Model-Driven Development of Service Compositions for Enterprise Interoperability

Ravi Khadka[1], Brahmananda Sapkota[2], Luís Ferreira Pires[2], Marten van Sinderen[2], and Slinger Jansen[1]

[1] Utrecht University, P.O. Box 80.089, 3508TB Utrecht, The Netherlands
{ravi,s.jansen}@cs.uu.nl
[2] University of Twente, P.O. Box 217, 7500AE Enschede, The Netherlands
{b.sapkota,l.ferreirapires,m.j.vansinderen}@ewi.utwente.nl

Abstract. Service-Oriented Architecture (SOA) has emerged as an architectural style to foster enterprise interoperability, as it claims to facilitate the flexible composition of loosely coupled enterprise applications and thus alleviates the heterogeneity problem among enterprises. Meanwhile, Model-Driven Architecture (MDA) aims at facilitating the development of distributed application functionality, independent from its implementation using a specific technology platform and thus contributes to deployment in different platforms. In this paper we propose an MDA-based transformation technique for service composition. The contribution of the paper is two-fold. First, our approach shows how enterprise interoperability is supported by service composition at two different technical levels, namely at choreography and orchestration level. Second, the approach contributes to the management of changes that affect enterprise interoperability, by defining a (semi-)automated transformation from choreography to orchestrations in which the interoperability constraints specified at the choreography level are preserved.

Keywords: SOA, MDA, Metamodel Transformation, Enterprise Interoperability, Choreography, Orchestration, Service Composition, Service Interoperability.

1 Introduction

Enterprise interoperability denotes the ability of organizations, hereafter called enterprises, to interoperate in order to achieve certain business goals [13]. To a large extent, the competitiveness and success of modern enterprises is determined by their ability to achieve enterprise interoperability. Advances in Internet technologies have contributed substantially to the propagation and efficiency of cross-organizational collaboration. Nonetheless, important obstacles remain due to business-driven proprietary developments. Therefore, enterprise interoperability problems should be addressed coherently at business and technical levels as a whole [22]. Enterprises are especially challenged by the accelerating pace of changes, such as intra-organizational changes, changes in market demands

M. van Sinderen and P. Johnson (Eds.): IWEI 2011, LNBIP 76, pp. 177–190, 2011.

and opportunities, and, consequently, changes in partners and ways and intent of collaboration, and the changes in supporting technologies. To manage these changes, automatic support is essential.

Service-Oriented Computing (SOC) is one of the promising technologies to realize the necessary automated support for change management. The underlying concept of SOC is service, which is described in [16] as a self-contained, platform-agonistic computational element that supports rapid, low-cost and easy composition of loosely coupled distributed software applications. A service can be described, published, and discovered by an enterprise with a focus on reusing external behavior and hiding implementation details. However, in an enterprise collaboration, no single service alone may be available that fulfills all collaboration goals. A typical approach to this problem is to identify the basic (non-composite) services, based on an analysis of the collaboration goals, and then aggregate them into a larger composite service with the appropriate value-added functionality. We call this process of aggregating services "service composition", which not only realizes a required (composite) service, but also accelerates application development through the application of reuse at service level [11].

Service compositions can be considered at different abstraction levels, notably at choreography and orchestration levels [17,2]. A service choreography is a decentralized perspective, which describes the public message exchanges, and thus defines how participating services should interact with each other. At a lower level, it is necessary to define how to realize the responsibilities specified at the choreography level in terms of the concrete processes. A service orchestration is a centralized coordination of participating services, which defines the message exchanges along with the necessary internal actions, like data transformations and internal function invocations [17]. In an enterprise collaboration, a service choreography specifies requirements in terms of message exchanges to support the collaboration goals, while a service orchestration realizes the required message exchanges in terms of executable processes.

It has been widely recognized that SOC technology and SOA have emerged as evolutionary steps to achieve enterprise interoperability [13]. Enterprises can publish their services to alleviate the heterogeneity problem. The collaboration among these enterprises is realized by composing their individual services through the service composition process. In this phase, we need service interoperability at the technical level to realize enterprise interoperability. In the service composition process we specify the interoperability constraints at service choreography level, which focuses on the message exchanges between the participating services. At a lower level, such as in the service orchestration level, interoperability constraints as specified at the choreography level are preserved. Therefore, an approach is needed to transform a given choreography to one or more orchestration(s). Currently these transformations are mostly performed manually [12], which is a time consuming and error prone. A (semi-)automated transformation from choreography to orchestrations, such that it preserves the interoperability constraints specified at the choreography level and automates the enterprise collaboration process, would represent an improvement. In this

paper, we propose an approach to (semi-)automate the transformation from choreography to orchestrations using metamodel transformation techniques from MDA [15]. We refer to this process as model-driven service composition. In this approach, we define the metamodels for choreography and orchestration and mappings between model elements of these metamodels. Our proposed approach of (semi-)automated transformation of choreography into orchestrations improves the productivity of the composition process and correctness of the resulting compositions.

The rest of the paper proceeds as follows: Section 2 introduces our approach to model-driven service composition for enterprise interoperability. Section 3 describes an example scenario that we used for illustrating the proposed approach. Section 4 discusses how this approach was implemented and Section 5 discusses the evaluation of our approach. Section 6 presents related work and finally Section 7 concludes the paper and gives an outlook of future research directions.

2 Approach

Our approach is to perform service composition by (semi-)automatically transforming the given choreography into one or more orchestration(s) using model-driven transformation techniques. In this section, we investigate the relationship between choreography and orchestration, and indicate how model-driven transformation can be useful to relate these concepts.

A choreography describes the public message exchanges, interaction rules and agreements in a service composition at a high level of abstraction [17,2]. A choreography is defined from a decentralized perspective, and specifies how the individual services interact with each other. A service choreography does not describe any internal actions of the participating services, like internal computations or data transformations. A choreography captures message exchanges from a global perspective where all the participating services are treated equally, and as such it provides a convenient starting point to define how the participating services must interact to realize enterprise collaboration. At the choreography level, the interactions among each service participating in the composition are described as a global abstract protocol with which all participating services should comply [2].

After the message exchanges between the participating services are defined in a choreography, we need to define how the added-value of the composite service can be achieved in terms of a concrete implementation. We call the process that realizes the business goal of the enterprise collaboration in terms of executable process an orchestration. There may be one orchestration for each service, or a central authority that coordinates the overall composition process, so called decentralized orchestration and centralized orchestration, respectively [5,4]. In centralized orchestrations, the central authority is called an orchestrator. Orchestrations describe the communication actions and the internal actions, like data transformations or invocations to internal software modules. In an orchestration, execution orders of the interactions and method invocations of the implementations also need to be defined [17,2] in terms of an executable process that contains enough information to enable execution by an orchestration engine.

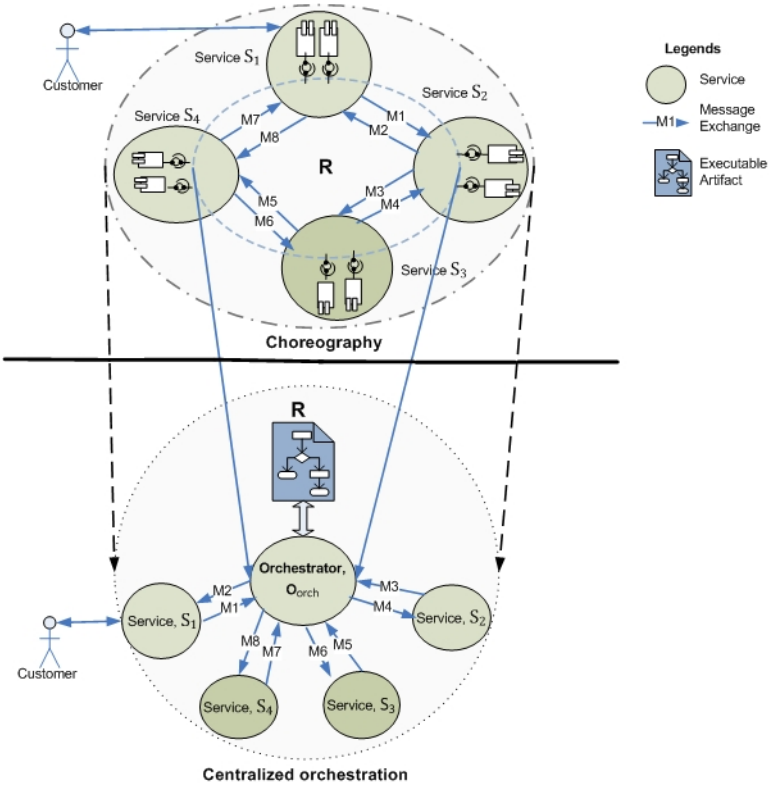

Fig. 1. Transformation from choreography to centralized orchestration

A given choreography can be transformed to either a decentralized or a centralized orchestration. In this paper we only consider the transformation from choreography to centralized orchestration because centralized orchestration is widely used in the service composition process. A diagrammatic representation of the transformation from a choreography to a centralized orchestration is presented in Figure 1. In the choreograph level of Figure 1 the message exchanges and interactions rules are represented by the directed arrows among the services. The overall responsibility of choreography, represented as R, includes message exchange, interactions rules, and the service level constraints. The transformation from choreography to centralized orchestration is achieved by transferring the overall responsibility of choreography R to the orchestrator and establishing the message exchanges, interactions rules, and the interoperability constraints between the orchestrator and the individual services accordingly. The responsibility R is represented as an executable artifact and is depicted as the flow diagram in the orchestration level of Figure 1.

In our approach we specify service choreographies and orchestrations using Web Service Choreography Description Language (WSCDL or CDL in short) [9]

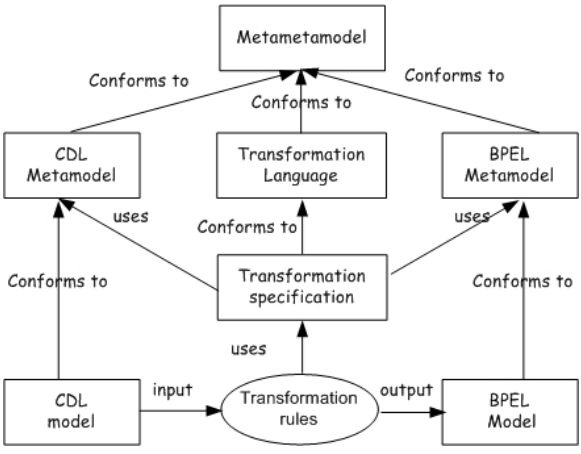

Fig. 2. Model-driven transformation

and Web Service Business Process Execution Language (WSBPEL or BPEL in short) [1], respectively, because of the wide industry acceptance of these languages.

We apply model-driven transformation to (semi-)automate the transformation from a choreography specified in CDL to a centralized orchestration specified in BPEL. The model-driven transformation is diagrammatically presented in Figure 2. In this transformation, we take the CDL metamodel as source metamodel and BPEL metamodel as target metamodel. We used the language syntax of [9] and [1] to develop the metamodels of CDL and BPEL, respectively. We defined the mappings between CDL and BPEL metamodel elements. These mappings are then used to create transformation specifications in Atlas Transformation Language (ATL) [7], and hence executed by ATL transformation engine. The transformation mappings of the metamodel elements of CDL and BPEL are presented in Table 1.

3 Working Example

The Build-To-Order (BTO) application scenario, adopted from [19] is used as an example to illustrate the usability of our approach. We briefly explain this scenario with the sequence diagram shown in Figure 3. The BTO scenario consists of a customer, a manufacturer, and suppliers for CPUs, main boards and hard disks. We consider all the suppliers to be different enterprises. The manufacturer offers assembled IT hardware equipment to its customers. For this purpose, the manufacturer has implemented a BTO business model. It holds a certain part of the individual hardware components in stock and orders missing components if necessary. In the implemented BTO scenario, the customer sends a quote request with details about the required hardware equipment to the manufacturer.

Table 1. Transformation mapping from CDL to BPEL

CDL	BPEL	Remarks
roleType	process per role	*bpel:targerNamespace* attribute is derived from the *cdl:targetNamespace* of cdl:package
participantType	partnerLink	-
relationshipType	partnerLinkType	-
variable	variable	bpel:messageType attribute is derived from the cdl:type of related *cdl:informationType*
channelType	correlationSet	*bpel:properties* is derived from *cdl:name* of *cdl:token* within *cdl:identity*
sequence	sequence	-
Parallel	flow	-
Choice	if-else	*bpel:condition* is manually provided
workunit		
repeat= false , block=false	if-else	*bpel:condition* is manually provided
repeat= true	while	*bpel:condition* is manually provided
block= true	-	no mapping
interaction		
action= request	invoke	current party is mentioned in *cdl:fromRole*
action= request	receive	current party is mentioned in *cdl:toRole*
action= respond	reply	current party is mentioned in *cdl:fromRole*
action= respond	receive	current party is mentioned in *cdl:fromRole* (synchronous reply)
assign	assign	-
finalize	compensationHandler	-
noAction	empty	-
silentAction	sequence with nested empty	BPEL designer have to manually specify the silentActions

The latter sends a quote response back to the customer. As long as the customer and the manufacturer do not agree on the quote, this process is repeated. If a mutual agreement is reached the customer sends a purchase order to the manufacturer. Depending on its hardware stock, the manufacturer has to order the required hardware components from its suppliers. If the manufacturer needs to obtain hardware components to fulfill the purchase order it sends an appropriate hardware order to the respective supplier. In turn, the supplier sends a hardware order response to the manufacturer. Finally, the manufacturer sends a purchase order response back to the customer.

4 Implementation

Based on the BTO application scenario, we specified the choreography in CDL, and (semi-)automatically generated the BPEL based orchestration specification for the Manufacturer as an orchestrator. The orchestration process is specified in BPEL. To implement our proposed approach, we defined the CDL and BPEL

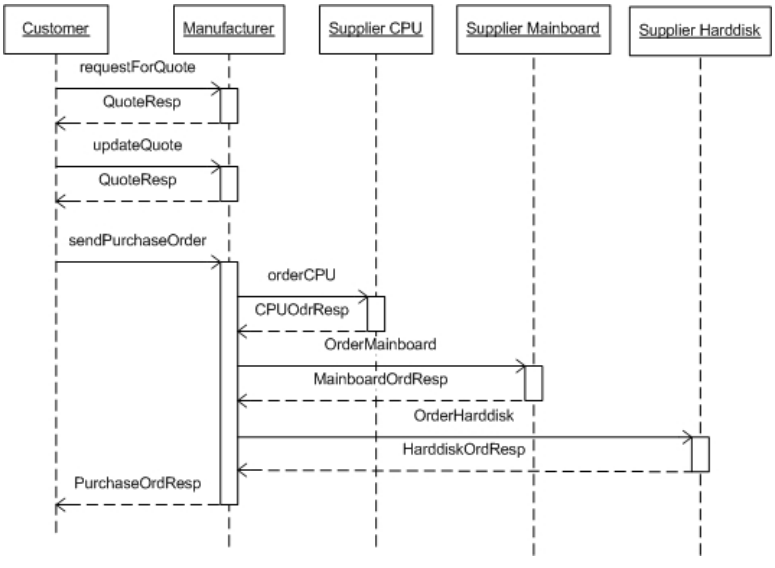

Fig. 3. BTO example

metamodel using Eclipse Modeling Framework (EMF)[1]. EMF is an Eclipse-based modeling framework and has code generation facilities for building tools and other applications based on a structured data model. The transformation specification has been written based on the transformation mapping presented in Table 1 using ATL[2], which contains the ATL transformation engine that runs ATL transformation specifications as transformation rules.

Conceptually, the transformation approach presented in Figure 2 represents the model transformation that takes a CDL model (conforming with the CDL metamodel) and runs the transformation rules in ATL engine to generate a BPEL model (conforming with the BPEL metamodel). However, to execute the ATL transformation rules, an ATL engine expects every model and metamodel (i.e., input/output models and metamodels) to be serialized in XML Metadata Interchange (XMI) format [8], whereas the input/output models (i.e., CDL and BPEL models) in our scenario are in XML format that conforms with the XML schema of CDL and BPEL. Hence, we need a transformation chain that contains the transformations as indicated in Figure 4. This transformation chain is discussed as follows:

4.1 CDL Model (XML) to CDL Model (XMI)

The CDL XML model to CDL XMI model transformation, which is indicated as T1 in Figure 4, is used to transform a CDL model (XML format) to the CDL

[1] `http://www.eclipse.org/modeling/emf/`
[2] `http://www.eclipse.org/atl/`

model (XMI format) that conforms to CDL metamodel. This XMI model can then be read by the ATL engine to run the transformation specification. We use the eXtensible Stylesheet Language Transformation (XSLT) to implement this transformation, which takes a CDL model in XML and converts it to a CDL model in XMI.

4.2 CDL Model (XMI) to BPEL Model (XMI)

The CDL XMI model to BPEL XMI model transformation, which is indicated as T2 in Figure 4, is the core of our approach and is performed by executing the ATL transformation specification. The ATL engine reads a CDL XMI model as input, executes the transformation rules, and generates a BPEL XMI model as defined in the ATL transformation specification.

4.3 BPEL Model (XMI) to BPEL Process (XML)

After transformation T2 is performed, a BPEL XMI model that conforms to the BPEL metamodel is generated. The BPEL XMI model cannot be executed by orchestration engines, so we perform another transformation, indicated as T3 in Figure 4, which transforms a BPEL XMI model to a BPEL process (in XML). We use AtlanMod MegaModel Management (AM3)[3] for T3 transformation. This transformation performs an XMI-to-XML conversion of a BPEL specification by using the ATL's XML Extraction process [18]. An XML extraction is defined as a transformation from model-driven technical space to another technical space using an XML extractor, which is a tool that implements the extraction process.

Figure 4 shows the transformation chain of our approach which consists of 4 stages.

1. We use the Pi4soa[4] CDL editor to model the choreography specification, which is stored in this tool in the CDL XML format.
2. We use XSLT transformation to transform CDL XML specification to the XMI format that conforms to the CDL metamodel.
3. We execute the transformation specification as transformation rule in ATL engine that generates a BPEL model. The generated BPEL model is in an XMI serialized format that conforms to the BPEL metamodel.
4. We use ATL to transform from the BPEL XMI model to the BPEL XML model. This ATL transformation takes the BPEL metamodel as source metamodel and the XML metamodel as the target metamodel. We use ATL's XML Extraction process to extract the BPEL proces (in XML) from the BPEL XMI model [18]. In this way, a BPEL process in XML format is obtained as output that can be executed by orchestration engines.

[3] `http://wiki.eclipse.org/AM3`
[4] `http://sourceforge.net/apps/trac/pi4soa/wiki`

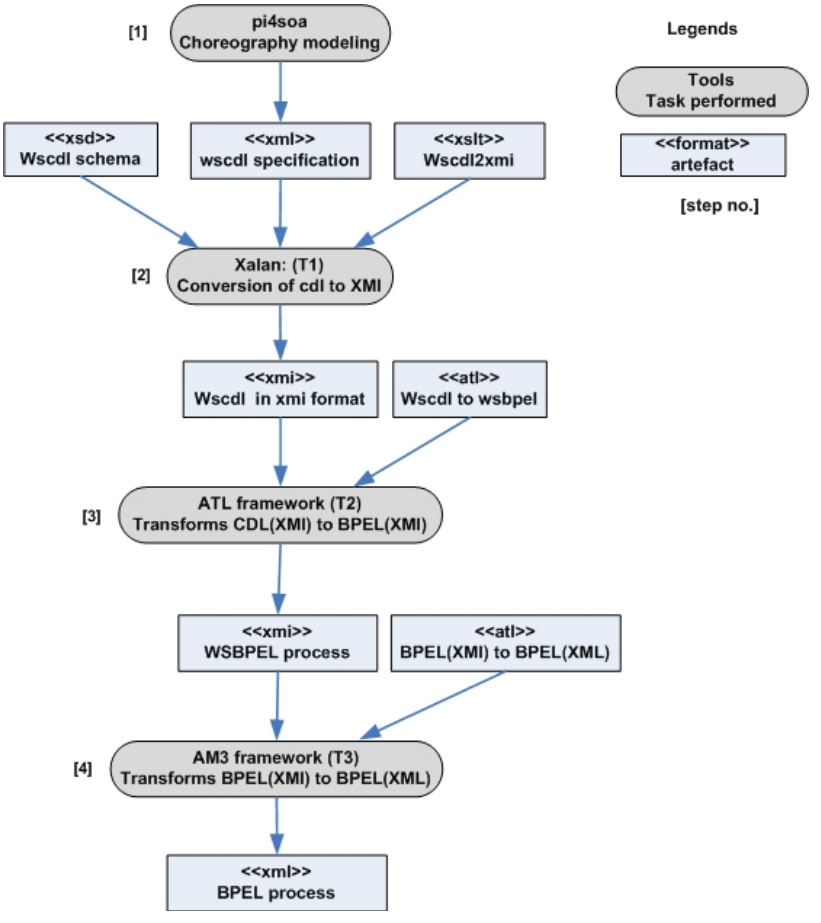

Fig. 4. Transformation Chain

5 Validation and Discussion

We implemented the example described in Section 3 to determine the feasibility of our model-driven transformation approach. We have modeled the choreography of the BTO example by using the Pi4soa choreography editor. The choreography has a higher abstraction level than the orchestration and choreography does not represent the internal details of the participating services in the collaboration, so we could not generate a complete executable BPEL process with necessary internal details. However, our approach successfully generated the BPEL skeleton for the Manufacturer from the given CDL specification. In our approach, the BPEL designer has to manually provide the missing details, like branching conditions, wherever necessary. This is also shown in the transformation mappings given in Table 1, in which we indicated that the BPEL

designer has to specify conditions manually. For instance, Listing 1 presents a code snippet of the generated BPEL skeleton in which the BPEL designer has to manually add a condition to replace the empty condition *default=0*.

Listing 1. Conditional branching

```
<flow name ="parallel">
 <if name ="Choice_CPUnotInStock">
   <condition >"default=0"</condition>
       <sequence>
          <invoke operation ="orderCPU" ../>
       </sequence>
```

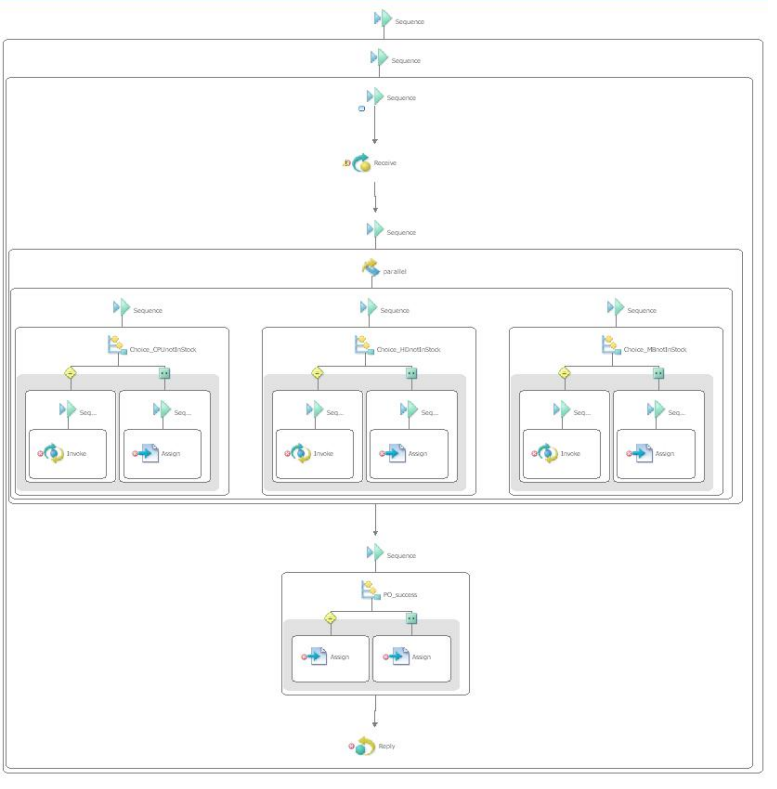

Fig. 5. BPEL process of Manufacturer

After the necessary conditions were added, we validated the BPEL process against the executable schema of BPEL to ensure syntactical correctness. We later imported the BPEL process in the ActiveBPEL designer to check the behavior of the orchestrator, and tested this behavior for correctness, with successful results.

Figure 5 shows the BPEL process of Manufacturer of BTO example in the ActiveBPEL designer.

The proposed approach is also validated for Purchase Order scenario [10] and tested with successful results.

6 Related Work

In this section we discuss existing Model-Driven SOA based approaches for enterprise interoperability as well as choreography to orchestration transformation approaches.

6.1 Model-Driven SOA Based Approaches for Enterprise Interoperability

Recently, model-based initiatives for enterprise interoperability have achieved significant attention in academic research and various interoperability frameworks have been proposed. The IDEAS Interoperability Framework [6] focuses on structuring the interoperability into business, knowledge, semantics, and architecture and platform issues. The ATHENA Interoperability Framework (AIF) [3] is a Model-Driven Interoperability (MDI) framework that has evolved from IDF. The AIF framework focus on the solution approaches for enterprise interoperability and defines different level of interoperations: enterprise/business, process, service, and information/data. The service level interoperability of the AIF framework is concerned with identifying, composing, and executing various services. The Platform-Independent Model for Service-Oriented Architecture (PIM4SOA) metamodel supports the execution and composition of services. PIM4SOA tool facilitates the transformation of PIM4SOA model (e.g., generated from higher level tooling) to a BPEL model. The AIF framework prescribes model-driven interoperability [3] and explicitly provides solution approach (PIM4SOA) for services. The PIM4SOA Eclipse plugin can be used for identifying, composing, and executing the services. However, compared to our approach PIM4SOA lacks the (semi-)automated transformation from choreography to orchestration. Though both the approaches are model-driven, our approach aims to achieve automation in enterprise collaboration and enterprise interoperablity.

6.2 Choreography to Orchestration Transformation

The work of [14] presents the mapping rules for the derivation of a BPEL process from a CDL specification. These mapping rules inspired the mappings shown in Table 1. In [14], the mappings are implemented in a recursive XSLT script, as a proof-of-concept, to realize the transformation. The transformation is bi-directional, i.e., the XSLT script can generate a BPEL process from a CDL specification, and vice-versa.

In [19], Rosenberg et al. proposed a top-down modeling approach to generate a BPEL process from a given CDL specification. This transformation also considers Service Level Agreements (SLAs), which are defined as annotations to the CDL specification, and are transformed into policies that can be enforced by a BPEL engine during execution. This transformation approach is implemented in Java 1.5 using a simple Swing-based graphical user interface.

Weber et al. [23] present a CDL to BPEL transformation in the context of a virtual organization, which is possibly also applicable to other domains. This transformation introduces the concept of information gap, which is the term used to mention the different levels of detail between a choreography and an orchestration, and indicates that the sum of orchestrations contains more knowledge than the choreography they implement. Given a choreography, their approach generates executable processes and respective WSDL specifications for each role. This transformation is implemented in Java.

In our approach, we use metamodel-based transformations, in which we define the abstract syntax (metamodel) of both the source and target models, and use ATL as transformation language. We follow MDA principles to perform the core transformation (T2) in the transformation chain, which allows us to express the mappings from source model to target model at a higher abstraction level than in case a programming language is used. Our approach falls under the "Transformation Language Support" category as described in [20]. In transformation language support, the transformation uses a language that provides a set of constructs for explicitly expressing, composing, and applying transformation. Our approach uses metamodels to express the mapping from source model to target model at a higher level of abstraction when compared with the other approaches. XSLT-based transformations require experience and considerable effort to define and maintain the transformation, whereas transformations written using general-purpose programming language tend to be hard to write, comprehend, and maintain the transformation. Hence, our metamodel-based transformation approach is favorable and is more efficient than other available techniques (i.e., Direct Model Manipulation, Intermediate Representation) [20,21], used in aforementioned related works.

7 Conclusion

In this paper, we presented a model-driven approach to service composition, based on combining concepts from Service-Oriented Architecture (SOA) and Model-Driven Architecture (MDA). The approach aims at facilitating enterprise collaboration by addressing enterprise interoperability problems caused by system heterogeneity and business-technology misalignment in the context of continuous change. The heart of the proposed model-driven service composition is the definition of a (semi-)automatic transformation from service choreography to service orchestrations. The approach has been illustrated with an implementation prototype and validated with a working example. We use service choreography to define service interoperability constraints in terms of message exchanges among collaborating enterprises. In our approach such constraints are preserved

in the transformation to service orchestration level. We successfully validated the approach using a Build-To-Order (BTO) application scenario as example, where we represented the generated service orchestration as a BPEL process and tested the behavior of the BPEL process for correctness. We achieved the two objectives of our research, namely supporting enterprise interoperability by service composition at choreography and orchestration level and guaranteeing consistency between these levels by defining an (semi-)automated transformation from choreography to orchestrations. Our transformation represents an improvement over the currently existing manual transformations. In particular, our transformation automatically preserves the interoperability constraints when going from choreography to orchestration level.

As future work we can still improve our approach in several ways. For instance, automatic generation of WSDL specification for each participant from the choreography description can ease the deployment and testing of the behavior of the generated BPEL process. We can further contribute to enterprise interoperability by including SLAs in the choreography description and transforming them to BPEL policies [19]. Additionally, transformation T1 of our approach, which is currently an XSLT-based transformation (see Figure 4), can also be performed using ATL by XML injection [18] so our approach is solely based on metamodel transformations.

References

1. Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guizar, A., Kartha, N., Liu, K.C., Khalaf, R., Konig, D., Marin, M., Mehta, V., Thatte, S., Van der Rijn, D., Yendluri, P., Yiu, A.: Web Services Business Process Execution Language Version 2.0. OASIS, pp. 1–126 (2007), `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`
2. Barros, A., Dumas, M., Oaks, P.: Standards for web service choreography and orchestration: Status and perspectives. In: Bussler, C.J., Haller, A. (eds.) BPM 2005. LNCS, vol. 3812, pp. 61–74. Springer, Heidelberg (2006)
3. Berre, A., Elvesæter, B., Figay, N., Guglielmina, C., Johnsen, S., Karlsen, D., Knothe, T., Lippe, S.: The ATHENA Interoperability Framework. In: 3rd International Conference on Interoperability for Enterprise Software and Applications (I-ESA 2007), pp. 771–782. Springer, Madeira (2007) ISBN 9781846288579
4. Binder, W., Constantinescu, I., Faltings, B.: Decentralized orchestration of composite web services. In: Proccedings of the International Conference on Web Services, ICWS 2006, pp. 869–876. IEEE Computer Society, Los Alamitos (2006)
5. Chafle, G., Chandra, S., Mann, V., Nanda, M.: Decentralized orchestration of composite web services. In: Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters, pp. 134–143. ACM, New York (2004)
6. IDEAS: A Gap Analysis Required Activities in Research, Technology and Standardisation to close the RTS Gap - Roadmaps and Recommendations on RTS activities, IDEAS, Deliverable D.3.4, D 3.5, D 3.6 (2001)
7. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Science of Computer Programming 72(1-2), 31–39 (2008)

8. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., Valduriez, P.: ATL: a QVT-like transformation language. In: Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications, pp. 719–720. ACM, New York (2006)
9. Kavantzas, N., Burdett, D., Ritzinger, G., Lafon, Y.: Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation, World Wide Web Consortium (November 2005)
10. Khadka, R.: Model-Driven Development of Service Compositions: Transformation from Service Choreography to Service Orchestrations. Master's thesis, University of Twente (August 2010), http://essay.utwente.nl/59677/
11. Khadka, R., Sapkota, B.: An Evaluation of Dynamic Web Service Composition Approaches. In: Proceeding of the 4th International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing, ACT4SOC 2010, pp. 67–79. INSTICC Press, Athens (2010)
12. Kopp, O., Leymann, F.: Choreography Design Using WS-BPEL. Data Engineering 31(2), 31–34 (2008)
13. Li, M., Cabral, R., Doumeingts, G., Popplewell, K.: Enterprise interoperability research roadmap. An Enterprise Interoperability community document (2006)
14. Mendling, J., Hafner, M.: From inter-organizational workflows to process execution: Generating BPEL from WS-CDL. In: Chung, S., Herrero, P. (eds.) OTM-WS 2005. LNCS, vol. 3762, pp. 506–515. Springer, Heidelberg (2005)
15. Miller, J., Mukerji, J., et al.: MDA Guide Version 1.0. 1. Object Management Group (2003), http://www.omg.org/cgi-bin/doc?omg/03-06-01
16. Papazoglou, M.: Web services: principles and technology. Addison-Wesley, Reading (2008)
17. Peltz, C.: Web services orchestration and choreography. Computer 36(10), 46–52 (2003)
18. Ribarić, M., Gašević, D., Milanović, M., Giurca, A., Lukichev, S., Wagner, G.: Model-Driven engineering of rules for web services. Generative and Transformational Techniques in Software Engineering II, pp. 377–395 (2008)
19. Rosenberg, F., Enzi, C., Michlmayr, A., Platzer, C., Dustdar, S.: Integrating quality of service aspects in top-down business process development using WS-CDL and WS-BPEL. In: Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2007, p. 15. IEEE Computer Society, Los Alamitos (2007)
20. Sendall, S., Kozaczynski, W.: Model transformation: The heart and soul of model-driven software development. IEEE Software 20(5), 42–45 (2003)
21. Sendall, S., Kozaczynski, W.: Model Transformation the Heart and Soul of Model-Driven Software Development. Tech. rep. (2003)
22. van Sinderen, M.: Challenges and solutions in enterprise computing. Enterprise Information Systems 2(4), 341–346 (2008)
23. Weber, I., Haller, J., Mulle, J.: Automated derivation of executable business processes from choreographies in virtual organisations. International Journal of Business Process Integration and Management 3(2), 85–95 (2008)