

Gabble: Managing Integration Knowledge in IoT-Systems with Logical Reasoning

Fabian Burzlaff¹, Maurice Ackel² and Christian Bartelt²

¹*osapiens Services GmbH, Julius-Hatry-Straße 1, Mannheim, 68163, Germany*

²*Institute for Enterprise Systems (University of Mannheim), Schloss, Mannheim, 68131, Germany*

Abstract

Service interoperability for embedded devices is a mandatory feature for dynamically changing Internet-of-Things and Industry 4.0 software platforms. Service interoperability is achieved on a technical, syntactic, and semantic level. If service interoperability is achieved on all levels, plug-and-play functionality known from USB storage sticks or printer drivers becomes feasible. This reduces the manual effort for system integration for home automation systems and, in the case of the producing industry, allows for micro-batch size production, individualized automation solution, or job order production. However, interoperability at the semantic level is still a problem for the maturing class of IoT systems. In this work, we present a software engineering tool that allows storing, sharing, and reusing integration knowledge between software interfaces incrementally by looking at integration cases instead of domain models.

Keywords

Knowledge-driven Architecture Composition, Software Architecture, Component Coupling, Artificial Intelligence, Knowledge Management, Internet of Things

1. Introduction

Architectural mismatch due to semantic differences in software interfaces is a well-known problem [1, 2]. For example, current Internet-of-Things platforms require system integrators to implement point-to-point adapters, enforce a domain standard or rely on more abstract interface description languages when coupling embedded devices. However, both standards and machine-understandable interface descriptions cannot be applied effectively to IoT systems as they rely on the assumption that the semantic domain is completely known when they are created. This assumption of complete integration models hardly holds in the real world as ever-changing environments render a complete and final description of a domain impossible. Consequently, practitioners rely on implementing software adapters manually without technical support to store, share and reuse integration knowledge between interfaces.

This work introduces a novel tool called Gabble that explicitly allows for an incomplete semantic domain model by looking at integration cases instead of domain models. "Gabble" is inspired by the fast-growing number of connected devices which talk so quickly that devices cannot understand each other. Therefore, it assists the system integrator at design time with

ECSA 2021: 15th European Conference on Software Architecture

✉ fabian.burzlaff@osapiens.com (F. Burzlaff); m.ackel@icloud.com (M. Ackel); bartelt@es.uni-mannheim.de (C. Bartelt)

ORCID  0000-0003-0632-5933 (F. Burzlaff); 0000-0003-0426-6714 (C. Bartelt)

 © 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

logical reasoning capabilities by 1) proposing interface mappings based on previous integration cases and by 2) generating a software adapter in an automated way. We assume, that valid interface descriptions are present but the adapter design can be incomplete (i.e., missing functionalities).

2. Use Case

To illustrate the tool functionality, we will take a look at an exemplary use case. In our setting, Alice and Bob work on an app that controls a Philips smart light. This device has a public API which is described in a syntactic specification standard (e.g., OpenAPI). Based on this specification, the client code to interface with the device was created using existing code generators. Subsequently, the necessary client logic to work with the generated library was developed.

The development team is now tasked to make the app work with a different device (i.e. by Yeelight). This light has a different, yet semantically identical API with a corresponding API specification. The interfaces of both devices are depicted in Fig. 1. This figure also shows how a different person integrated the Philips API with the LIFX (smart light brand) API (at time t=1), and yet another system integrator mapped the LIFX to the Yeelight API (at t=2).

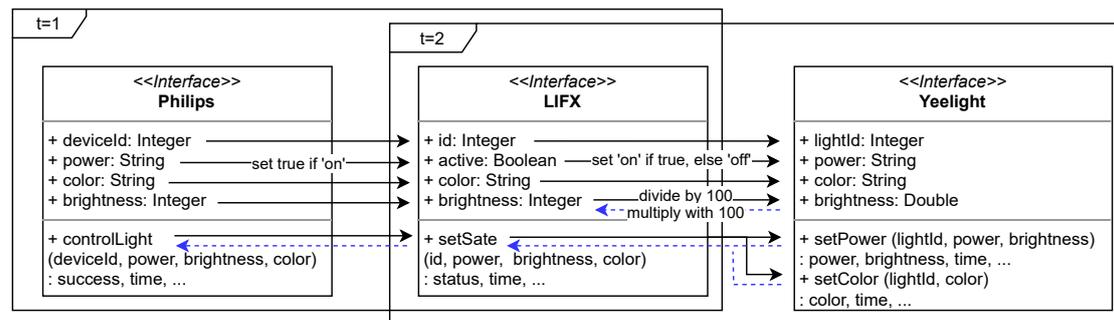


Figure 1: Integration Knowledge Example

To make the Yeelight device work with the existing app in a practical way, Bob would have to generate new client code and adjust the application logic so that it can handle the new data model. This process has to be done manually every time such a change needs to be performed and does not allow the reuse of existing integration knowledge (see arrows that illustrate mappings in Fig. 1). The Gabble tool allows for such easy reuse of integration knowledge which is defined in a case-based manner. The main benefit of the tool support is the ability to handle simple scenarios as displayed and more complex ones with thousands of integration cases where manual knowledge extraction would get infeasible.

3. Functionality & Architecture

The underlying theoretical approach of Gabble is Knowledge-Driven Architecture Composition (KDAC) – a novel paradigm of system integration that allows reusing atomic integration

knowledge preserved from previous integration cases [3]. To do this, the approach captures integration knowledge in a graph-like knowledge base, where APIs are vertices and mappings between them are edges. The approach explicitly does not assume that integration knowledge is complete once it is defined. Instead, it accepts that integration knowledge is always incomplete and aims to reuse as many mappings as possible. Gabble implements this approach, offering system integrators, and developers a new way to create integrations to make the formalization process easier, faster, and less error-prone.

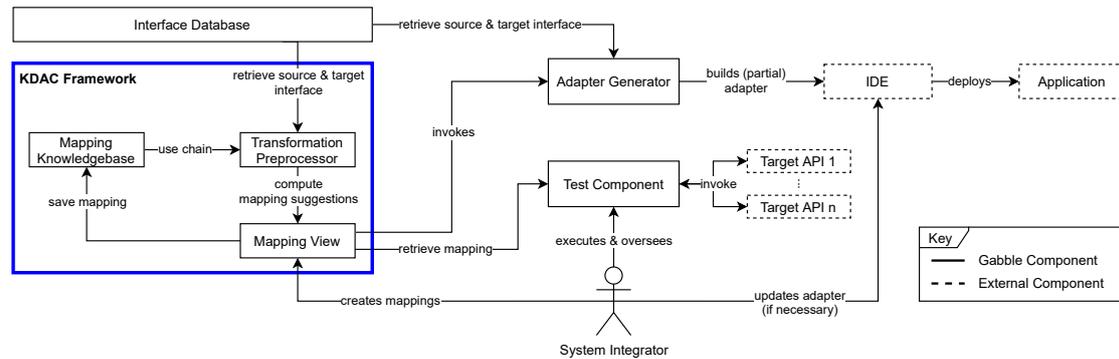


Figure 2: Logical System Architecture

The tool consists of several logical components (see Fig. 2). To start with, it allows users to add interfaces to the *Interface Database* which are then available later in the process. When adding an interface, the user has to provide either an AsyncAPI or OpenAPI specification and the interface name. Those interfaces are then available in the *Mapping View* where the user can create mappings between them. To do this, they select a source interface and one or many target interfaces. As soon as a selection is made, the *Transformation Preprocessor* is invoked. It uses the integration knowledge stored in the *Mapping Knowledgebase* to compute mapping suggestions. Those are directly added to the mapping view and color coded, based on their generation type.

The generation of suggestions is done in two ways. For transitive mapping suggestions, the preprocessor searches for paths from the source to the target interface(s). The resulting mapping chain is combined into a single mapping operation from source to target if a path is found. In our example, a mapping chain would be Philips → LIFX → Yeelight. The other type of suggestion is based on knowledge graphs on single attributes instead of whole mappings. Those knowledge graphs capture the equality relationships (e.g., arrows in Fig. 1) between single interface attributes. Invoking the *Transformation Preprocessor* on every update allows generating new mapping suggestions whenever the user adds new integration knowledge in the integration case at hand.

It is important to note that the generated mapping suggestions might not be able to cover all the required attributes of the target interface(s), in which case the user can also add manual mappings. Those mappings can either be one-to-one (i.e., one provided attribute to one required attribute) or many-to-one (i.e., several provided attributes to one required attribute). Mappings are created by selecting the involved attributes in the *Mapping View* and can either be *simple* (i.e., only value replacement, no computations) or *complex* (i.e., mathematical functions to

convert or combine attribute values). In our example, this is displayed by arrows without a label or with a label.

Once all required attributes are assigned, the user can test the mapping using the *Test Component*. To do this, the user provides the input data in the source interface's data model and executing the mapping. This will transform the source to the target data model and perform the requests against the target API(s). The final result and all intermediate transformations are then shown to the user to ensure the correct functionality.

Finally, the user saves the complete mapping in the Mapping Knowledgebase, making it available for other users. In this way, case-based integration knowledge is preserved and made available for reuse. Once a mapping is completed, a software adapter can be automatically generated. This adapter encapsulates the mapping of the data models defined by the user. The result is a code library that has the same API as the client library for the source interface if it had been directly generated by an OpenAPI code generator. In our case, we added support for JavaScript adapters, but the *Adapter Generator* can also be extended to support other programming languages.

The Gabble tool's underlying services are containerized using Docker and can be deployed using one single command on any cloud infrastructure. Extensions are feasible as we rely on current software frameworks such as React, TypeScript, OpenAPI, JSONata, and Mustache templates. The benefits of the tool and the underlying method have already been demonstrated in a home automation scenario by decreasing the engineering time of software adapters and increasing the reliability of interface mappings [4, 5].

4. Related Research and Industry Efforts

Research-driven approaches to achieve semantic interoperability are focusing on bottom-up interface integration and can be exemplified using the tools MatchBox [6], and MICS [7]. MatchBox presents a highly customizable interface matching framework based on interface descriptions, whereas MICS enables software architectures to synthesize software connectors from formal mapping specifications automatically. The mentioned approaches and most other approaches apply formal ontologies to describe the desired domain based on the available interface descriptions. Once defined, these ontologies are hard to evolve for system integrators.

Approaches from the industry include top-down designed standards such as OPC UA in combination with ecl@ss. For instance, BaSys 4.0 [8] takes a capability-based description that contains references to an integration layer for manufacturing systems. These approaches assume that there exists a formal vocabulary that can be used by all parties involved by linking their interfaces to it.

The Gabble tool and the underlying approach do not try to (partially) formalize a domain and link interface elements to it, but look at integration knowledge based on concrete integration cases that can evolve over time.

5. Conclusion

In this work, we presented Gabble. Gabble is a tool accompanying the knowledge-driven architecture composition approach, which resolves architectural mismatch at the semantic level for highly interconnected and dynamic IoT software platforms. We believe that Gabble can lower the amount of manually written adapter code until one domain-specific standard emerges. In the future, we plan to apply Gabble to other integration scenarios demanding a higher degree of dependability, such as automated production lines.

Acknowledgments

This work has been developed in the project BIoTope (Research Grant Number 01IS18079C) and is funded by the German Ministry of Education and Research (BMBF).

References

- [1] D. Garlan, R. Allen, J. Ockerbloom, Architectural mismatch: Why reuse is still so hard, *IEEE software* 26 (2009) 66–69.
- [2] F. Burzlaff, N. Wilken, C. Bartelt, H. Stuckenschmidt, Semantic Interoperability Methods for Smart Service Systems: A Survey, *IEEE Transactions on Engineering Management* (2019) 1–15.
- [3] F. Burzlaff, 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)*, IEEE, 2017, pp. 108–111.
- [4] F. Burzlaff, C. Bartelt, Knowledge-driven architecture composition: Assisting the system integrator to reuse integration knowledge, in: M. Brambilla, R. Chbeir, F. Frasincar, I. Manolescu (Eds.), *Web Engineering*, Springer International Publishing, Cham, 2021, pp. 305–319.
- [5] F. Burzlaff, Knowledge-driven architecture composition, Ph.D. thesis, Mannheim, 2021.
- [6] M. C. Platenius, Fuzzy matching of comprehensive service specifications, PhD Thesis, Universitätsbibliothek, Paderborn, 2016.
- [7] M. Autili, P. Inverardi, R. Spalazzese, M. Tivoli, F. Mignosi, Automated synthesis of application-layer connectors from automata-based specifications, *Journal of Computer and System Sciences* 104 (2019) 17–40.
- [8] Capability-based semantic interoperability of manufacturing resources: A basys 4.0 perspective, *IFAC-PapersOnLine* 52 (2019) 1590–1596. 9th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2019.

Online Resources

- Tool Demonstration Video: <https://www.youtube.com/watch?v=6IuChI6Q4E4>
- Source Code: <https://github.com/mauriceackel/Gabble/tree/demo>
- Web Page: <https://iot.informatik.uni-mannheim.de/>