Towards service orchestration through software capability profile

Abdelhadi Belfadel, Jannik Laval, Chantal Cherifi, Néjib Moalla

▶ **To cite this version:**

Abdelhadi Belfadel, Jannik Laval, Chantal Cherifi, Néjib Moalla. Towards service orchestration through software capability profile. 9th International Conference on Interoperability for Enterprise Systems And Applications (I-ESA 2018), Mar 2018, Berlin, Germany. hal-01689071

HAL Id: hal-01689071
https://hal.science/hal-01689071v1

Submitted on 20 Jan 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Towards service orchestration through software capability profile

Abdelhadi Belfadel, Jannik Laval, Chantal Bonner Cherifi, and Nejib Moalla

University Lyon 2, DISP Laboratory
{abdelhadi.belfadel,jannik.laval,
chantal.BonnerCherifi,nejib.moalla}@univ-lyon2.fr

Abstract. Open source solutions offer great reuse opportunities. However, the difficulty lies on the appropriation of these solutions to meet specific business requirements. We aim in this work to decompose an open source application in specific functionalities as a conceptual view for service orchestration. We provide a solution to characterize a functionality of an open source application in a readable and standardized way. Then we provide an automated solution for the externalization of these characterized functionalities as an Application Programing Interface. Our approach uses capability profile provided by ISO 16100 series which is a standardized methodology for interoperability of manufacturing software. As a result, we generate reusable components for service orchestration needs.

Keywords: open source application; software reuse; service orchestration; API; ISO 16100; capability profile; enterprise application integration;

1 Introduction

Small and Medium-Sized Enterprises (SMEs) are the most common firms in many countries. In Europe (EU28), 23 million SMEs employ more than 90 million people. They represented, in 2015, 99.8% of all enterprises [15], and they are increasingly doing open innovation to bring ideas in the market to improve productivity, increase competitiveness and facilitate entrance to new markets [9]. In order to do so, SMEs need to explore new solutions and integrate new functionalities quickly. This results in prototyping new business needs in short time period without cost or engagement with a software vendor. But in most cases, SMEs operate under limited resources which restrict their innovativeness [9]. High number of open source solutions results from the Factories of the Future (FoF) initiatives. This aims to develop the necessary key enabling technologies and help manufacturing enterprises to adapt to global competitive pressures.

Currently, the development of software applications is based on the reuse of existing functionalities instead of developing them from scratch [11, 19]. Application Programming Interfaces (APIs) are considered as the most commonly used

entities supporting software reuse [18, 11]. APIs provide an implemented, tested and high quality functionalities, and they increase software quality and reduce the effort spent on coding, testing and maintenance activities [19].

In this context, we provide a framework to facilitate the appropriation of the open source applications and bring adequate solutions to SMEs. This framework consists of four steps as source code analysis, evaluation, servitization and orchestration model. In the source code analysis step, we analyze the source code to detect existing services and potential candidates (functionalities) to externalize. In the evaluation step, we qualify existing services of an open source application and elect reliable and trustworthy candidates. In the servitization step, we characterize and servitize the elected candidates in an automated process. Finally in the orchestration step, we propose an orchestration model for the externalized functionality to save the business logic offered by the open source application. For this purpose, we present in this paper one part of the proposed framework with a proof of concept applied on an open source application resulted from a FoF initiative (FITMAN[1]). Starting from the characterization of the functionalities done in a readable and standardized way. Then we provide an automated solution for the externalization of the characterized functionalities as REST (Representational state transfer) APIs for service orchestration. Our approach uses capability profile provided by ISO 16100 series which is a standardized methodology for interoperability of manufacturing software. As a result, we generate reusable components for service orchestration needs. This paper is structured as follows. Section 2 focuses on the related work and Section 3 presents an overview of some useful standards for our solving approach. Section 4 is dedicated to the proposed solution and Section 5 presents an implementation of the proposed solution applied on an open source application. Finally, conclusion and future works are drawn in section 6.

2 Related work

In this section, we discuss works related to source code analysis, software reuse and legacy to SOA migration.

2.1 Source code analysis & software reuse

Metrics are powerful support tools in software development and maintenance. They are used to assess software quality, to estimate complexity, cost and effort, to control and improve processes [16]. The metrics that are important to calculate reusability are related to inheritance, cohesion and coupling. In [16], the authors measure the association between numbers of classes, check the direct dependencies, indirect dependencies, IO dependencies, number of out and in metrics in object oriented programming. In [12], the authors propose a method to display dependencies between modules in reuse-based embedded software development, and adding development management property data to each module

[1] www.fitman-fi.eu

in order to support developers to know which modules will be affected when some parts of the reused software are modified. Other authors in [13] identify components from object oriented source code based on quality-centric metrics.

2.2 Legacy to SOA evolution

In the literature, the concept of SOA is interpreted in many different ways. Different approaches to SOA migration are proposed. A brief overview of legacy to SOA evolution is reported by [7] that divides the legacy to SOA evolution approaches into four categories: replacement, redevelopment, wrapping and migration. In [17], the authors report a systematic literature review of SOA migration approaches. They propose a reference model, called SOA migration frame of reference, that can be used for selecting and defining SOA migration approaches. In [6], the authors outline a semi-automated approach to migrate dynamic legacy web applications to web services-based SOA applications by using two technologies, Service Component Architecture (SCA) and Service Data Object (SDO). They used a manual approach to identify the potential service within each function. Other authors propose a framework and guidelines for the identification of specific services from legacy code [5]. Their approach focuses on defining the services based on a Model-Driven Architecture approach.

3 Standards

In order to reach service orchestration using functionalities of open source applications, we need to know more about the entities that have to inter operate. Two standards intend to provide this knowledge by offering a way to create profiles of the selected entities, ISO 15745 (Industrial automation systems and integration - Open systems application integration frameworks) [1] and ISO 16100 (Industrial automation systems and integration - Manufacturing software capability profiling for interoperability) [4]. The ISO 15745 defines an Application Integration Framework (AIP) which is a set of elements and rules for describing integration models and application interoperability profiles as well as their component profiles, process profiles, information exchange profiles, and resource profiles. The standard defines also the technology specific elements and rules for describing both communication network profiles and the communication related aspects of device profiles based upon particular fieldbus technologies. The middle section of figure 1 shows the AIP, consisting of one process profile, one or more resource profiles, and one or more information exchange profiles. Underlying the AIP are the relevant integration models which represent the application requirements [14]. With its focus on shop floor application, figure 1 shows the next level of details for the resource profile consisting of communication, device, human, material and equipment profiles. On the other hand, the standard ISO 16100 targets the representation of a software capability profile [4]. It specifies a framework for assessing the interoperability of a set of software products used in the manufacturing domain, and provides a method which is independent of a particular

system architecture or implementation platform for constructing profiles of manufacturing software capabilities [8]. Figure 2 shows the concepts defined in the different parts of ISO 16100.

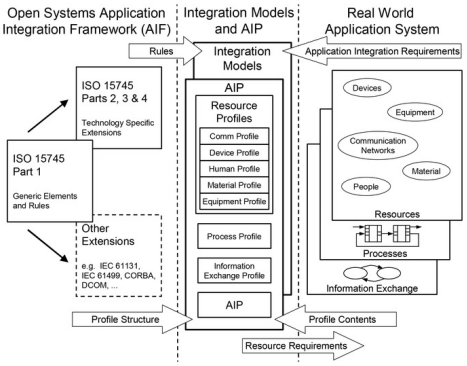

Fig. 1. Context of ISO 15745 [1]

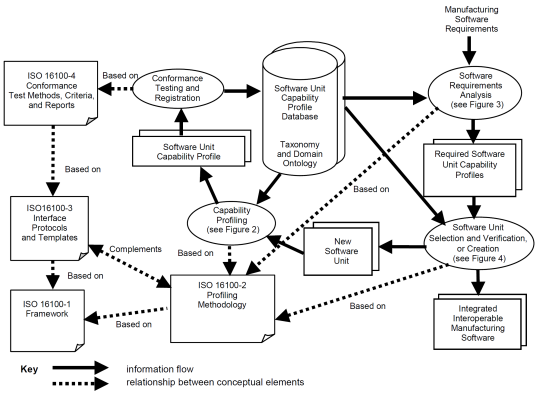

Fig. 2. Context of ISO 16100 [3]

3.1 API Documentation

An API is the published interface and a service is the concrete implementation of an API running in the back-end. It is typically a black box, which means that source code is not publicly accessible. API documentation is very important for the successful adoption of an API. APIs expose data and services and should be designed with an interface that the consumer can understand. The documentation should help developers to learn the functionalities offered by an API and enable them to start using it quickly. The API document should provide all necessary information to developers or API consumers in a human-readable format and help them assess its suitability for use in their client app. It should provide information about its licensing policy and usage requirements-input and output parameters, message format, error messages, and more. Similarly, the API interface should be documented such that its interface can be parsed by a machine to generate client stubs and server-side skeleton code that can be further developed. To make API documentation effective, it should include the following aspects about the API [10]: title, endpoint, method, URL parameters, message payload, header parameters, response code, error code. Many tools and technologies are available for API documentation. We can find RESTful API Modeling Language (RAML) [2], API Blueprint [3] and Swagger [4] (was renamed the OpenAPI Specification when it was donated to the Open API Initiative).

[2] http://raml.org/
[3] https://apiblueprint.org/
[4] https://www.openapis.org/

4 Contribution

As outlined in the previous sections, our goal is to help companies select the most suitable open source application, and its reusable components to meet their business requirements which is composed by user and service type tasks. The final choice of reusing an open source application depends on the precision of the expression of the needs. The more the needs are precise, the more the selection of the open source application is easy. As a result, the company will choose the application which answers most in term of reuse to service tasks of the new business application. Figure 3 shows the proposed framework, composed by 4 main parts. The first step is the code extraction and analysis. In this step,

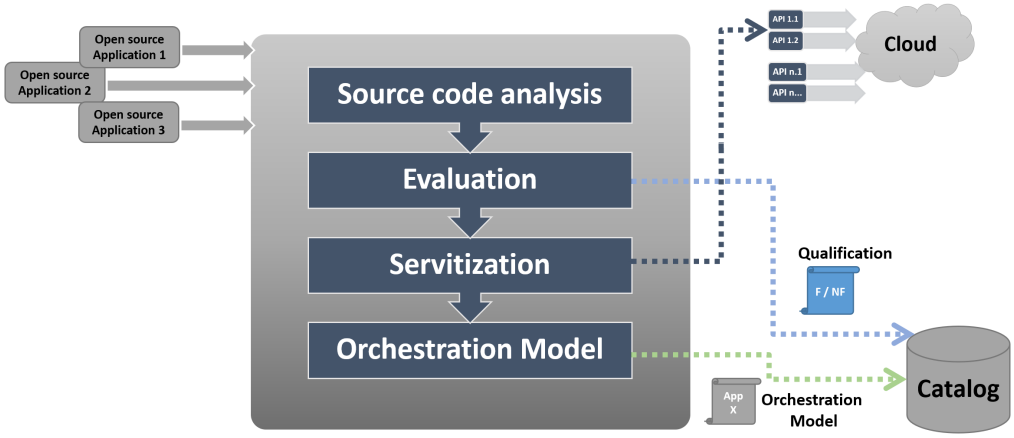

Fig. 3. Proposed Framework

we transform the extracted source code into a model, allowing us, with the help of visualization techniques, to detect the existing services and identify potential candidates (methods) to expose as a service. The second step is the Evaluation part. This step focuses on the technical aspect by applying some metrics helping to elect reliable candidate to externalize (next step process), and qualify existing services of the open source application to explore their capabilities, system properties, conditions of use and limits. All these informations are gathered in a single catalog for discovery, maintenance and reuse purpose. In the third step, we characterize the candidate functionality in a readable and standardized way, and an automatic process is applied to generate the API to be deployed on the Cloud. The last part represents the orchestration model generation. In this step, we generate an orchestration template when the candidate method belonging to the core of the application and depend on another service. An orchestration template is generated to not modify the business logic offered by the application. The originality of this work is to transform a servitized, semi-servitized or legacy

application into SOA application. For (semi-)servitized applications, it allows to qualify, give visibility and secure existing services with APIs in order to ease the reuse. It provides also a way for semi-servitized applications to characterize the functionalities to expose, and automate the steps to reach the service. For non-servitized applications, the goal is to come out with a full service application that facilitates its reuse. In this paper, we focus only on the third step of the proposed framework which is composed by the characterization and servitization steps (detailed in Figure 4). In order to reach our objective of the

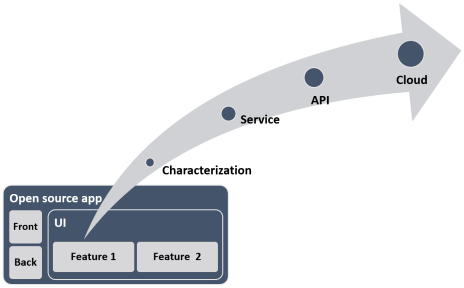

Fig. 4. Servitization steps

service orchestration, we have selected the framework offered by ISO 16100 for the characterization part because it focuses on the interfacing requirements for interoperability, instead of ISO 15745 which identifies a larger set of elements needed to support interoperability between application components.
Our contribution is the proposal of an ISO 16100 capability template with its capability class described with XML Schema. We propose also an automated way to reach a reusable component based on OpenAPI Specification, which is one of the most popular API documentation framework. OpenAPI Specification provides standard, language-agnostic way of defining a REST API Interface, and allows the consumer to understand the capabilities of the REST API without any prior access to the service implementation code or network inspection [10]. We expose in the following (section 4.1), the capability profile process as described in ISO 16100 standard. Then we propose in section 4.2, the steps to reach the API documentation from the capability profile allowing to get the desired API.

4.1 Capability profile process

First, we have to identify candidate methods to externalize and expose as an API. In this paper, we do not present this step. We will deal with it during our future work. In the following, we take the output of the evaluation step, which are candidate methods, and we apply the next step of the framework which is the characterization step allowing to get the ISO 16100 capability profile of the method. Each application is represented as an activity tree structure that is

hierarchical. And an activity is considered as a software unit in the ISO 16100 specification. The interoperability of software units can be described in terms of their capabilities that are associated with the aspects of functionality, interface and structure. The profiling of a software unit involves the generation of a concise statement of capabilities enabled by the software unit in terms of the functions performed, the interfaces provided, and the protocols supported. The part of the concept of capability profile for software interoperability shown in figure 2 related to the capability profiling process is detailed in figure 5. A software unit to be profiled shall be analyzed and a template shall be filled to make a profile. In this work, we have formed a new capability class and a new capability template in XML Schema, helping to retrieve information needed to externalize a feature of an application. Figure 6 shows an example of informations contained in the specific part of a capability profile.

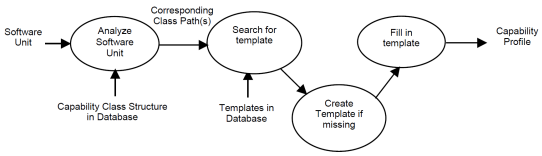

Fig. 5. Capability profiling process [2]

```
<Specific>
    <ReferenceCapabilityClassStructure
        id="cea_1001" name="CAM_FunctionSample" version="001" url="">
        <Activities>
            <Worker>
                <Function ID="CreateAsset" action="create" level="4" />
            </Worker>
        </Activities>
        <InformationExchange>
            <InputDataTypes level="1" ID="inputDataTypesId">
                <Numerical level="2" ID="numericalId">
                    <One-Dimensional level="3" ID="objId">
                        <string level="3" ID="name" />
                        <string level="3" ID="modelName" />
                        <string level="3" ID="ownerName" />
                    </One-Dimensional>
                </Numerical>
            </InputDataTypes>
            <OutputDataTypes level="1" ID="outputDataTypes">
                <Numerical level="2" ID="numericalId">
                    <One-Dimensional level="3" ID="objId">
                        <string level="3" ID="name" />
                        <string level="3" ID="modelName" />
                        <string level="3" ID="ownerName" />
                    </One-Dimensional>
                </Numerical>
            </OutputDataTypes>
        </InformationExchange>
    </ReferenceCapabilityClassStructure>
</Specific>
</CapabilityProfile>
</CapabilityProfiling>
```

Fig. 6. Example of a Specific part of capability profile

4.2 Capability profile to an API

Once the common and the specific part of a capability profile are filled, the difficulty resides on how to reach a service from this formal structure (xml file). The solution that we propose is to transform the capability profile to an OpenAPI Specification, allowing to generate the interfaces of the new service. The following table 4.2 shows the correspondence to be carried out, and a proof of concept is presented in section 5 applied on an open source application.

5 Use case

In order to validate our approach, we applied characterization and servitization process on an application called Collaborative Asset Manager (CAM) from FIT-

Table 1. Capability profile to an API

Capability Profile XML Tag	*OpenAPI Specification property*
<Owner> <ComputingFacilities> <Performance>	API Description
<Function id=””>	API Path
<Function action=””>	Method of API
<InformationExchange> <InputDataTypes>	Parameters of the API
<InformationExchange> <OutputDataTypes>	Response Objects Parameters of API

MAN [5] Project. The FITMAN-CAM app is a web-based, integrated platform for the management of virtualized Assets in the scope of service-oriented Manufacturing Ecosystems (the term asset represents any item of economic value owned by an Enterprise). This application offers CRUD operations of virtualized assets (Create, Read, Update, Delete) using the user interface, and exposes its own REST-based APIs to retrieve information about assets from database. For matter of reusing this application for new business requirements, there is a need to create assets without going through the user interface and orchestrate this action with other services. Our implementation of the servitization step helps to generate project skeleton in order to expose the REST-Based API of any functionality already characterized by the proposed ISO 16100 capability profile. The characterization step (which is done manually for now) of the create asset method gives the profile on the left of figure 7.

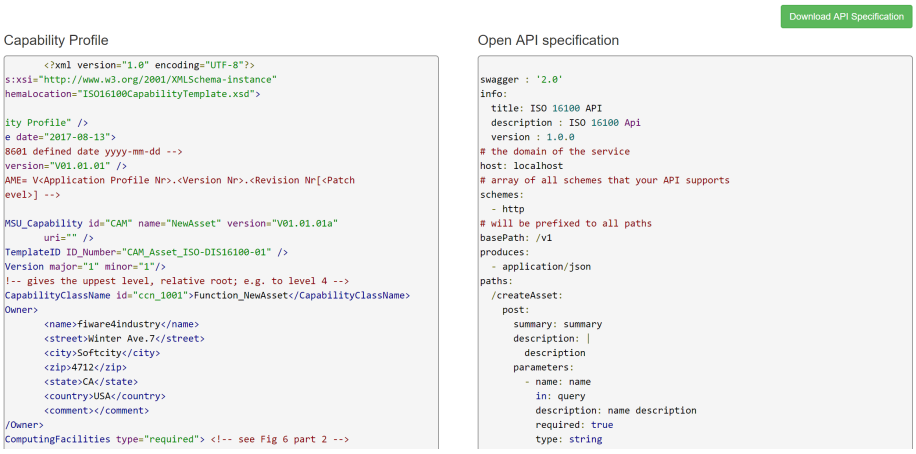

Fig. 7. Application screen shot : Capability profile to Swagger

[5] http://www.fitman-fi.eu/

The content of the capability profile reflect the signature of Java method *public void createAsset(String name, String modelName, String ownerName)*. Once the profile filled, the next step is to upload the capability profile to our developed application, in order to transform it into an Open API Specification following the steps described in section 4.1. The generated Open API Specification is described using YAML, a data serialization standard. From this stage, we generate the server stubs and client SDKs, by using Swagger Codegen process. The last step is the implementation of the business side and deployment of the generated API on an API Manager running on Cloud to get a reusable component for service orchestration.

6 Conclusion and future work

In this paper, we have presented the characterization and servitization steps of the proposed framework with a proof of concept applied on a open source application. Starting from the characterization of a functionality done in a readable and standardized way, using a proposed ISO 16100 capability template with its capability class described with XML Schema. Then we provide an automated solution for the externalization of the characterized functionality as a REST-Based Application Programing Interface for service orchestration. As a result, we generate reusable components for service orchestration. As future work, we plan to focus on the source code analysis step. The aim of this step is to transform the source code of the open source application into a queryable model, and generate a visual to explore and qualify the existing services. The analysis step is followed by the evaluation step, where the use of metrics will help to elect reliable candidates to characterize and expose as presented in this work.

7 Acknowledgment

This paper presents work developed in the scope of the project vf-OS. This project has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement no. 723710. The content of this paper does not reflect the official opinion of the European Union. Responsibility for the information and views expressed in this paper lies entirely with the authors.

References

1. Iso 15745, industrial automation systems and integration open systems application integration frameworks, iso/tc/184/sc5, 2000.
2. Iso 16100-2:2003 industrial automation systems and integration manufacturing software capability profiling for interoperability part 2: Profiling methodology, 2003.

3. Iso 16100-3:2005 industrial automation systems and integration manufacturing software capability profiling for interoperability part 3: Interface services, protocols and capability templates, 2005.
4. Iso 16100-1:2009 industrial automation systems and integration manufacturing software capability profiling for interoperability part 1: Framework, 2009.
5. S. Alahmari, E. Zaluska, and D. De Roure. A service identification framework for legacy system migration into soa. In *2010 IEEE International Conference on Services Computing*, pages 614–617, July 2010.
6. Asil A. Almonaies, Manar H. Alalfi, James R. Cordy, and Thomas R. Dean. Towards a framework for migrating web applications to web services. In *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '11, pages 229–241, Riverton, NJ, USA, 2011. IBM Corp.
7. Asil A Almonaies, James R Cordy, and Thomas R Dean. Legacy system evolution towards service-oriented architecture. In *International Workshop on SOA Migration and Evolution*, pages 53–62, 2010.
8. Abdelhadi Belfadel, Jannik Laval, Chantal Bonner Cherifi, and Néjib Moalla. Capability Profile for Enterprise Application Integration. In *2017 International Conference on Engineering, Technology and Innovation (IEEE/ICE/ITMC)*, 2017.
9. Jean-Claude Boldrini, Guy Caverot, and Maxime Ezequel. The journey in Open Innovation to develop a SME: A longitudinal case study in a French robotics company. working paper or preprint, April 2017.
10. Brajesh. *API management : an architect's guide to developing and managing APIs for your organization*. Apress, New York, 2017.
11. W. B. Frakes and Kyo Kang. Software reuse research: status and future. *IEEE Transactions on Software Engineering*, 31(7):529–536, July 2005.
12. Hidetoshi Kambe, Shinji Kitagami, Jun Sawamoto, Hiroyasu Mitsui, and Hisao Koizumi. A method for analyzing and visualizing intermodule relations to support the reuse-based embedded software development. 100(7):18–31.
13. Selim Kebir, Abdelhak-Djamel Seriai, Sylvain Chardigny, and Allaoua Chaoui. Quality-centric approach for software component identification from object-oriented code. In *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on*, pages 181–190. IEEE, 2012.
14. Kurt Kosanke and R Martin. Enterprise and business processes-how to interoperate? the standards view. In *Workshop on Standards for Interoperability*, 2008.
15. Patrice Muller, Shaan Devnani, Jenna Julius, Dimitri Gagliardi, Chiara Marzocchi, and Editor Karen Hope. *ANNUAL REPORT ON EUROPEAN SMEs 2015/2016 SME recovery continues*.
16. K Patidar, R Gupta, and Gajendra Singh Chandel. Coupling and cohesion measures in object oriented programming. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(3), 2013.
17. Maryam Razavian and Patricia Lago. A systematic literature review on soa migration. *Journal of Software: Evolution and Process*, 27(5):337–372, 2015. JSME-14-0028.R2.
18. Anas Shatnawi, Abdelhak-Djamel Seriai, Houari Sahraoui, and Zakarea Alshara. Reverse engineering reusable software components from object-oriented apis. *J. Syst. Softw.*, 131(C):442–460, September 2017.
19. M. F. Zibran, F. Z. Eishita, and C. K. Roy. Useful, but usable? factors affecting the usability of apis. In *2011 18th Working Conference on Reverse Engineering*, pages 151–155, Oct 2011.