Generation of Component Based Architecture from Business Processes: Model Driven Engineering for SOA

Karim Dahman, François Charoy, Claude Godart

Generation of Component Based Architecture from Business Processes: Model Driven Engineering for SOA

Karim Dahman, François Charoy, Claude Godart
Université de Lorraine, UHP - LORIA
BP 239 54506 Vandoeuvre-lès-Nancy Cedex, France
{*karim.dahman, francois.charoy, claude.godart*}*@loria.fr*

***Abstract*—Service-Oriented Architecture (SOA) has significantly enhanced inter-organizational systems enabling business flexibility, Information Technology (IT) agility, and value generation. However, building a SOA that reduces technology-driven business and leverages process management seems referring to the recurrent issues of business process logic and IT alignment. This paper presents a model-driven development approach where long-running business service composition models drive their supporting service implementation models. To progress on the successful route to a SOA engineering with minimal design decisions losses, we propose a model-to-model transformation that preserves the architectural alignment between the business process and their supporting service implementation infrastructure. The result of the transformation is a component configuration model based on a SOA. It promotes the separation of business concerns, enabling quick and localized evolutions of the IT infrastructure.**

***Keywords*-Service Engineering, BPM, MDD, SOA, BPMN, SCA, Business Process Logic and IT alignment**

I. Introduction

Service-Oriented Architecture (SOA) refers to an architectural paradigm and discipline that may be used for organizing and building distributed software systems utilizing and providing capabilities based on business electronic services being under the ownership of different domains [1]. It is said to enable inexpensive companies' Information Technology (IT) systems adaptation to changing business conditions [2]. Nevertheless, this flexibility regarding the readjustment of the affected IT systems makes sense only when considering the SOA as a part of the Business Process Management (BPM) [3].

As business processes define the order of work, they are directly related to the efficiency and effectiveness of the company. To avoid technology-driven business, the managed business processes must be designed at the level of business value exchange between the organizations. Accordingly, the business-driven systems based on a SOA solution must be considered independently from a specific supporting IT infrastructure [4]. However, when organizations must reflect evolutions in their existing business process models, the IT must also adapt to changing company goals. Certainly, the SOA enables flexible service-oriented systems, IT agility, and value generation, but obviously, it refers to the endless interweaving and alignment issues of the IT-level with the business-level logic. Hence, we need to combine the business process modeling space with the core SOA concepts.

In this paper, we present a Model-Driven Development (MDD) approach where long-running service-oriented business process specifications drive the supporting IT assets: starting from an abstracted view of the service interactions pertaining to a domain being modeled (i.e., in the design space), it is possible to generate a system architecture (i.e., in the runtime space) as a configuration of interacting components that are based on a SOA. Our first motivation is to automate the production of component configuration models that are aligned with the business process models. The provisioned configurations must provide an explicit architectural view of the business process structure that has led to this service implementation infrastructure. A design framework for this scenario requires a model-to-model transformation method. We propose to define a conceptual mapping rule specification and its implementation between subsets of two specific metamodel: the Business Process Modeling Notation (BPMN 2.0 [5]) and the Service Component Architecture (SCA 1.0 [6]). The BPMN is used by business analysts to design business process logic for software solution specifications in an SOA manner. In contrast, the SCA is used by system architects to logically modularize and compose business functions in assemblies of components. The business functions are encapsulated in software blocks that consume and offer business services. These composite application models can contain both new business services created specifically for the application and business function from existing system applications, reused as part of the composition. These models describe and guide the final system deployment architecture and its implementation as a solution for an organization-specific architecture or for an extended network of connected enterprises. However, the different usage spaces of the business processes and the system architecture causes a conceptual gap between the produced models. To overcome this de facto gap, we introduce a technique that improves the service-oriented semantic of process models before they are mapped into aligned component configuration models. Likewise, IT developers adapt these assets with less design decisions losses, and without diverging from the business goals.

This paper is organized as follows. In Section II, we situate our MDD approach through the related work. In Section III, this approach starts off from the modeling of software inter-organizational systems at a business-level perspective, and leads through a model transformation, in Section IV, to a supporting SOA infrastructure. The

specification of the conceptual mapping rules and their implementation, presented in Section V, bridge the business process design with how the service implementations can be assembled, composed, deployed and monitored. Finally, Section VI concludes this paper, and presents the future directions of our research.

II. Background and Related Research

Building inter-organizational systems in a SOA manner that leverages BPM best practices implies a range of technical and methodological issues including the design and the implementation of systems models. From an architectural perspective, the model of a service-oriented system (e.g., the SOA Reference Model [7]) is a specification, for some particular purpose, of distributed business capabilities in a domain of a partnership federation. From a process flow perspective, it relates to coordinate an explicit exchange of information through automated service interactions under the control of single endpoint, called service orchestration [1]. Generally, a long-running service orchestration comprises many interactions between a set of federated business services. It refers to an automated execution of a machine-readable description that decouples the service composition layer from its implementation. This model is obtained through a task-centric methodology in the design space. In the runtime space, the process tasks are executed by a supporting component configuration that contains effective artifacts realizing the business services.

Several frameworks have emerged to help organizations to charter successful SOA solution implementations. The Object Management Group (OMG) with its well known Model-Driven Architecture (MDA) approach, provides a mean for using model transformations to direct the course of system development stages [4]. Applications are designed independently of their supporting platform-induced constraints. They stand at a Platform Independent Model (PIM) view without the details of a particular platform type and its technical requirements. A model instance mapping approach, called conceptual mapping, can be used to identify elements in the technology-neutral system model which should be particularly transformed, given the choice of a Platform Specific Model (PSM) view. To automate the PIM to PSM transformation, it is necessary to determine a conceptual equivalence between the abstract elements of the used metamodels. In the SOA space, MDA enables business-level functionalities to be modeled with the Unified Modeling Language (UML) at a PIM level while being separated from their lower-level implementation [8]. Certainly, SOA models used within a MDD approach can be expressed using the UML [4], [9]. However, as UML is neither fully service-centered, nor business process-oriented [10], it is necessary to provide alternative methods to design the business process logic independently of their implementation infrastructure. Thus, model-driven design techniques needs to address the gap in existing UML modeling methodologies for the business service design and the used lower-level standards.

A top-down SOA methodology where a business process perspective leads to an IT execution perspective, is suggested in [2]. It advocates the integration of a business values perspective to the modeling process. Unfortunately, the approach is largely focused on the linkage between business operational view and functional business service view. It defines a methodology for the BPM design space, but fails to provide a practical SOA development scenario. In [3], the authors present a MDD scenario that uses the Eclipse SOA Tools Platform Intermediate Metamodel. The scenario starts off with a business process specified in the BPMN, leading to an intermediate model and resulting in a SCA assembly model [6]. The approach is promising. However, the absence of an accurate mapping between the two metamodels makes the model transformation style not viable. For example, every process task is transformed into a simple component representing a single business service. First, it requires further manual adaptation to reflect the initial process requirements. Of course, adapting the generated component configuration necessarily leads to their misalignment with the business process layer, and implies design decision losses. Second, this automated transformation is fictional, since source metamodel concepts are mapped to conceptually different target elements.

Unlike the previous approaches, we focus on the generation of a canonical component configuration model that supports a high-level business process logic. Adopting a MDD approach to build software solutions based on a SOA may require to use different technologies in the same system architecture, when implementing service infrastructure. In fact, the SCA assembly model [6] provides a framework for creating software solutions in a multi-language and multi-platform environment that are based on a SOA. Therefore, there is a need for agnostic industry standard development techniques that can help business analysts to specify business process models at a PIM level, and transform them to service implementation models. Moreover, by using a model-to-model mapping, the different system development stakeholders [11] provide to each other, without any conceptual mix, readily understandable solution models that are defined for their specific space. The next section shows the first step of our approach: a PIM design using the BPMN. Then, we show how this PIM maps to a PSM expressed in a domain specific language like the SCA. Finally, the PSM can be instantiated for a runtime environment and benefit from the SCA framework for deployment, execution and monitoring.

III. From a Platform Independent Model

To study the issues of a development scenario for a SOA solution, we have chosen the well-known travel booking service composition sample [12] as our motivating example. This sample illustrates a few situations that occur when modeling business process logic based on a SOA with the BPMN. Figure 1 depicts two BPMN's conversation diagrams representing the collaboration logic in the travel booking service network. In this business network, the *travel booker* participant provides the *travel booking*

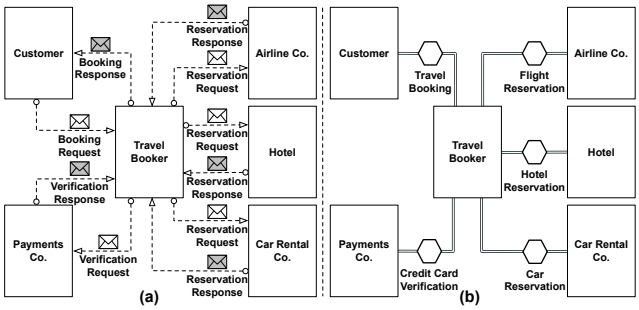

Figure 1. A sample conversation diagrams with the BPMN.

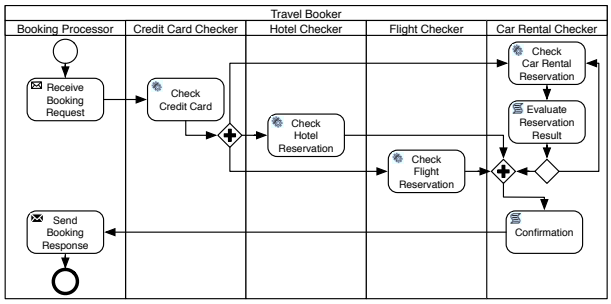

Figure 2. An orchestration process diagram with the BPMN.

service to the *customer* and sells flight tickets, reserves hotel rooms, and provides car rental. In order to provide these business services, the *travel booker* establishes business links with other partners (e.g., an *airline co.*, a *hotel*, a *car rental co.*, and a *payments co.*) exposing other services: *flight reservation*, *hotel reservation*, *car reservation*, and *credit card verification*. The diagram (a) in Figure 1 shows the message flow exchanges between those participants, and defines their interactions. The diagram (b) in Figure 1 defines a logical grouping of the message flows and service dependencies shown as diamonds, called communications.

The BPMN allows to specify system capabilities at a PIM level that support cross-domain sharing. The BPMN models express and put in logical relation the messages exchanged between internal processes of a domain's partners. Processes are focused on composition of services, and in that sense services become process activities. In turn, the processes become composite services. The business services are modeled as processes that expose their capabilities through provided interfaces and consume other services through required interfaces. A collaboration refers to the interactions between a collection of participants that represent business entities or roles, and which form a business service network. To define the business process logic of each partner pertaining to a domain being modeled, the BPMN refers to a service orchestration as a participant's private process. For example, Figure IV shows a process of the *travel booker* participant of Figure 1.

The process starts with the receipt of a travel booking request from a *customer*. After a check on a credit card verification, the reservations are made for a flight, a hotel, and a car rental with the specific business service providers (e.g., *payments co.*, *hotel*, *airline co.*, and *car rental co.*). The car reservation may need more than one attempt before to succeed. After the *confirmation* of the three reservations, a response is sent back to the *customer*. The lanes are used to organize and categorize process steps as internal sub-roles defined within the *travel booker* process. To sustain the separation of concerns [11] in the service orchestration logic, each sub-role contains tasks invoking or providing operations for a single external service provider or consumer (i.e., relative the participant). Generally, this design is simpler and less error prone than the one consisting in separating the orchestration sub-roles in separate process pool. Of course, the combinatorial considerations introduced by the later design approach makes the design activity very complex. To describe the automated orchestration behavior, the atomic process steps of work are modeled as tasks:

- A *service task* defines a service usage. It invokes operations on another external participant that provides the requested service (e.g., *check car reservation*, in Figure IV, interacts with the *car rental co.*).
- A *send task* is designed to send a message to external participants. For example, the *send booking response* task in Figure IV invokes an operation on the *customer* to send back a *booking response* message.
- A *receive task* is a task that is designed to wait for a message arrival. For example, in Figure IV, the *receive booking request* task receives the *booking request* message containing the *customer* requirements.
- Other tasks can define scripts to be interpreted by the engine executing the process (e.g., *confirmation* and *evaluation reservation result* tasks in Figure IV).

Using the BPMN at a PIM level can provide the sufficient business logic and conceptual information needed to build a complex software solution in a cross-organizational setting which is based on a SOA. From an architectural perspective, a business process model is not only intended for modeling the embedded automated process logics of the business services. It also specifies the inter-processes logic and their architectural composition settings.

IV. Mapping of the Business Process Models

Software architecture is a discipline related to the design of high-level structures in complex systems. In the context of the service-oriented engineering along with the component-based architectures, we understand this term as a framework for describing the functional attributes of a software system and their relations. Especially, the attributes which are consistent across the system specification, implementation and deployment. The BPMN provides features to specify business process models for complex service-oriented business networks. In addition, the SCA provides a component-based model to build software solutions in a SOA manner. It offers a way to logically modularize complex systems into component configurations which are assembled together to create solutions that serve a particular business need. It is a serious candidate to describe a system architecture.

Starting from a business service composition model with the BPMN, it is possible to obtain a deployable

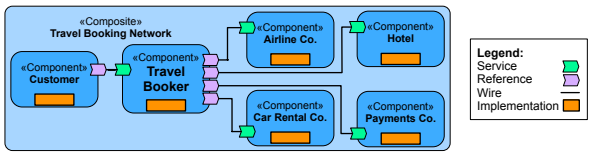

Figure 3. A travel booking monolithic implementation with the SCA.

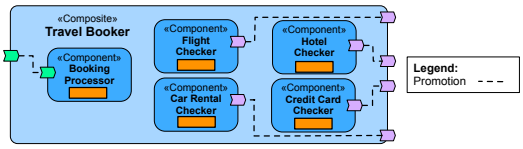

Figure 4. A travel booker implementation with the SCA.

component configuration architecture that captures the inter-process collaboration topology. For example, Figure 3 shows how the business process model of Figure 1 can map to a component configuration depicted with the graphical syntax of the SCA. The *travel booking network* composite groups and links components built from different implementation technologies, allowing appropriate technologies to be used for each business activity. Each participant in the business process model is mapped to a component in the configuration, and it is configured to interact with its partners through inter-component dependencies, called *wires*. To interact with service providers or consumers, the component exposes *services*, and consumes provided services by means of *references*. The interaction interfaces are attached to the references and the services. Each component contains the *implementation* of the business logic specified by the mapped participant's orchestration process. For example, according to a specification [13] of the Open SOA collaboration, the *travel booker* component can be implemented by a Web Services Business Process Execution Language definition that maps with the orchestration process of Figure IV. Also, depending on the SCA runtime, an executable BPMN process model can be attached to the component. The variety of the languages used to the component implementations is related to the known variety of the implementations for the SCA specifications and available tools. Obviously, depending on the IT developers choices, the component's function can be hard coded, or can be enacted with an executable business process language. It has to be noticed that we do not make any recommendation on the use of any implementation technology. For more detailed explanations on the SCA, refer to Section V.

This simplistic mapping is viable, but it has two major drawbacks. First, it restricts developers to a specific stack of technology standards. Second, it implies to fit already self-describing and deployable business process definitions into other SCA constructs to be used as deployment artifacts. Hence, this mapping implies an important coupling between the BPMN and its implementation layer. For example, it is necessary to revise both the executable processes and the SCA definitions, just to make a minor change on service access mechanisms. Moreover, to take advantage of the flexibility with the SOA and the prescriptiveness with the BPM, we must fill the gap of the weakness in the BPMN for capturing the architectural aspects of a SOA that supports the business process models. The BPMN neither specify the roles (e.g., consumer or provider) that participants are expected to play in a business collaboration, nor foreshadow the architecture of the supporting service implementation infrastructure.

Unlike the previous monolithic method, we argue that the mapping must preserve the separation of business functional concerns as they are defined in the business process models. The functions of the obtained components should be aligned with the specified functional separation of the business roles. For example, the sub-roles defined in the orchestration process of Figure IV should match with the component configuration of Figure 3. Like it is depicted in Figure 4, the concerns of the *travel booker* should be shared between separate components in a composite, rather that being hold by a monolithic implementation. Each component contained in the *travel booker* composite might depend on a single external business service provider or consumer. The references and services of the composed components are promoted externally through the composite boundary and wired to *customer*, *airline co.car rental co.*, *payments co.* external components. However, partitioning the roles of the business process tasks into different components implies resolving a problematic mapping of their process flow dependencies into equivalent inter-component dependencies. For example, in [3] the authors argue that the control flow between the *Receive Booking Request* task and the *Check Credit Card* task of Figure maps to a dependency between the *Booking Processor* and the *Credit Card Checker* components of Figure 4. Evidently, such a straight mapping of a inter-task flow into a dependency between components is a nonsense.

Furthermore, to correctly map a BPMN model to a component configuration model based on the SCA, it is necessary to differentiate the business process constructs that are used to expose business services, from those used for the service consumption. This is crucial to decide on the services and references of the components to soundly construct the assebly architecture. Of course, by introspecting the business processes, it is possible to find the tasks which capture the service interaction points. Those points represent the service interfaces that are provided and/or required by the processes. However, a mapping of inter-task flows (i.e., with a priori single execution context) into inter-component dependencies [14] (i.e., with a posteriori potentially different execution contexts) is tedious, context-dependent and pointless. In contrast, mapping the inter-process [15] dependencies to inter-component links is a more obvious method. The BPMN inter-process dependency means the topology of the message exchanges between the business services, and binds their business process logic. The SCA inter-component dependency captured by wires (i.e., with appropriate multiplicities) means a structural association between component instances, and an invocation channel for the message exchanges. The precise meaning given to a wire is defined by the wired

component function. Generally, it is related to a definition of state-related interface instantiations [16].

The inter-component interactions are similar to the inter-process invocations (i.e. with appropriate multiplicities). Consequently, the message exchanges between processes can map to the wires between components. However, to obtain a sound and canonical service component configuration from a high-level business process modeling language, we need to introduce a technique that emphasizes the separation of concerns between the BPMN design space and the SCA assembly space. This technique is intended for avoiding the straight mapping of inter-task dependencies to inter-component wires, which we call *service orchestration partitioning*. It transforms the inter-task control flow of a single orchestration process, with different roles, into message flows between separated processes, with singular roles. Also, this partitioning technique decomposes functionally monolithic business process logic into functionally equivalent decoupled logics. It ensures a more manageable component configuration architecture when the business process model is mapped to the component configuration model. For lack of space, we focus on the control flow dependencies, and intentionally put the mapping of BPMN information models out of the scope of this article.

A. The Service Orchestration Partitioning

Generally, specifying a separation of the business process concerns in a BPMN service orchestration process is limited to modeling sub-roles with separated concerns, and no detail on service realization. Mapping those business process models to the supporting system architecture is a challenge to grasp and elaborate. Certainly, mapping the business-level roles into separated component concerns can be resolved manually. However, such an intuitive transformation can produce evident design decision losses, and a misalignment of business process models with their further implementations. To produce a component configuration that sustain this pillar motivation of the SOA, we introduce a practical technique to partition the business process models. This technique transforms a BPMN process task flow into a set of independent flows that are related to each others with message exchanges.

In order, to ensure a more manageable architecture, we have to decompose a complex system into smaller decoupled components with separate concerns. The decomposition provides a better view on the dependencies between the component inside the system and their relationships with the outside of the system. Also, it makes the overall component configuration more robust and flexible to business process or IT changes. In Section IV, we have mentioned that a single BPMN process potentially maps to a component, and that the inter-process dependencies map to component wires. To separate component concerns according to their interactions with external components, we have to differentiate the business process flow parts that interact with external participants. Those process flow parts have to be partitioned into separate processes. Eventually, they can be mapped into separate components. Each component will implement the business concerns of the process to which it maps. As long as each process interact with a single external participant, the mapped component will also interact with the correspondent external service.

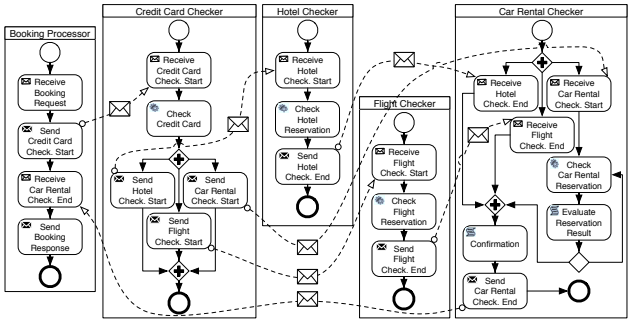

Figure 5. A partitioned orchestration process.

Based on a process flow analysis, the partitioning method takes a service orchestration, and produces an equivalent set of orchestration that aggregate tasks interacting with separate external business service provider or consumer. It derives peer-to-peer interactions among process partitions from a service orchestration process with the BPMN. We consider that the business process models that have to be mapped are valid and sound. For this purpose, we adapt an algorithm presented in [17] that was originally proposed for the decentralization of abstract service composition models. The resulting process partitions interact with each other using asynchronous messaging, and without requiring any central orchestrator process. For example, Figure 5 shows a simplified partitioning result of the *travel booker* process of Figure IV. Each obtained process represent an autonomic business role. It interacts with a unique external business service (i.e., relative to the *travel booker* participant) through the send, receive and service initially modeled tasks. The combined behavior of those partitions provides the same execution behavior as in the input service orchestration, but without any orchestrator process.

Formally, a service network, denoted $\mathcal{S}$, contains a collection of services, denoted $\mathcal{S}_i \in \mathcal{S}$. Let's consider that one of those services, denoted $\mathcal{S}_o \in \mathcal{S}$, is an orchestrating service which contains a BPMN process definition, denoted $\mathcal{O}_{\mathcal{S}_o}$. $\mathcal{O}_{\mathcal{S}_o}$ defines a set of tasks $\mathcal{A}$ providing or invoking operations on the interfaces of the other services $\mathcal{S} \setminus \mathcal{S}_o$. The subset of tasks that refers to the same service $\mathcal{S}_i \in \mathcal{S} \setminus \mathcal{S}_o$ is denoted $\mathcal{A}_{\mathcal{S}_i}$. By applying the algorithm to $\mathcal{O}_{\mathcal{S}_o}$, we obtain a new BPMN collaboration between a set of BPMN process definition partitions $\{\mathcal{O}_{\mathcal{S}_1}, \mathcal{O}_{\mathcal{S}_2}, ..., \mathcal{O}_{\mathcal{S}_n}\}$, where $\mathcal{S}_i \in \mathcal{S} \setminus \mathcal{S}_o$. Each partition $\mathcal{O}_{\mathcal{S}_i} \subseteq \mathcal{O}_{\mathcal{S}_o}$ includes only a subset of the tasks $\mathcal{A}_{\mathcal{S}_i} \subseteq \mathcal{A}$ that invoke operations on (resp., or provide for) a same service provider (resp., or consumer) $\mathcal{S}_i$. Those partitions represent proxy orchestration entities that are related to each external service. They are fine-grained internal services composing the initial service $\mathcal{S}_o$. Therefore, the partitioning introduces a view of the service registry that was not visible in the initial BPMN service composition model.

In order to generate the component configuration model, the initial participant is mapped to a high-level component. The obtained sub-participants map to distinct sub-components that are attached to the high-level component (e.g., Figure 4). They contain independent service contracts, dependencies and access mechanisms with separate service providers and/or consumers. Finally, in order to compose the component, the initial service interactions are mapped to wires between the high-level components. The generated message exchanges between the sub-participants are mapped to lower-level inter-component wires. In the following, we use the result of this partitioning to generate a PSM representing a canonical component configuration described with the SCA. This component configuration architecture is intended for representing the high-level structures of the system architecture. The configuration architecture can be refined by the system architect, and the component implementation can be detailed.

V. From Processes to Components

Defining a formal conceptual mapping between the BPMN and the SCA is an essential step toward a MDD approach. In this section, we present a subset of concepts from the PIM metamodel perspective and show how those concepts can be defined in similar PSM constructs. The rule specification consists in a mapping from any model built using types specified in the BPMN to models expressed using types from the SCA. We emphasize on how the service-oriented systems specified at a technology-neutral view can be (automatically) transformed into a PSM level. Due to lack of space, Table I presents a non-exhaustive classification of the mapping rules between a subset of the BPMN and the SCA metaclasses. This classification is based on a model type mapping [4]. An explanation on the rules and their implementation in a proof-of-concept prototype is given in the following.

Table I
Mapping rules between the BPMN and the SCA.

Name	BPMN Metaclass	SCA Metaclasses
R1	Collaboration	Domain Composite
R2	Conversation	Included Composite
R3	Participant	Component
R4	Participant Association	Composite Implementation
R5	Process	Implementation
R6	Service Task	Reference and/or Callback
R7	Send Task	Reference and/or Callback
R8	Receive Task	Service and/or Callback
R9	Communication	Wire
R10	Interface	Interface or Callback Interface

A SCA assembly model consists of a series of composites defining the configuration of a SCA domain. Configuring a SCA domain is a pure deployment concern. However, we consider a SCA domain from a conceptual view, as an area of business capabilities that are controlled by a particular organization, and contribute with those capabilities to a service network. Each capability is modularized in a *component* that represents configured instances for providing or consuming basic business functions. In turn, a *composite* is used to logically assemble

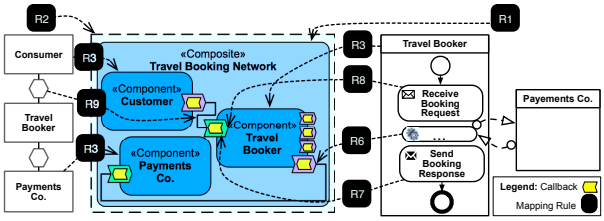

Figure 6. A mapping sample from the BPMN to the SCA.

other components in units of deployment, and contributes with its composed business logic to a domain. When a composite represents a root-level construct that contains all the composites deployed to the domain, then it is considered as a *domain composite*. To provide a recursive inclusion of capabilities, a domain composite can include other composites [16], called *included composites*. The BPMN *conversation* and *collaboration* concepts express the logical relation of message exchanges between participants involved in a service network. A *participant* represents a role in a collaboration. It executes a private business process to produce an excepted business function. These conceptual similarities with the SCA implies that a participant maps to a component (i.e., *R3*), and that its containing collaboration and/or conversation map to composites holding the component. To form a coherent component configuration, a collaboration must be mapped to a domain composite (i.e., *R1*), and a conversation mapped to an included composite (i.e., *R2*). For example, Figures 6 and 7 show how the rules from the Table I are applied to the travel booking sample depicted in Figure 1. The *travel booking network* composite represents the architecture of the collaboration. The SCA allows a hierarchical construction of component where high-level components are implemented by composites containing lower-level components.

For a service orchestration owned by a BPMN participant, our partitioning method produces a new sub-collaboration between a set of process partitions owned by sub-participants. Each resulting participant contains its own private process, and it is matched up with the initial participant through a BPMN *participant association*. Moreover, to reproduce equivalent control flow semantics to those captured by the initial orchestration, the obtained process partitions are supposed to have supplementary synchronization mechanisms. Those sub-processes directly interact with each other using asynchronous messaging with additional message flows. Since a participant maps to a component, then the obtained sub-collaboration maps to an implementing composite. Recursively, each obtained participant is mapped to a component which is placed within the implementing composite. For example, Figure 7 shows the mapping of participants resulting from the *travel booker* process partitioning. Each associated participant is mapped to components with the rule *R3*. The participant association is mapped to an implementation of the *travel booker* component with the rule *R4*. The collaboration obtained after the partitioning is maps to the *travel booker* composite which is attached to the implementation of the

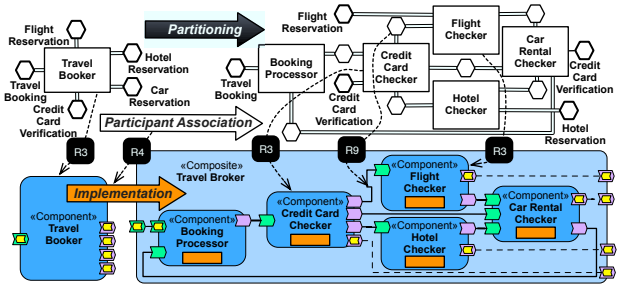

Figure 7. A mapping of BPMN conversation sample to SCA.

travel booker component. The SCA allows a broad range of technologies to be used as component implementations, and different component configurations can coexist in the same composite. The *process* of each sub-participant maps to a specific component *implementation* (i.e., *R5*).

The BPMN communication concept correspond to both *service* and *reference* in the SCA. Probably, the BPMN communications can capture basic aspects of the service contracts. However, they are not differentiated based on the business service usage direction. In the SCA, a service is different from a reference. Each of them characterizes the direction of the first exchanged message between two components. The sender of the first message is considered to be a client that invokes operations on the *interface* of a service provider component. If there is a response message going back to the client, the service provider use the *callback interface* of the sender component. In this case, a *callback* is used to provide the asynchronous messaging. When a component plays both consumer and provider roles, callbacks are defined for its services and/or references. Therefore, in order to correctly compose the components, it is necessary to determine which process task sends the first message. A simple static analysis of the process flow is considered to distinguish tasks by outgoing or incoming message interactions with the external participants.

After the orchestration partitioning (i.e., Section IV-A), each process in the collaboration has dedicated interaction interfaces with the other participants. The first *service task* maps to a reference (i.e., *R6*). Also, when it has defined incoming messages, then it is also mapped to a callback. Accordingly, a first *send task* is mapped to a reference and a first *receive task* maps to a service (i.e., *R8*). Obviously, the process flow specifications can contain a range of other send tasks (resp., receive tasks) after the first receive task (resp., a first service task) to interact with the same external participant (e.g., Figure 6). In this case, a send task (resp., a receive task) is mapped to a callback (i.e., *R7*) and it is associated with the first mapped reference (resp., service). The BPMN operations used by the tasks (i.e., contained in BPMN *interfaces*) map to SCA operations. They are associated with a SCA *interface* and/or a *callback interface* depending messaging style. Unfortunately, the BPMN 2.0 specification does not define sufficient semantics for operation calling styles between participant's interfaces, when the SCA makes the differentiation between operation call semantics. As shown

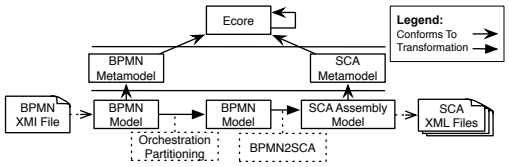

Figure 8. BPMN to SCA transformation process overview.

in Figure 6, depending on how the *travel booker* process tasks interact with the external services (i.e., relative to the participant), they map to services, references, and/or callbacks defined for the composite implementing the *travel booker* component. Also, to be correctly resolved, the external contracts defined for component services or references (i.e., contained in a composite) must be promoted by services and references defined for the composite. The synchronization tasks added by the service orchestration partitioning are similarly mapped by the previous rules. For example, the tasks defining the message flows between the process partitions of Figure 5 map to the constructs defining the interaction between the component interfaces contained in the *travel booker* composite of Figure 7. Obviously, the component configuration architecture described with the wires is structurally aligned with the inter-process relations defined by the BPMN *communication*. Figures 6 and 7 show how communications map to wires (i.e., *R9*) between the components.

Finally, to automate the generation of deployable component configuration models from the business process models, we use the ATLAS Transformation Language [18] chain as depicted in Figure 8. This proof-of-concept prototype involves metamodels transformations. The BPMN and the SCA metamodels are created using the core Eclipse Modeling Framework (Ecore [19]). The BPMN diagrams and the SCA definitions metamodels are specified in terms of the eXtensible Markup Language schema definition (XML). Rather than using intermediate model facilities, we advocate a direct transformation process (i.e., based on XML Schemas) that starts off from a BPMN XML Metadata Interchange (XMI) file containing a business service composition model (e.g., Figures IV and 1). The orchestrations contained in the previous model are partitioned into a set of sub-orchestrations. They form a second BPMN model (e.g., Figure 5). Subsequently, this model is transformed into a SCA assembly model contained in *XML files*. The resulting component configuration and composite structures (e.g., Figures 6 and 7) describe the system architecture. Eventually, realizing some change in a specific single business service contract brings into play only a part of the component configuration architecture. Also, it becomes possible only when looking at the component configuration architecture to localize the concerned parts to be changed, without spying on the component implementations.

VI. Summary and Outlook

In this article, we have presented a model-driven development methodology for inter-organizational systems that

are based on a SOA. The approach advocates a service-oriented IT infrastructure provision from a business process specification while tackling with their architectural alignment issues. The method relies on development scenarios where long-running business process model specifications based on a SOA drive component configurations for runtime platforms based on distributed architectures. When designing PIMs with the BPMN, business analysts focus on the business logic without being bothered by any implementation-specific configuration information.

The introduced mapping rules and their implementation in a proof-of-concept prototype enable the automatic transformation of BPMN models to SCA assembly models representing deployable component configurations. The mapping does not require neither further manual architectural adjustment on the generated component configuration, nor intermediate models. Thus, less design decisions are lost during the model transformation process. The presented method of BPMN orchestration partitioning provides to system architects a "what you model is what you get" design manner for the MDD. Once partitioning takes place, each orchestration specification depends on a single service provider or consumer. This principle provides a strict indication on the final PSM output that consists of a SCA assembly model. It adds a clear architectural view about how fine grained services are composed, that the BPMN fails to capture. Finally, when component configuration models are generated, they are aligned at the best with the business-level logic. The reuse of some existing services as well as creating new services to be reused in other parts of a global SOA solution become more evident for IT developers.

From an IT perspective, the obtained component configurations can be instantiated for a SCA runtime environment and benefit from the SCA framework for deployment, execution and monitoring. The decomposition in smaller decoupled components ensures a more manageable architecture. It also provides a better view on the dependencies between the component inside the system and their relationships outside of the system, making the overall environment more robust to business process or implementation changes.

At the actual stage of our work, we are not tackling with the expression of non-functional requirements. However, it is an important aspect of service orchestration and has an impact service implementation. In our future work, we want to integrate the SOA Modeling Language profile into our MDD approach to accommodate the BPMN conceptual limits. Also, we want to improve our methodology to reach an effective business-IT agility and a round-tripping between the component configuration and the business process spaces.

REFERENCES

[1] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *Computer*, vol. 40, no. 11, pp. 38–45, 2007.

[2] C. Huemer, P. Liegl, R. Schuster, H. Werthner, and M. Zapletal, "Inter-organizational systems: From business values over business processes to deployment," in *DEST'08*, 2008, pp. 294 –299.

[3] A. Mos, A. Boulze, S. Quaireau, and C. Meynier, "Multi-layer perspectives and spaces in soa," in *SDSOA '08*, New York, USA, 2008, pp. 69–74.

[4] T. Stahl and M. Völter, *Model-Driven Software Development: Technology, Engineering, Management*. Chichester, UK: Wiley, 2006.

[5] *Business Process Model and Notation (BPMN) 2.0, Beta 1*, OMG, May 2009.

[6] *SCA Assembly Model Specification 1.1*, Open SOA Collaboration, Mar 2009.

[7] *Reference Model for SOA 1.0, Committee Specification*, OASIS, Feb 2006.

[8] R. Gitzel, *Model-driven Software Development Using a Metamodel-based Extension Mechanism for Uml*. Peter Lang Publishing, 2006.

[9] D. Lopes, S. Hammoudi, J. Bézivin, and F. Jouault, "Generating transformation definition from mapping specification: Application to web service platform," in *CAiSE '05*, 2005, pp. 309–325.

[10] R. Ravichandar, N. C. Narendra, K. Ponnalagu, and D. Gangopadhyay, "Morpheus: Semantics-based incremental change propagation in soa-based solutions," in *SCC '08*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 193–201.

[11] M. Rosen, B. Lublinsky, K. T. Smith, and M. J. Balcer, *Applied SOA: Service-Oriented Architecture and Design Strategies*. Wiley P., 2008.

[12] S. A. White, *Using BPMN to Model a BPEL Process*, BPTrends, May 2005.

[13] *SCA WS-BPEL Client and Implementation 1.1*, Open SOA Collaboration, Mar 2009.

[14] J. A. Stafford and A. L. Wolf, "Architecture-level dependence analysis in support of software maintenance," in *ISAW '98*. New York, NY, USA: ACM, 1998, pp. 129–132.

[15] G. Grossmann, M. Schrefl, and M. Stumptner, "Modelling inter-process dependencies with high-level business process modelling languages," in *APCCM '08*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2008, pp. 89–102.

[16] J. Marino and M. Rowley, *Understanding SCA (Service Component Architecture)*. Addison-Wesley Professional, 2009.

[17] W. Fdhila, U. Yildiz, and C. Godart, "A flexible approach for automatic process decentralization using dependency tables," in *ICWS '09*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 847–855.

[18] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, "Atl: A model transformation tool," *Sci. Comput. Program.*, vol. 72, no. 1-2, pp. 31–39, 2008.

[19] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2009.