Supporting Service Versioning
MDE to the Rescue

Iván Santiago, Juan M. Vara, Jenifer Verde, Valeria de Castro and Esperanza Marcos
Kybele Research Group, Rey Juan Carlos University, Avd. Tulipán S/N, Móstoles, Madrid, Spain

Keywords: Service Orientation, Model-Driven Engineering, Traceability.

Abstract: In the field of Service-Oriented Architecture (SOA), evolution is a key issue given the non-trivial nature of updating widely distributed and heterogeneous systems. In particular, the evolution of a service is expressed through the creation and decommissioning of different service versions during its lifetime. These versions must be aligned with each other in such a way as to allow a service developer to track the various modifications introduced over time and whether the resulting service version is compatible with existing consumers. Having all this in mind, this work aims at define a plan to provide a complete framework to support service evolution by means of Model-Driven Engineering techniques.

1 INTRODUCTION

Evolution is inherent to software systems because of the rapid improvement of technologies and business logic. In the field of Service-Oriented Architecture (SOA), evolution is a key issue given the non-trivial nature of updating widely distributed and heterogeneous systems (Papazoglou and van den Heuvel, 2007). To alleviate the inherent complexity of evolving service-based systems, this paper takes a step further a proposal to deal with service evolution using Model-Driven Engineering (MDE) techniques (Schmidt, 2006).

Formally speaking, service evolution is the disciplined approach of managing service changes and is defined as *the continuous process of development of a service through a series of consistent and unambiguous changes* (Andrikopoulos et al., 2012). The evolution of a service is expressed through the creation and decommissioning of different service versions during its lifetime. These versions must be aligned with each other in such a way as to allow a service developer to track the various modifications introduced over time and their effects on the original service.

To control service development, a developer needs to know why a change was made, what its implications are, and whether the resulting service version is compatible with existing consumers. A service version is compatible if it does not render its consumers inoperable, i.e., it does not break them in the sense that consumers are still able to use the same type of data they used as inputs and get back the same type of data they got as outputs.

This way, our proposal aims at providing a model-driven technological framework to support service evolution at interface level. To that end, in previous works (Vara et al., 2012) we presented a DSL toolkit for modelling the structural part of Abstract Service Descriptions (ASD) and a reasoning mechanism that assesses whether two versions of a service are compatible with respect to its consumers. Such work served to provide a proof of concept of the possible synergy between MDE and Service Orientation. As well, it served to lay the foundation for future work. This paper is the first stage of such work: it defines the following steps to progress in the building of the service evolution framework.

To that end, the rest of this work is structured as follows: Section 2 presents the building of technological bridges to move from WSDL descriptions to ASDs; Section 3 introduces the materialization of the textual reports about service versions compatibility into trace models; Section 4 defines a model-driven process to support contracts generation and finally, Section 5 concludes by highlighting the main contribution of this paper.

2 FROM WSDL DOCUMENTS TO ASD MODELS

In (Andrikopoulos et al., 2012) present a rigorous for-

Santiago I., M. Vara J., Verde J., de Castro V. and Marcos E..
Supporting Service Versioning - MDE to the Rescue.
DOI: 10.5220/0004523402120217
In *Proceedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering* (ENASE-2013), pages 212-217
ISBN: 978-989-8565-62-4

mal framework based on type safety criteria and algorithms which controls and delimits the evolution of services. The framework extends and applies theories and methods of controlling evolution from object-oriented programming languages like subtyping and co- and contra-variance of input and output (Meyer, 1997).

Based on these principles a reasoning mechanism is presented that allows deciding when a change to the service interface leads to a compatible version of the service for the consumers. More specifically, the framework is based on a technology-agnostic notation for the representation of service interfaces in the form of Abstract Service Descriptions (ASDs).

In order to provide tooling support for such framework, a DSL toolkit to model ASDs was introduced in previous works (Vara et al., 2012). Nevertheless, despite there are different initiatives for services representation, most of existing services use WSDL (W3C, 2001) to provide a standard description of their interface. Therefore, to widen the scope of the approach the first task to address is the building of technological bridges to extract the information gathered in any existing WSDL document to express it in terms of the DSL already introduced for ASD modeling. As a result, the level of abstraction at which reasoning about the interface of any given set of services is made can be raised.

The process to extract ASD models from WSDL files is briefly illustrated in Figure 1.

- The starting point is one or more WSDL les describing different versions of the same service, namely S_1 and S_2. Being XML les, they conform to the WSDL XML Schema (`WSDL.xsd`).
- In order to bring them to a model-engineering context, a TCS (Jouault et al., 2006) text-to-model transformation (`XML2Ecore`) is used to inject such les into two XML models, which conform to the XML metamodel (`XML.ecore`). The objects of these models are basically XML elements and attributes. This way this transformation allows moving from grammarware to modelware (Wimmer and Kramler, 2006).
- Next, we map the XML models to WSDL models conforming to the WSDL metamodel (`WSDL.ecore`). To do so we use an ATL (Jouault et al., 2008) model-to-model transformation (`XML2WSDL`).
- Another model-to-model transformation (`WSDL2-ASD`) extracts the structural and non-functional information of the WSDL models to produce two ASD models that conforms to the ASD metamodel (`ASD.ecore`). Such models provide high-level descriptions of the services interfaces, which abstract completely of any technological detail.

It is worth mentioning that the development of the inverse transformation chain will be addressed as well. Thus, a WSDL document could be derived from an ASD model. To that end, note that when moving from WSDL to ASD all the non-structural data is lost. Figure 2 shows the solution that will be adopted in order to solve this issue.

To support the extraction of complete WSDL files, such data will be collected into partial WSDL models. Then, the different reasoning will be performed over the ASD by means of the MDE processing techniques needed. Once the ASD desired is obtained, it will be merged with the partial WSDL model (containing the non-structural information). As a result a WSDL model that can be directly serialized into a WSDL document will be obtained.

3 TRACKING SERVICE VERSIONS COMPATIBILITY

As mentioned before, a first version of the service version comparer was introduced in previous works. In particular, the Service Representation Modeler (SR-Mod[1]) provides a textual report on the compatibility of two given versions of the same service. Such assessment is based on evaluating if sub-typing relationships hold between the elements of both versions.

Next movement on the development of the MDE framework to support service evolution is to extend the use of MDE technologies to improve the handling of the comparison process output. More specifically, version comparison results will be expressed in the shape of traces model (Santiago et al., 2012) that relates the ASD models representing each version of the service. This way, each element of the trace model would inform of the level of compatibility between two given elements of the service versions compared, together with a short description of the issues related to the compatibility assessment. The use of trace models to represent the output of compatibility assessment results in two main advantages:

- On the one hand, being persisted as models, such results could be later processed by means of any other MDE technique, such as model merging or model checking (Bernstein, 2003).
- On the other hand, the nature of the information gathered from the assessment process is eminently relational. It consist mainly of statements

[1] http://srmod.wordpress.com/

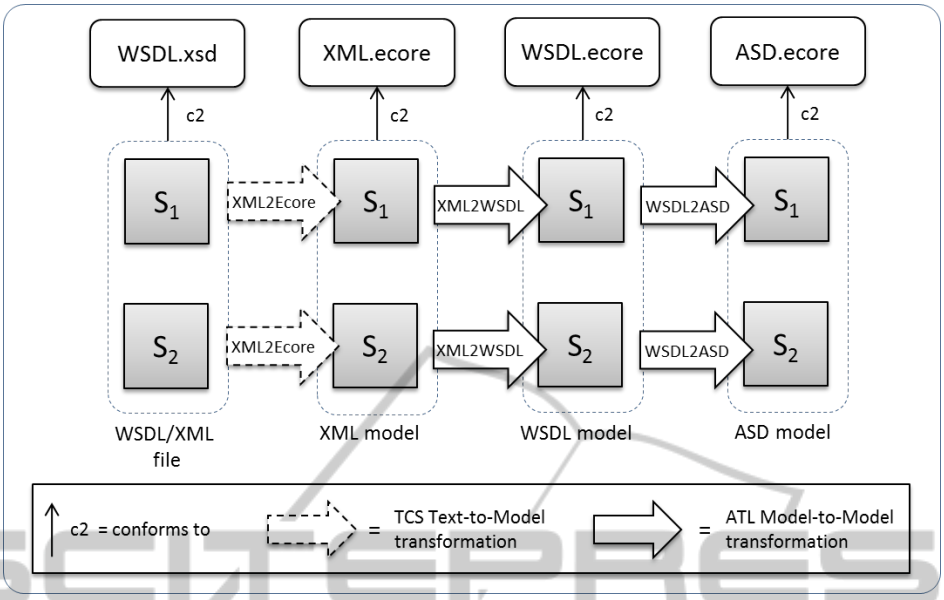

Figure 1: From WSDL files to ASD models.

such as foo element (from S_1) is (non-)compatible with bar element (from S_2). Hence, the use of proper tooling could ease the understanding and comprehension of this information whose nature is mainly relational.

All in all, iTrace[2] will be used to collect the output of SRMod into a trace model. The iTrace framework was devised to support the management and analysis of traceability information in MDE projects. Note that from a high-level point of view, a trace object is nothing but the materialization of the relationship between two or more artifacts. Therefore, traces can be used to collect information about the relationship (whether they are compatible or incompatible) between two or more objects of two given ASD models. Broadly speaking, distinguish two types of traces will be distinguished: `Compatible` and `Incompatible`.

Figure 3 illustrates the idea by showing a simplistic scenario. The comparison of two given ASDs, namely S_1 and S_2 produces a trace model that informs of the compatibility relationships between the elements of S_1 and S_2. For instance, the trace model shows that there is no equivalent element for A_1 in S_2; both B_1 and C_1 do have an equivalent and compatible object in S_2, namely B_2 and C_2; and finally, though D_1 is equivalent to D_2, they are not compatible. Moreover, incompatible trace objects will contain low-level information on the reasons that resulted in the non-compatible relationship between the referenced objects.

[2]http://www.kybele.etsii.urjc.es/itracetool/

4 SERVICE CONTRACTS

The next step in demonstrating the suitability of MDE techniques in a SOA setting, a similar approach to the one described below is proposed to be applied to existing work on *service contracts*.

According to (Andrikopoulos et al., 2012), a service contract is an intermediary construct interposed between service providers and consumers, expressed also in ASD form, which can be used to represent a technical Service Level Agreement (SLA) between them. The use of contracts allows for greater flexibility in evolving both interacting parties (i.e., providers and consumers) in a compatible manner. Furthermore, even the contract itself can evolve under certain conditions. The fact that the authors (re-)use the ASD notation and the subtyping relation discussed so far provide an ideal setting for an extension of our ongoing work.

4.1 Service Contracts Generation

Figure 4 provides an overview of the approach proposed to produce a service contract from two given versions of a service.

The first step is supported by an ATL refining transformation (`ASDRefine`) that produces enriched versions of the original ASD models representing each version of the service. Such refinement consists of adding annotations to the elements of the ASD model. Each annotation states whether there is a compatible element in the other ASD model for the one

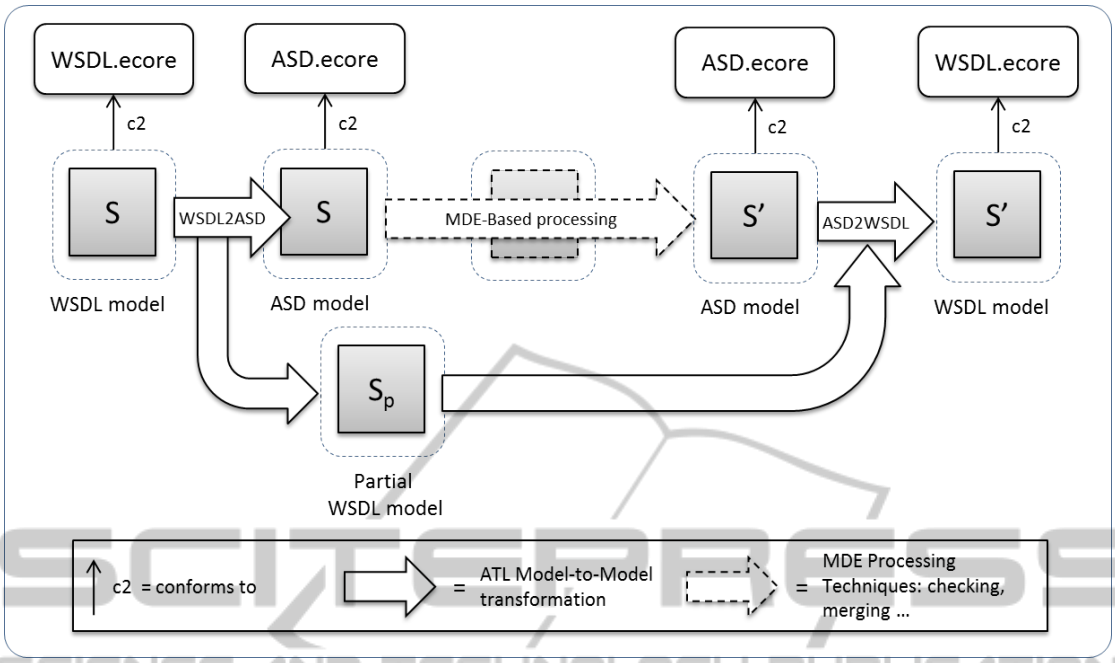

Figure 2: From ASD models to WSDL models.

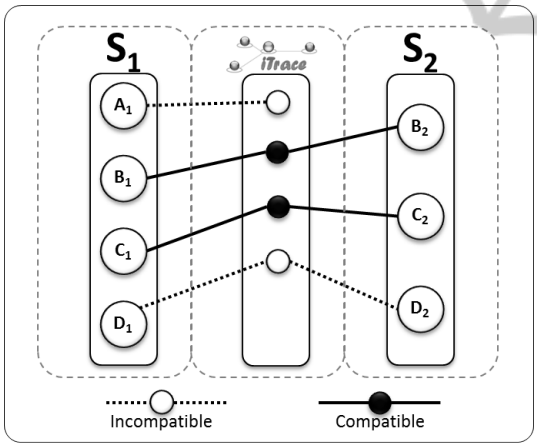

Figure 3: Using trace models to represent service versions compatibility.

being annotated. Consequently, each ASD model has to be annotated with regard to the other ASD. That is, S1 is annotated with regard to S2 and vice versa. As a result, the S_{12} and S_{21} models are obtained.

To that end, the following annotations are defined (recall that they are added to the elements of the ASD models). They are explained from the point of view of S_{12}. The same applies for S_{21}:

- `None`: there is no equivalent in S_{21} (there is no element owning the same name with the same type).
- `Equiv`: there is an equivalent in S_{21}. Note that this annotation refers just to the element itself. Nothing is said about its nested elements of attributes.
- `Super`: the element is a super-type of an element in S_{21}. That is to say that the number of elements nested in the annotated one is greater or equal than the number of elements of the corresponding element in S_{21}. Besides, such nested elements are more generic than those of the corresponding element and the value of its attributes are greater or equal than the value of the attributes of the corresponding element.
- `Sub`: the element is a sub-type of an element in S_{21}. The conditions that must hold can be derived from inverting those described for super-type annotated elements.

Next, the annotated ASDs are consumed by another ATL transformation (`ASD2Contract`) that produces a new ASD representing the service contract.

Such contract will contain one element of every pair of equivalent elements found in S_1 and S_2 plus some of those elements for which no equivalent was found. This ASD model can be seen as an intermediary construct in the form of a service contract interposed between service providers and consumers. It allows for greater flexibility in evolving both parties in a compatible manner as they relax some of the assumptions regarding the ability of services to evolve while preserving their compatibility (Andrikopoulos et al., 2012).

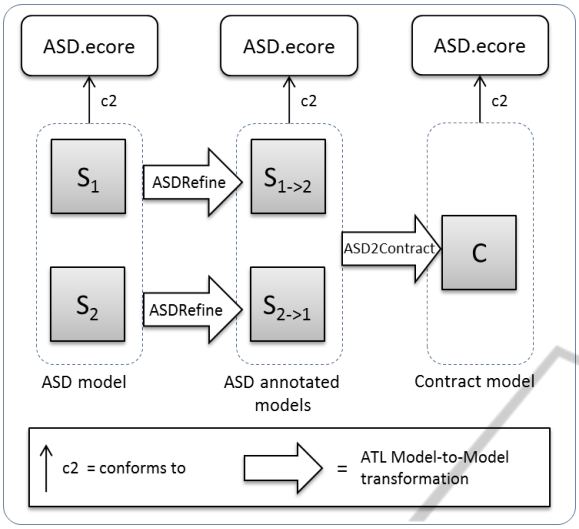

Figure 4: Service contract generation.

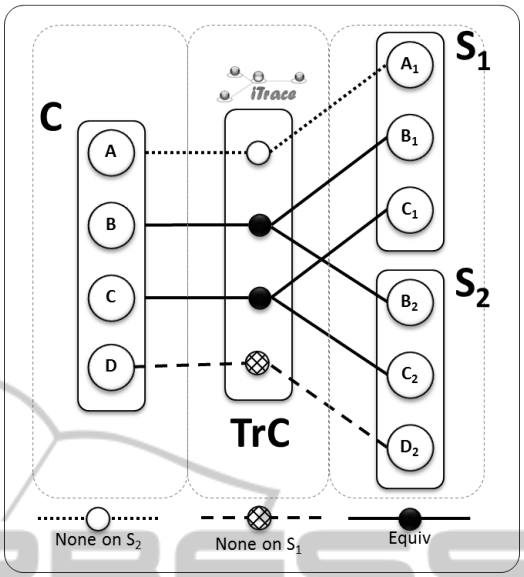

Figure 5: Tracking service contract generation.

4.2 Tracking Service Contracts Generation

Finally, following the idea collected in Section 3, the relationships between the contract and the two service versions are planned to be collected in a trace model. This would help decisively to get at a first sight which the problematic elements in the two versions being compared are: by selecting a given element in the contract, the user would be automatically informed of which elements in S_1 and S_2 are compatible with the selected element (if they exist).

To that end, four types of traces are considered. They correspond to the four types of annotations introduced in the previous section: None, Equiv, Sub and Super. The idea is illustrated in Figure 5: in this case, by selecting the B element in the contract (C.B), the user would know that there are compatible elements both in S_1 and S_2 (namely, $S_1.B_1$ and $S_2.B_2$). By contrast, the selection of A would show that there is no equivalent element in S_2.

In addition, filters can be defined over the trace model (TrC) to show just compatible, non-compatible or whichever selection of elements. In other words, the traces model relating S_1, S_2 and C can be filtered applying MDE-techniques, such as merging or transformation (Bernstein, 2003), to produce ad-hoc trace models. Those models might contain just those traces referring to the compatible or incompatible elements of S_1, S_2 and C.

Note that the trace model (TrC) relating the contract and the service versions would be automatically produced by the transformation that generates the contract. That is, the `ASD2Contract` transformation shown in Figure 4 consumes $S_{1\to2}$ and $S_{2\to1}$ and produces the C contract model and the TrC trace model.

To that end, the ATL transformation is enriched with the necessary machinery to support the production of trace models (Jouault, 2005). The maturity reached by some MDE tools allows the implementation of this kind of model-based implementations (Volter, 2011).

Furthermore, the contract and the trace model could be used to produce new versions of the service that ensure compatibility by fulfilling the contract since the contract defines the requirements that a service version has to meet to be compatible regarding the consumers.

5 CONCLUSIONS

This work sets the following steps to take in order to provide a complete framework to support service evolution by means of MDE techniques. In particular, it defines three main directions for future work: the development of technological bridges to move between WSDL files to Abstract Service Descriptions; the modeling of the output of service versions compatibility assessment as trace models; and the generation and tracking of service contracts (in the sense of Server Level Agreements).

The underlying basis of the proposal is a theoretical framework that allows the formal definition of the conditions under which the evolution of service interfaces respects service compatibility. Such framework depends on formal models for the representation of service interfaces that draw from a common

metamodel. In this context, the application of MDE techniques results ideal to provide with a prototype supporting the theoretical framework.

All in all, this paper is another stage in a research line initiated in previous works that demonstrates the synergy between MDE and SOA by discussing how a service-based theoretical framework can be implemented by means of MDE techniques and tools. The premise is that almost any SE problem can be expressed in terms of MDE since almost every software artifact could be abstracted as a model. Once there, one can benefit from the advantages brought by MDE in the form of less costly, rapid software development by leveraging the level of automation in the development process.

ACKNOWLEDGEMENTS

This research has been carried out in the framework of the MASAI project (TIN-2011-22617) and the Technical Support Staff Subprogram (MICCINN-PTA-2009), which are partially financed by the Spanish Ministry of Science and Innovation.

REFERENCES

Andrikopoulos, V., Benbernou, S., and Papazoglou, M. (2012). On the evolution of services. *Software Engineering, IEEE Transactions on*, 38(3):609–628.

Bernstein, P. (2003). Applying model management to classical meta data problems. In *First Biennial Conference on Innovative Data Systems Research*, pages 1–10, Asilomar, CA, USA.

Jouault, F. (2005). Loosely coupled traceability for atl. In *Proceedings of the Traceability Workshop of the First European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA 2005)*, volume 91, pages 29–37, Nuremberg, Germany.

Jouault, F., Allilaire, F., Bézivin, J., and Kurtev, I. (2008). ATL: A model transformation tool. *Science of Computer Programming*, 72(1-2):31–39.

Jouault, F., Bézivin, J., and Kurtev, I. (2006). TCS: a DSL for the specification of textual concrete syntaxes in model engineering. In *Proceedings of the 5th international conference on Generative programming and component engineering*, GPCE '06, pages 249–254, New York, NY, USA. ACM.

Meyer, B. (1997). *Object-oriented software construction (2nd ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Papazoglou, M. and van den Heuvel, W.-J. (2007). Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal The International Journal on Very Large Data Bases*, 16(3):389–415. 10.1007/s00778-007-0044-3.

Santiago, I., Jiménez, A., Vara, J. M., De Castro, V., Bollati, V., and Marcos, E. (2012). Model-Driven Engineering As a New Landscape For Traceability Management: A Systematic Review. *Information and Software Technology*, 54(12):1340–1356.

Schmidt, D. C. (2006). Model-Driven Engineering. *IEEE Computer*, 39(2):25–31.

Vara, J. M., Andrikopoulos, V., Papazoglou, M. P., and Marcos, E. (2012). Towards model-driven engineering support for service evolution. *Journal of Universal Computer Science*, 18(17):2364–2382.

Volter, M. (2011). From Programming to Modeling - and Back Again. *Software, IEEE*, 28(6):20–25.

W3C (2001). Web services description language (wsdl) v1.1. http://www.w3.org/TR/wsdl.

Wimmer, M. and Kramler, G. (2006). Bridging grammarware and modelware. In *Proceedings of the 2005 international conference on Satellite Events at the MoDELS*, MoDELS'05, pages 159–168, Berlin, Heidelberg. Springer-Verlag.