Using Comp-O to Build and Describe Component-Based Services

Grégory Alary, Nathalie Jane Hernandez, Jean-Paul Arcangeli, Sylvie Trouilhet, Jean-Michel Bruel

Using Comp-O to Build and Describe Component-Based Services

G. Alary, N. Hernandez, J.-P. Arcangeli, S. Trouilhet, J.-M. Bruel

Institut de Recherche en Informatique de Toulouse
University of Toulouse, France

Comp-O is an extension of OWL-S that we have proposed to describe the semantics of component-based services. Based on a proof-of-concept prototype, this demonstration shows how Comp-O is used to support assistance to developers of component-based services and automatically generate their descriptions for publication and discovery purposes[1].

1 Background and problem

Component-based software engineering [1] consists in building software as assemblies of reusable and versatile software components. Basically, this paradigm emphasizes composability and reuse and fosters software flexibility. In such a way, an application can be modified by replacing one component by another in the assembly.

Software components [2] are runtime units that implement and *provide* services that we call "component-based services". Symmetrically, at the same level as the services they provide, software components exhibit the services they *require* to be operational. Thus, since the provided and required interfaces are explicit, they are easily composable. Composing components consists in binding the components' required services to the provided ones to deliver composite services with added value. In order to make a component fully operational, i.e., actually providing its component-based services, its required services must be bound to (so realized by) a service that is provided by another component. As an example, Fig. 1 represents the implementation of a component-based service that provides the apparent temperature in Fahrenheit.

Developing composite services is a challenging task as it requires identifying components and services that are compatible, binding them to implement the service, and describing it semantically for discovery by third parties. The semantics of a component-based service depend on the semantics of the services that are required by the provider component. Since the latter are abstracted in the provider component, the actual semantics depend on the semantics of the provided services they are bound to. In a way, the semantics of a composite service are distributed among the components. Therefore, they must be synthesized from the semantics of the components that compose the assembly.

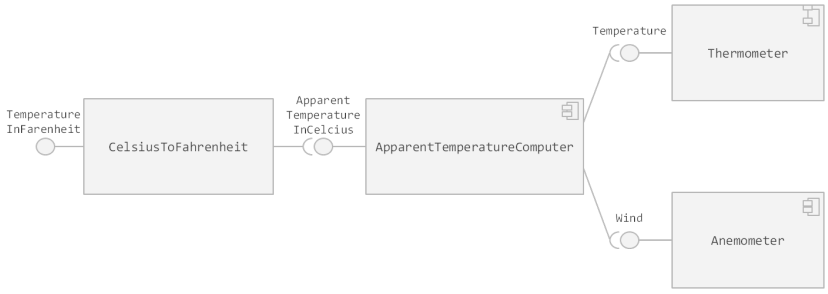

Fig. 1: Implementation of a component-based service

The problem is to describe the services provided by components that have required services, both to enable assistance to service developers when they assemble components and to combine such descriptions to automatically generate composite service descriptions. For that, the vocabulary used to describe component-based services must take into account the required services, and these atomic descriptions must be combinable. State of the art shows that considering ontologies when describing services improves their discoverability [3] and composition [3][4]. Several ontologies and approaches exploiting them have thus been proposed. However, existing solutions mainly consider Web services and are limited to the description and composition of provided services without considering required services. In this work, we propose to describe component-based services with ontologies in order to leverage the semantics of such knowledge representations regarding two issues : (i) to support a detailed description of composite services; (ii) to support the composition of services and produce a description of a composite service depending on the components participating in the assembly.

The next section presents the ontology we propose, called Comp-O. Then, Section 3 introduces a proof-of-concept implementation and shows how Comp-O supports the building of composite services and the automatic generation of their semantic descriptions.

2 Comp-O, an OWL-S extension for component-based services

The development of our ontology, compliant with the NeOn methodology [5], is detailed in [6]. The requirements of a component-based service ontology tested against six ontologies (hRests[2], MSM[3], OWL-S[4], SAREF[5], SOSA/SSN[6] and WSML[7]) motivated the reuse of OWL-S. With OWL-S, the service offered can be

[2] `https://lov.linkeddata.es/dataset/lov/vocabs/hr`
[3] `https://lov.linkeddata.es/dataset/lov/vocabs/msm`
[4] `https://www.w3.org/Submission/OWL-S/`
[5] `https://ontology.tno.nl/saref/`
[6] `https://www.w3.org/TR/vocab-ssn/`
[7] `https://www.w3.org/Submission/WSML/`

described using the pre/post conditions that are annotated in the service profile. The invocation of other services and the internal orchestration are described in the service process.

Comp-O[8], the extension of OWL-S we have proposed, is available on line at `https://comp-o.github.io/comp-o`. It allows representing service required interfaces. This representation can be exploited when defining composite services. Comp-O consists of

- three new classes:
  - *comp-o:ComponentBasedService*,
  - *comp-o:RequiredPerform*,
  - *comp-o:ServiceContract*,
- and one object property *comp-o:requiredPerformContract*.

The class *comp-o:RequiredPerform* is defined to describe the required interfaces of a component-based service. It specialises *owls-process:Perform*. This control construction references a *comp-o:ServiceContract* represented by a service profile that does not reference any process (a black-box). The class *comp-o:ServiceContract* specialises *owls-profile:Profile*. It is used to define the types of inputs and outputs and pre/post-conditions specified by a required interface. The class *comp-o:ComponentBasedService* specialises *owls-service:Service*. It is defined to describe a service that can have no or several *comp-o:RequiredPerform* (required interface) in its process, and that is not operational until all its *comp-o:RequiredPerform* are replaced with an actual perform referencing another process.

Finally, the *comp-o:requiredPerformContract* predicate is used to link a *comp-o:RequiredPerform* with the *comp-o:ServiceContract* it requires.

3 Using Comp-O to build composite services and automatically generate their descriptions

To demonstrate our solution, we have developed a proof-of-concept implementation. It is available at `https://github.com/comp-o/comp-o-poc`. In practice, a command line interface helps the user to select and assemble components, and outputs its OWL-S description.

A demonstration video is available online at `https://www.irit.fr/~Sylvie.Trouilhet/demo/iswc2020.mp4`. It briefly introduces the concepts of software components and component-based services using a simple example in the field of ambient systems. It sets out service description and discovery issues. Then, the above example is used to illustrate what can be done using Comp-O: relying on service descriptions as inputs, the developer is assisted when building a component-based service and its description is automatically generated.

[8] The prefixes used are comp-o: `https://comp-o.github.io/comp-o#`, owls-service: `http://www.daml.org/services/owl-s/1.2/Service.owl#`, owls-process: `http://www.daml.org/services/owl-s/1.2/Process.owl#`, owls-profile: `http://www.daml.org/services/owl-s/1.2/Profile.owl#`

The building process is described below and summarized in Fig. 2. A list of the available component-based services is first presented. To do this, all that is needed is to retrieve the set of resources typed by the *service:Service* class. Then, the developer must choose the "root" service, i.e., the service to implement. Comp-O helps to determine whether the component that provides it has any required interface, i.e., if one of the control constructions of the process of the service it provides is a *comp-o:RequiredPerform*. This property can be comprehensively checked considering OWL-S vocabulary using a SPARQL query.

If the component providing the service does not require any service, it can be described and published as an OWL-S service. If the component has one or more required services, the latter must be bound to external component-based services. If so, to ease the binding decisions, it can be determined whether a provided service matches a required one. This requires checking that the types of the inputs and outputs, the preconditions and the post-conditions match. This task must be repeated as long as there are unbound required services in the assembly. Then, the component-based implementation of the chosen service is operational, and its OWL-S description can be generated.

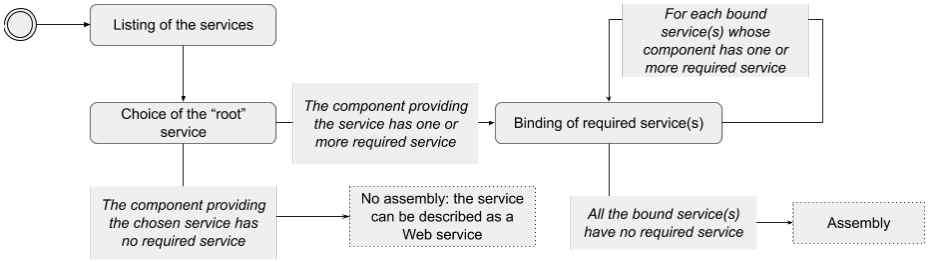

Fig. 2: Building process of a component-based service

To generate the semantic description, the first step consists in replacing every *comp-o:RequiredPerform* by an *owls-process:Perform* referencing the process associated in the assembly, using the *owls-process:process* predicate instead of referencing a *comp-o:ServiceContract* via the *comp-o:requiredPerformContract* predicate. The process of a component-based service can reference as variables the inputs and outputs of a *comp-o:ServiceContract* it requires. For each service, the second step is therefore to replace the references to these variables by references to the equivalent variable of the associated service. This step can be accomplished by processing all the *owls-process:fromProcess* predicates having as object a resource of the type *comp-o:ServiceContract*. Once these two steps are complete, the component-based services are then described as OWL-S services.

4 Conclusion

In this work, we have experimented Semantic Web technologies in the field of component-based software engineering. This paper has introduced Comp-O, an extension of OWL-S for component-based services which are services provided by software components, and the use of Comp-O to support the design of component-based services. Our proposal contributes to the state of the art in terms of assistance to component-based software developers and automated description of component-based services.

In a near future, we plan to use Comp-O in an ongoing project which aims to make user-oriented services emerge at runtime in ambient environments. There, an intelligent engine builds on the fly component-based services from software components that are present at the time in the environment, without having been required by the user. As a consequence, emerging services must be semantically described to the user [7] who can, in addition, edit them and participate to their design while being assisted in such a task.

5 Acknowledgment

This work is part of the AILP (Assistance InteLligente et proactive en environnement Professionnel) project, which is supported by the French region Occitanie and the operational program FEDER-FSE Midi-Pyrénées et Garonne.

References

1. I. Sommerville. Component-based software engineering. In *Software Engineering*, chapter 16, pages 464–489. Pearson Education, 10^{th} edition, 2016.
2. OMG. *Unified Modeling Language (OMG UML) Version 2.5.1*, chapter 11.6.3.1 Components semantics. 2017.
3. M. Klusch, P. Kapahnke, S. Schulte, F. Lecue, and A. Bernstein. Semantic Web Service Search: a Brief Survey. *KI-Künstliche Intelligenz*, 30(2):139–147, 2016.
4. K. Kurniawan, F.J. Ekaputra, and P.R. Aryan. Semantic Service Description and Compositions: A Systematic Literature Review. In *ICICoS*, pages 1–6, 2018.
5. M. C. Suárez-Figueroa, A. Gómez-Pérez, and M. Fernández-López. *The NeOn Methodology for Ontology Engineering*, pages 9–34. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
6. G. Alary, N. Hernandez, J.-P. Arcangeli, S. Trouilhet, and J.-M. Bruel. Comp-O: an OWL-S Extension for Composite Service Description. In *Proc. of the 22^{nd} Int. Conf. on Knowledge Engineering and Knowledge Management (EKAW)*, 2020. To appear.
7. M. Koussaifi, S. Trouilhet, J.-P. Arcangeli, and J.-M. Bruel. Automated user-oriented description of emerging composite ambient applications. In *Proc. of the 31^{st} Int. Conf. on Software Engineering and Knowledge Engineering (SEKE)*, pages 473–478, 2019.