matching Processes," in *Proceedings of the 18th International ACM SIGSOFT Symposium on Component-Based Software Engineering*, New York, NY, USA, 2015, pp. 75–84.

[10] F. Burzlaff, C. Bartelt, and L. Adler, "Towards automating Service Matching for Manufacturing Systems: Exemplifying Knowledge-Driven Architecture Composition," *Procedia CIRP*, vol. 72, pp. 707–713, Jan. 2018.

[11] F. Burzlaff & C. Bartelt - I4.0-Device Integration: A Qualitative Analysis of Methods and Technologies Utilized by System Integrators (ICSAW 2018 - to be published)