Enabling the Evolution of Service-Oriented Solutions Using an UML2 Profile and a Reference Petri Nets Execution Platform*

Javier Fabra[1] Joaquín Peña[2] Antonio Ruiz-Cortés[2] Joaquín Ezpeleta[1]

[1]Department of Computer Science and Systems Engineering, University of Zaragoza, María de Luna 1, E-50018 Zaragoza (Spain)

[2]Department of Languages and Computer Systems, University of Seville, Av. Reina Mercedes s/n, E-41012 Seville (Spain)

E-mail: jfabra@unizar.es,joaquinp@us.es,aruiz@us.es,ezpeleta@unizar.es

Abstract

The activities developed by a company (business processes) have to change frequently to adapt to the environment. The implementation of business processes should support these changes without any recoding. In this work, we provide with an approach for modelling and executing agile and adaptable business processes. Our approach is based on UML 2 separating choreography (stable interaction patterns) and orchestration (implementation of the evolving business process, also called workflows), allowing the transformation and execution of the models by means of a flexible SOA-based dynamic platform based on reference Petri nets.

Keywords: SOC, SOA, Web processes, MDD, UML, Petri nets, Choreography, Orchestration.

1. Introduction

One of the primary goals of SOC and SOA is easing the integration of businesses by means of IT infrastructures. As shown by Lee in the field of Economy [11], business processes followed by businesses that interact in a supply chain must change over time. Lee emphasises that in order to succeed in the business integration, processes needs to be agile, capable to rapidly change to respond to unexpected situations, and adaptable, *id est*, able to change considering new trends of markets. For instance, agility can be observed in the changes needed in the business process of a dealer and a logistic platform when a flow or earthquake happens. In normal situations, the dealer executes a business process to track packages in order to ensure a given level of service quality (QoS). But when an earthquake or a flow happens, the business process has to change to track such packages with a cost higher than a given threshold, just in order to minimise losses. In this scenario, the changes in the business process should not be overcome by difficulties in the changes required in the IT infrastructure.

For enabling this flexibility, the business process and the interactions between IT infrastructures must be separated. This separation allows to change business processes without having to perform time and money costly changes in the IT infrastructure [14]. Two terms are used to refer to each part of this decomposition. On the one hand, the term *orchestration* describes the business process followed by each party without detailing the interaction logics. On the other hand, the term *choreography* describes the interactions which take place between IT infrastructures of several parties without detailing the internals of business processes. Therefore, changes in the business process do not have to force changes in the way IT infrastructures interact, and viceversa.

In this paper, we have systematically analysed current results for performing this separation in SOC and SOA. As a result of this study, we have concluded that the support for this separation is deficient at design/modelling stages and implementation/execution platforms. Thus, this fact hinders one of the key success factors in business integration: the need for agile and adaptable business processes [11, 13].

Fortunately, there exist a branch of research devoted to overcome this problem, being Model-Driven Development (MDD) one of the used approaches. MDD emphasises the use of models as the main tool for designing and implementing systems. Thus, MDD approaches provide models in order to specify the system and transformations that help to obtain, as much automatically as possible, a running system.

In this paper, we propose a set of UML 2 models for separately modelling orchestration and choreography aspects

*This work has been partially supported by the European Commission (FEDER) and the Spanish Ministry of Science and Technology under grants TIN2006-00472 and TIN2006-13301

allowing the evolution of a business process independently from the interaction logic. These models are based on an agent-oriented software engineering methodology (AOSE) called MaCMAS [16]. We also provide an implementation and execution platform based on reference Petri-nets which also keeps this separation, allowing changes at runtime [8]. The main advantage of this approach is that it allows changes in the business process in a well-known notation such as UML and a rapid deployment of such changes in a SOA-based platform, enabling the modelling and execution of adaptable and agile business process.

The paper is organised as follows. Section 2 presents the related work regarding modelling, implementation and transformation between both UML 2 models and choreography and orchestration implementations. Section 3 introduces the UML 2 notation used for modelling orchestration and choreography through a case study. Keeping this separation, the implementation and execution platform used, DENEB, is shown in Section 4. Section 5 presents an overview of the keypoints of the transformation between the models and the implementation. Finally, Section 6 contains some concluding remarks.

2. Related Work and Motivation

According to [2], an analysis framework with a set of components has to be defined in order to perform a survey in the software engineering field. Given the limitation of space, summarising, the main research questions of this paper are: (i) Are current MDD approaches able to separate orchestration and choreography? and, (ii) Is this separation performed in execution frameworks? In Table 1, we classify the approaches in those developed for modelling and those developed for implementation. In the table, $-$ means that the study does not deal with the row item; X means that the study considers the row item but it does not give any approach; $\sim$ means that the study gives a brief overview and, finally, $\surd$ means that the study gives a valid approach for the item.

Regarding MDD approaches, as shown in Table 1, there are approaches based on UML which separate choreography and orchestration [23, 21]. In [23] the impact of SOA implementation projects is analysed in conjunction with a modelling approach. However, as the authors state, the notation proposed is immature and they do not provide a technique to deploy models into an implementation platform. In addition, there are other approaches which do not perform the separation, but where business logics and interaction aspects are identified [9, 12, 4]. In [9], authors propose an UML profile and an MDD transformation. However, they neither separate orchestration or choreography in models nor generate code for these aspects. Similar mappings are followed in [21, 12]. However, in these approaches a preliminary version of notation and transformations is provided where a complete separation between orchestration and choreography is not achieved.

With respect to implementation details, there are many proposals which fit into the separation between choreography and orchestration aspects. In [4] and [19] authors propose and define a way to generate implementations separating choreographies and orchestrations. Unfortunately, these implementations hard-code orchestration with choreographies, thus disabling run-time evolution of business processes. The general rule is the transformation of models into orchestration languages, such as BPEL [20], WSCI [1] or BPML [22].

Regarding current standards, BPEL uses WSDL interfaces to describe the functionality it offers and also to invoke functionalities required from other Web services [18, 6]. As a consequence, the management of interactions provided by BPEL is based on one-shot interactions instead of a long-lived conversational approach, causing business and conversation logics to be highly coupled [8, 23]. Furthermore, the pre-defined and inflexible nature of BPEL does not cater for flexible and adaptive business collaborations. Nevertheless, a natural evolution of the BPEL4WS specification should replace current WSDL-based abstractions with new conversation models such as WCSI [1], WSCDLand OWL-S. The main problem of these initiatives is that they have a declarative nature and cannot, by itself, be executed, limiting dynamic aspects.

Thus, the main motivation of this work is that, by the best of our knowledge, there is not a suitable approach in the SOC field following MDD techniques, nor any execution and implementation platform that performs a correct separation between choreography and orchestration aspects.

3. An UML 2 Profile for Modelling Choreography and Orchestration

Agent-Oriented Software Engineering (AOSE) is devoted to develop highly collaborative systems where aspects similar to the ones in the SOC field have been dealt with. Models presented in this section are based on an AOSE methodology called *methodology for analysing complex multiagent systems* (MaCMAS) that is being integrated in several research fields such as autonomic computing or software product lines [16, 17].

MaCMAS uses two different diagrams for representing each view of the process: *a dynamic diagram* representing how interactions and business processes take place along time based on UML State Machines; and a *static view* showing the relationships between services in a static way by means of extended UML 2 collaborations. Orchestration is represented as an abstract description instantiated at runtime or design time, with concrete services and a concrete

Prop/Research Questions	[9]	[23]	[21]	[12]	[4]	[19]	[20]	[1]	[22]	Our App.
Models for Chor.& Orch.	$\sim$	$\surd$	$\surd$	$\sim$	$\sim$	X	$-$	$-$	$-$	$\surd$
Transf. UML2Impl.	$\surd$	X	$\sim$	$\sim$	X	X	$-$	$-$	$-$	$\sim$
Impl. separating Chor.& Orch.	$\sim$	X	X	X	$\surd$	$\surd$	$\sim$	$\sim$	$\sim$	$\surd$

Table 1. Related work study.

Metamodel element	Stereotype	Uml Base Class	Tags
Role	$<< Role >>$	CollaborationRole	multiplicity
EnvironmentalRole	$<< EnvironmentalRole >>$	CollaborationRole	–
mRI	$<< mRI >>$	Collaboration	pattern:String
mRI Postcondition	–	Postcondition	–
mRI In	–	Collaboration	Out: String
mRI Out	–	Collaboration	In: String
Role State	$<< RoleState >>$	State	–
mRI Transition	$<< RoleMRITransition >>$	Signal in Trans.	–

Table 2. UML 2 profile for Choreography and Orchestration

choreography. Figure 1 depicts the role model of our case study, a business collaboration. The *Dealer* sells products which are sent to customers by the *Logistic* platform, allowing the tracking during the delivery process. To statically show the relationships between services we propose to use extended MaCMAS role models. These models are built of a set of *roles*, the portion(s) of a software artifact(s) which is exposed as a service, collaborating by means of several *multi-Role Interactions* (mRIs). In Figure 1 boxes represent roles, and ellipses interactions; the top view represents the orchestration (business process), while the bottom view represents the choreography. mRIs in the external package represent cross-organisation interactions, while the other mRIs represent internal uses of services.

mRIs are used at the orchestration level in order to describe the business process without detailing the set of needed messages. They are also used to represent concrete interactions at the choreography level. Thus, mRIs at the orchestration level also permit linking with an specific interaction protocol at design time, or relegate this linking at run-time. For example, in Figure 1, in the top model the mRI *TrackOrder* abstractly represents the set of messages needed to track an order, while in the bottom model mRIs *getOrders* and *trackOrder* represent the concrete messages that have to be exchanged. The tagged value *Pattern* represents a collaboration pattern, for example a request-reply pattern which can be used to instantiate at run-time well-known MEPs (*Message Exchange Patterns*) from a repository. An arrow from a role to an mRI means that the role initiates the mRI, while an arrow presenting an association responds to the initiator.

Table 2 presents the profiles for *roles* and *mRI*s. As shown, the role element is attributed by means of UML tagged values. The *multiplicity* tag specifies the number of concrete services that can play this role. The *Environmental Role* is an stereotype that indicates that the role is not a service, but a software component. The interactions between these roles and services represent communication acts with the legacy system. Regarding *mRI*s, *Postcondition* represents the condition that must hold after the execution of an mRI. *In* and *Out* elements represent the information consumed and produced by an *mRI*.

In addition, the dynamic part, involving time, is represented by means of UML 2.0 **State Machines**. In Figure 1, the dynamic view of the case study is shown on the right and bottom of the figure. The right model represents the orchestration between the *Dealer* and the *Logistic* platform, while the bottom model represents the choreography for tracking orders. In both models each transition represent the execution of a mRI.

Finally, notice that the *MaCMAS CASE Tool* allows to develop these models as an extension of the ArgoUML Case tool. Therefore, the presented models can be drawn by means of the presented profile. ArgoUML allows exporting models in XMI format (a standardised XML file with the description of the UML models), which is taken as input of our transformation (see Section 5).

4. Implementation and Execution Platform for Separating Orchestration and Choreography

DENEB (platform for the Development and Execution of iNteroperable dynamic wEB processes) is a dynamic Web processes implementation and execution framework based on the Nets-within-Nets paradigm and the Renew

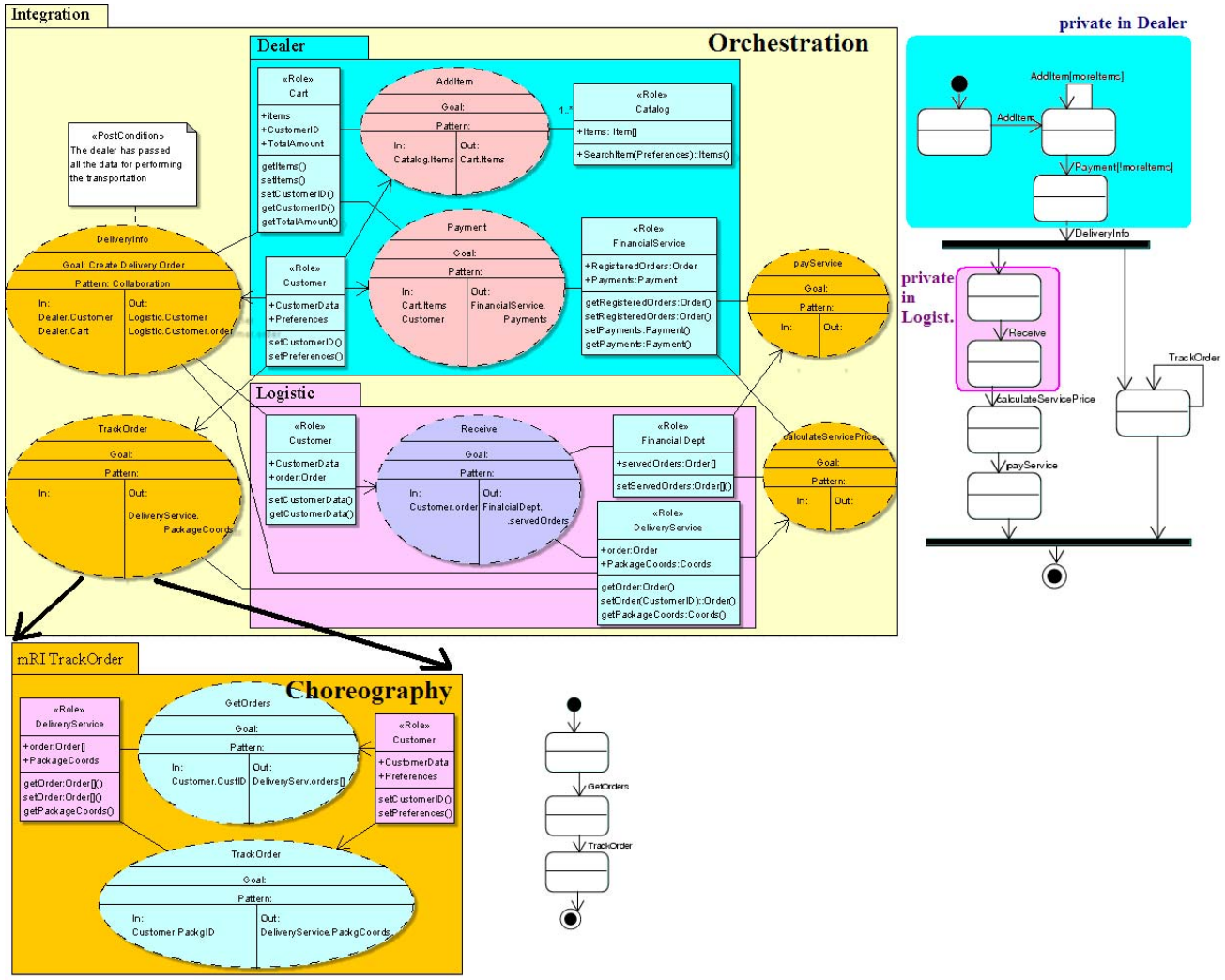

Figure 1. Role model of our business collaboration case study.

tool [8]. This framework allows the correct separation between choreography and orchestration aspects and manages all the communication-related aspects through a message broker based on RLinda [7], an implementation of the Linda coordination system [5]. The SOA-based architecture of DENEB is composed of three main components, depicted in Figure 2.

The message broker or *enterprise server bus* is the core of DENEB. It provides the necessary infrastructure to decouple and support interactions among different components and separates the logic of the message exchanges from the concrete way a message is delivered or received. It contains two main components: a message repository; and the *binding components* (also called *mediators*), which are responsible of sending and receiving messages among remote or local peer components through the message repository [21]. These components allow processes to use different technologies and communication protocols independently of the way they interact with the message repository.

As shown in Figure 2, the software components which act as service engines are articulated around the message broker. They can take a wide variety of forms depending on the type of function they supply, such as business logic or transformation services, for instance. SOC avoids any knowledge of the programming model of the service engine that plugs into the broker infrastructure, so the service components interact with external entities by means of the exchange of messages through the message broker using a common and defined format. The binding components are responsible of sending and receiving messages to and from the concrete endpoint. The different functionalities that exists in the service engines allow the separation in two classes

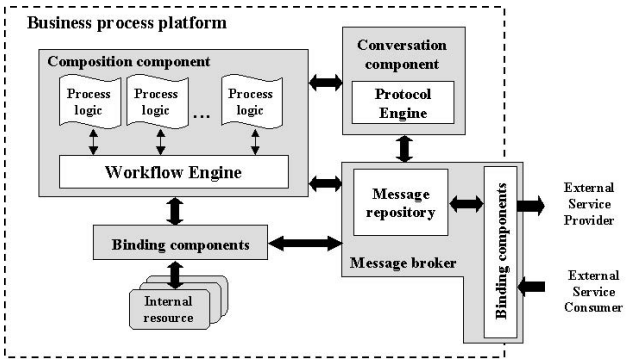

Figure 2. High-level view of the architecture of DENEB.

of service engines. On the one hand, the *workflows engines*, which represent the business logics of the process and are in charge of executing the orchestration processes. On the other hand, the *protocol engines*, which represent the interaction logics and executes the choreographies. The use of both, workflow and protocol engines, allows the correct and independent separation of choreography and orchestration aspects, whose models are described and directly executed in terms of Nets-within-Nets in DENEB [10, 3]. In addition, all models are exportable using an extended PNML description language (ISO/IEC 15909), which can be generated in an automatic manner (see Section 5).

Figure 3 depicts the automatically generated orchestration workflow corresponding to the *DeliveryService* role depicted in Figure 1 and the *trackOrder* choreography, which will be executed by the orchestration process in response to the initial customer's invocation (according to Figure 1). For the sake of simplicity, only these two figures are shown to demonstrate the transformation. Note that orchestration implementations are executed by the workflow engine, whereas choreographies are created by orchestrations and run by the conversation engine in the DENEB framework. Both orchestration and choreography implementations are executed on an instance of the DENEB platform.

DENEB provides with some mechanisms in order to allow workflows to communicate with conversations, and viceversa. *Channels* are a mechanism which allow two or more nets to synchronise by means of the fire of a transition which contains a channel inscription [10]. Additionaly, channels allow workflows and conversations to exchange information. Those are the basics of the `:absCond()` channels used in the workflow and conversation nets. Finally, conversations can interact with the message broker by means of the use of channels `:w()` and `:t()`, which correspond to the write and read operations of the RLinda implementation, respectively.

Let us now describe the orchestration workflow which corresponds to the *delivery service*. Transitions `t1` and `t20` implement the starting and the ending of the execution of the orchestration, respectively. According to the state machine depicted in Figure 1, the firing of transition `t1` starts the parallel execution of two branches in the case the order has not still been received (guard *!received*). The left one corresponds to the sequential creation and execution of the choreographies *deliveryInfo*,*Receive* and *CalculateServicePrice*, while the right one represents the creation and execution of the *trackOrder* choreography. Let us concentrate on the *trackOrder* branch. Transition `t11` creates a new instance of the *trackOrder* choreography, putting it into the conversation space of the DENEB framework by means of the synchronised firing of channel `:participateConv`, in Renew's terminology [10]. This step implements the participation of the orchestration process in a choreography initiated by another process (by the customer, in this case). Then, firing transition `t12` the conversation engine starts the execution of the choreography in a distributed manner. Transitions `t13` and `t16` allow the orchestration workflow to receive and pass data from and to the running choreography through *abstract conditions* (channel `:absCond`). These data are processed by the binding components which manage the access to internal or external resources in a proper way. The mechanism to interact with the system binding components using an `:execute` channel is implemented in transitions `t14` and `t15`. The choreography's execution finishes firing transition `t17`, and a token is placed in the top place of the right branch. The process can repeat until the order is received. In that case, the guard of `t11` disables the firing of transition `t11` and the only possible evolution is to fire transition `t18`. Finally, when the *CalculateServicePrice* choreography finishes, both branches synchronise and the orchestration process ends.

Let us now concentrate on the *trackOrder* choreography. Once loaded into the conversation engine, the *trackOrder* choreography starts by means of the synchronisation of transition `t30` through the DENEB's system net. Transition `t31` processes the invocation of a customer request for the availability of track orders through the message broker using the Linda-based operation *take* [5]. Then, the request is passed to the orchestration workflow and the response of its execution is obtained by means of the use of abstract conditions (transitions `t32` and `t33`, respectively). Once the result has been obtained, a response is sent to the customer trough the message broker using the `write` operation, firing transition `t34`. A similar interaction process is repeated again (transitions `t35-t38`) to process the tracking of a specific order selected by the customer. Finally, the *trackOrder* choreography finishes the execution of the protocol by means of the firing of transition `t39`.

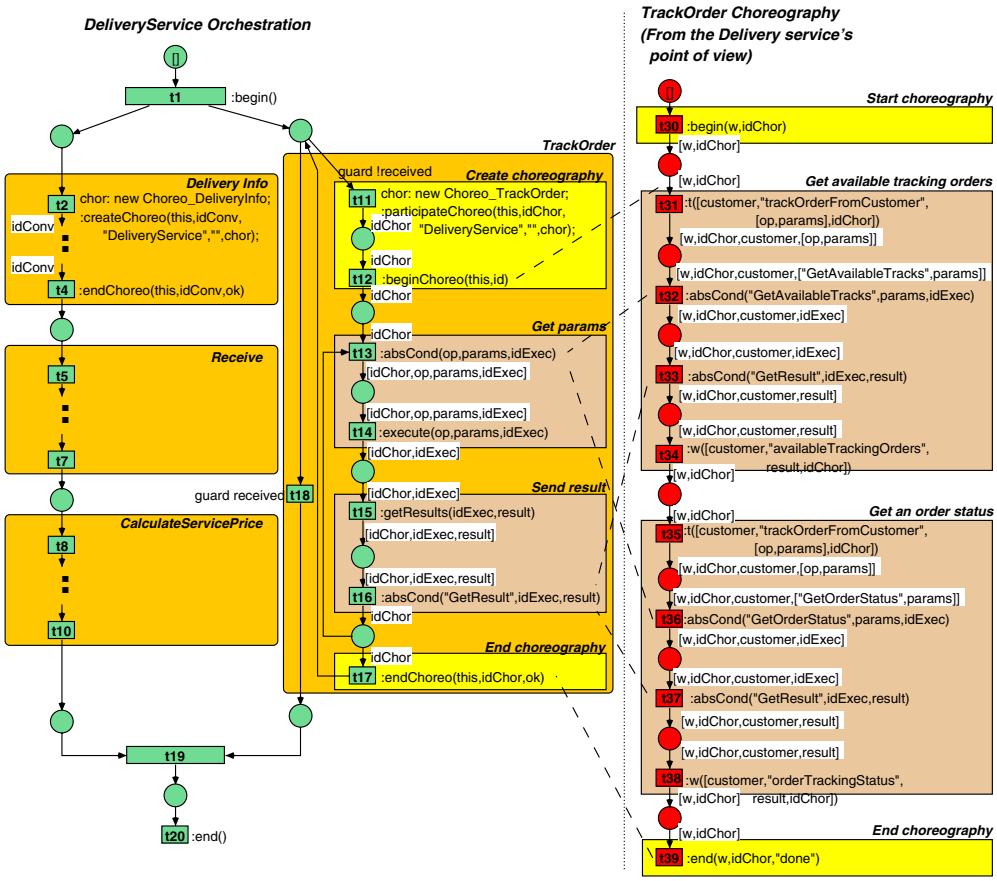

Figure 3. Delivery Service orchestration workflow and trackOrder choreography.

5. Overview of the Transformation

Although the transformation process is not the aim of this paper, the main correlations between the concepts at the modelling and the implementation levels are provided. The transformation for regular Petri-nets has been performed in the literature, but the transformation from the modifications of UML to reference Petri-net and its direct execution have not been provided yet.

Implementing abstract *mRIs* into DENEB workflows implies, as was shown, the execution of a choreography process initiated from an orchestration process (both, the corresponding skeleton in the orchestration part and the choreography, are automatically generated). Depending whether the role is the interaction initiator or not (this information is extracted from the role model) the workflow *creates* the choreography or *participates* in it by means of the use of `:createChoreo()` or `:participateChoreo()` synchronisation channels, respectively. These channels synchronise through the DENEB's system net. Therefore, orchestration workflows and choreographies can execute in an independent manner. The parameters in mRIs (*input* and *output*) are implemented in DENEB through the `:begin` and `:end` synchronisation channels which allow to pass information from the orchestration workflow to the choreography and viceversa, respectively.

Further details about the correlation and transformation process can be obtained in [15]. From the point of view of implementation, transformations among MacMAS/UML models and DENEB's entities are performed using XSLT code in order to transform the XMI of the UML model into extended Petri-Nets Modelling Language (PNML), which is also based on XML. The automatically generated PNML code represents the DENEB's implementation nets, which can be loaded and executed in the platform at run-time.

6. Conclusions

Successful development and execution of flexible and adaptable SOA systems need from best practices to solve some important drawbacks that appear when dealing with

choreography and orchestration aspects. Although they are two sides of the same coin, it is commonly accepted that both of them need to be dealt in a separated, but coordinated way along the full development and execution life cycle. In this paper, we have shown how the integration of a MDD approach, MaCMAS, with a flexible development and execution framework for Web processes, DENEB, exploits this separation at the design and execution stages. The integrated approach allows a very flexible way of dealing with business processes, facilitating their evolution in highly dynamic scenarios. From this study, new challenges have been opened, as for example trying to perform direct transformations from models to implementation using well known description standards (such as the Business Process Modeling Notation – BPMN, for example) and rule-based transformation processes (such as the ATLAS Transformation Language – ATL, for example).

References

[1] A. Arkin *et al*. Web Service Choreography Interface (WSCI). Technical report, World Wide Web Consortium (W3C), Aug. 2002.

[2] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software*, 80(4):571–583, 2007.

[3] L. Cabac, M. Duvigneau, D. Moldt, and H. Rölke. Modeling dynamic architectures using nets-within-nets. In *26th International Conference on Application and Theory of Petri Nets – ICATPN 2005*, pages 148–167, 2005.

[4] T. Cottenier, A. V. D. Berg, and T. Elrad. Modeling Aspect-Oriented Compositions. In *7th International Workshop on Aspect-Oriented Modeling*, Oct 2005.

[5] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–121, 1985.

[6] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana. The next step in web services. *Communications of the ACM*, 46(10):29–34, 2003.

[7] J. Fabra, P. Alvarez, J. A. Bañares, and J. Ezpeleta. RLinda: a Petri net based implementation of the Linda coordination paradigm for Web services interactions. In *7th International Conference on Electronic Commerce and Web Technologies – EC-Web 2006*, number 4082 in Lecture Notes in Computer Science, pages 184–193. Springer Verlag, Sept 2006.

[8] J. Fabra, P. Álvarez, J. A. Bañares, and J. Ezpeleta. A framework for the development and execution of horizontal protocols in open BPM systems. In *Fourth International Conference on Business Process Management – BPM'06*, number 4102 in Lecture Notes in Computer Science, pages 209–224. Springer Verlag, 2006.

[9] S. K. Johnson and A. W. Brown. A model-driven development approach to creating service-oriented solutions. In *4th International Conference on Service-Oriented Computing – ICSOC 2006*, pages 624–636, 2006.

[10] O. Kummer, F. Wienberg, M. Duvigneau, M. Köhler, D. Moldt, and H. Rölke. Renew – the Reference Net Workshop. In E. Veerbeek, editor, *Tool Demonstrations. 24th International Conference on Application and Theory of Petri Nets – ATPN 2003.*, pages 99–102, June 2003.

[11] H. Lee. The triple-a supply chain: Adaptability, agility, and alignment. *Harvard Business Review*, 82(10):102–112, 2004.

[12] J. Mendling and M. Hafner. From Inter-Organizational Workflows to Process Execution: Generating BPEL from WS-CDL. In *On The Move to Meaningful Internet Systems and Ubiquitous Computing – OTM 2005*, number 3762 in Lecture Notes in Computer Science. Springer Verlag, Nov 2005.

[13] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing Research Roadmap. Technical report, Technical report/vision paper on Service oriented computing European Union Information Society Technologies (IST), 2006.

[14] C. Peltz. Web Service Orchestration and Choreography. A look at WSCI and BPEL4WS. *Web Services Journal*, pages 1–5, jul 2003.

[15] J. Peña, J. Fabra, A. Ruiz-Cortés, and J. Ezpeleta. A Model-Driven Development Approach for Specifying and Implementing the Orchestration and Choreography of Service-Oriented Solutions. Technical report, reference RR-0707, Department of Computer Science and Systems Engineering – I3A University of Zaragoza, July 2007.

[16] J. Peña, M. G. Hinchey, M. Resinas, R. Sterritt, and J. L. Rash. Designing and managing evolving systems using a mas-product-line approach. *Journal of Science of Computer Programming*, 2006.

[17] J. Peña, M. G. Hinchey, and A. Ruiz-Corts. Multiagent system product lines: Challenges and benefits. *Communications of the ACM*, 49(12), December 2006.

[18] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. Ferguson. *Web services platform architecture*, chapter Modeling Business Processes: BPEL, pages 313–340. Prentice Hall, 2005.

[19] N. C. Suazo and J. Aguirre. Aspect-oriented Web services orchestration. In *2nd International Conference on Electrical and Electronics Engineering*, Sept 2005.

[20] T. Andrews *et al*. Business Process Execution Language for Web Services (BPEL4WS). Technical report, BEA Systems, IBM Corp., Microsoft Corp., SAP AG, and Siebel Systems, May 2003.

[21] R. Ten-Hove and P. Walker. Java Business Integration (JBI) 1.0, final release. Technical report, BEA Systems & IBM & Microsoft & SAP AG & Siebel Systems, May 2005.

[22] R. K. Thiagarajan, A. K. Srivastava, A. K. Pujari, and V. K. Bulusu. BPML: A Process Modeling Language for Dynamic Business Models. In *Fourth IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems – WECWIS'02*, page 239. IEEE Computer Society, 2002.

[23] O. Zimmermann, P. Krogdahl, and C. Gee. Elements of Service-Oriented Analysis and Design. Technical report, IBM developerWorks, June 2004.