From Business Process to Component Architecture: Engineering Business to IT Alignment

Karim Dahman, François Charoy, Claude Godart

From Business Process to Component Architecture: Engineering Business to IT Alignment

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*—Maintaining the alignment between the Business and IT is of high strategic relevance in today's enterprise roadmap. In this paper, we follow our previous assumption that this alignment will be better maintained if we are able to ensure a clear conceptual alignment between the Business Processes and the Software Architectures. As our aim is to provide an environment that would flawlessly support evolutions of the processes or of the architecture while maintaining this alignment, we build on the formal foundation that we have developed to ensure it and shows how it can be actually developed with current Model Driven Engineering technologies.**

***Index Terms*—Unidirectional Model Transformation, Business-IT Alignment, BPMN, SCA, Conceptual Mapping, ATL**

I. Introduction

For any enterprise that aims to conduct a successful Business, being able to model its architecture is now an important asset. Furthermore, being able to relate this business architecture to its supporting Information Technology (**IT**) is recognized as a driver for the efficiency of enterprise computing resource management. One of the claimed goal for the integration between the business dimension and the IT dimension is to allow the agility in conducting the business, and adapting it to evolving business demands. The challenges to enforce this alignment from the business strategic perspective to the IT still need to be addressed. New approaches promoted through Service Oriented Computing (**SOC**), Business Process Management (**BPM**) and Enterprise Architecture are meant to be efficient enablers of this alignment and of its maintenance. Among these three disciplines, the BPM along with the SOC paradigm have an interesting quality. The BPM provides a direct bridge with almost no transformation between analysis, design, implementation and execution. Moreover, the SOC is supported by the Service Oriented Architecture (**SOA** [1]) architectural foundation, and has emerged as a new computing paradigm for designing, building and using software applications to support business processes. The Business Process Modeling Notation (**BPMN** 2.0 [2]) standard-related practices try to sustain this bridge by letting business analysts and application architects work on the same readily understandable model to obtain a (business process driven) software system.

We are still far from the streamlined experience where we could derive deployable software artifacts with sound architectural foundation from a BPMN collaboration diagram. This is even more true when we need to take into account the deployment topology of heterogeneous, distributed and continuously changing IT environments to support the process executions. Our goal, however, is to show that it would be possible to automate that business production line from the service-enabled process dependencies into the architectural software settings and its maintenance, while using the industry recognized standards like the BPMN and the Service Component Architecture (**SCA** 1.1 [3]). In a previous work [4], we have shown that its was possible to leverage the Model Driven Engineering (**MDE**) to generate SCA models from a subset of the BPMN models. The contribution of this paper is to demonstrate how these theoretical results can be implemented using current model transformation technologies. For this purpose, we have developed an ATLAS Transformation Language (**ATL** [5]) transformation chain that is able to generate canonical SCA artifacts from a BPMN collaboration model. We have also identified how this technology will be an issue for our future development to maintain the Business Processes/SOA Assets alignment as it does not support incremental transformations. This is an important problem regarding the goal that we are pursuing: being able to keep the consistency and to maintain the alignment of the processes with the software architectures in every directions (i.e., round tripping), thus supporting agility from the realistic business settings with complex IT environments.

The next section of this paper is a short reminder of our previous work. Section III describes the theoretical result for the implementation of our ATL transformation chain in the Eclipse development environment [6]. In Section IV, we situate our work through the related research and discuss the limit that we have encountered during this work. The last part conclude that work with a brief evaluation of other alternatives.

II. Background and Formal Foundations

Typically, the Business Process Modeling combines graphical and textual annotations to specify the *business process models* that are decoupled from their supporting technical and *software architectures*. Several languages can be used to specify the business service interactions at different levels of abstraction [7]. Specially, the **BPMN collaboration models** express and put in logical relation the exchanges between the internal processes of service-enabled business networks. They aim at coordinating an explicit exchange of information through automated service interactions under the control of a single endpoint, called *orchestration process* [1]. In the

SOC those processes are mapped into models that describe services in a uniform and technology-independent way. At this lower abstraction level, i.e., the IT level, they describe the configuration of the so-called **SCA assembly models** which are organized in a SOA. The SCA assemblies modularize and compose service-enabled business functions in a manner that is decoupled from their application code. The SCA provides an Architecture Description Language (**ADL**) that offers means to define platform independent component kit architectures and a path from the business goals to the program code (e.g., executable process languages). With a graphical notation support, it describes composites of components, their connections and other related artifacts which specify how the component functions are externally consumed and/or offered.

The SOC extends the Component Based Development paradigm which states that applications expose their functionality as services in a uniform and technology-independent way such that they can be provided and invoked over loosely-coupled architectures. Consequently, the service-enabled business functions can be implemented as components with remotely published interfaces in a network. Our previous work has introduced a MDE approach for the automated generation of **(target) canonical SCA assembly models** from **(source) core BPMN collaboration models**, and specified relations among their metamodels as *conceptual mappings* [4]. In our semantics, *models* are conform to a *metamodel*. As a part of this view, the models are viewed as assorted collections of typed objects with attributes, and typed links among the objects. The mapping between the heterogeneous BPMN and SCA models are viewed as rules that establish relations between the model types. We refer to the program executions that implement conceptual mappings as *transformations*. Figure 1 depicts an example of a model transformation which will be used through the reminder of this paper. It illustrates few situations that occur when the "task-driven process logic" has to be transformed into the "component composition logic". In the next section, we present the syntax of those models.

A. The BPMN and SCA Model Syntaxes

At the BPMN level, the processes are focused on the composition of services in a business driven-fashion, i.e., the orchestration process, and in that sense the *business services become process activities*. Moreover, the *processes become composite business services* that orchestrates other services and the business service networks are modeled as BPMN collaborations. A collaboration refers to the interactions between a collection of participants which represent the business entities or roles. They form a business service network and 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. Consider, the well-known BPMN *dealing network* example [2] depicted in Figure 1. In this business service network, a *retailer* provides an *ordering* service to a *customer* partner. In order to provide this service, the *retailer* establishes business links with other partners (e.g., *scheduler*, *invoicer*, and a *shipper*), and orchestrates

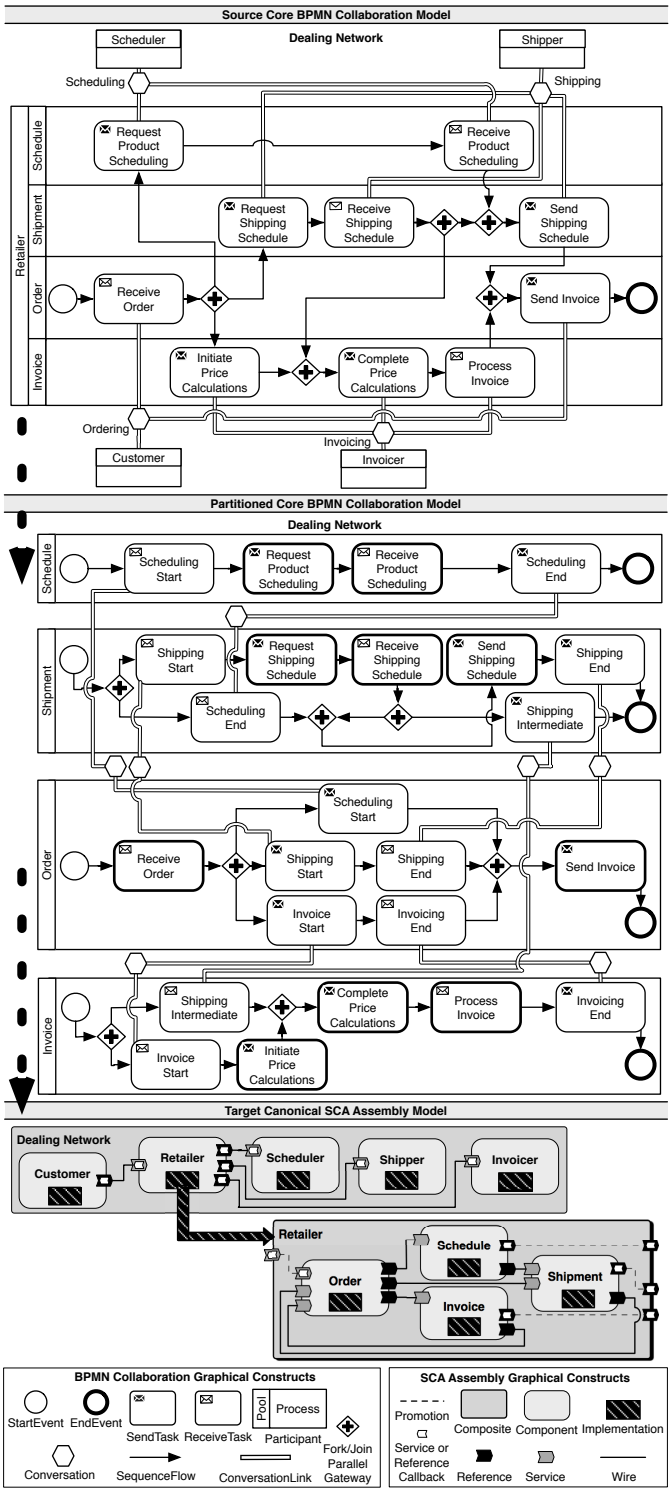

Fig. 1. BPMN-to-SCA transformation example and languages constructs.

their exposed services: *scheduling*, *invoicing*, and *shipping*. To sustain the separation of concerns in the service "orchestration logic" [8], the *retailer* participant's private process is modeled with different sub-roles (e.g., *order*, *schedule*, *invoice* and *shipment*). Those roles contain tasks which invoke or provide operations for a single external service provider (i.e., relative

to the *retailer*). The collaboration specifies the behavioral and architectural views of the business structures and their participant roles. The business links between participants are modeled as conversations that capture the service contracts. Formally, we define those models as below.

*Definition 1 (***Source Core BPMN Collaboration Model):** A source core BPMN collaboration model is a tuple $s = (\mathcal{P}, \mathcal{R}, \mathcal{A}, \mathcal{A}^{\triangleright}, \mathcal{A}^{\triangleleft}, \mathcal{E}, \mathcal{F}, conv, \mathcal{H}, \mathcal{L})$ where $\mathcal{P}$ is a set of **participants** which can be partitioned to disjoint sets of **sub-roles** $\mathcal{R}$ and **interaction tasks** $\mathcal{A}$ which can be partitioned into disjoint sets of **send tasks** $\mathcal{A}^{\triangleright}$, **receive tasks** $\mathcal{A}^{\triangleleft}$, $\mathcal{E} \subseteq \mathcal{P} \times \mathcal{P}$ is a **participant association** relation, $\mathcal{F} \subseteq \mathcal{A} \times \mathcal{A}$ is the **sequence flow** relation between the tasks, $conv : \cup_{p \in \mathcal{P}} \mathcal{A}_p \rightarrow \cup_{p \in \mathcal{P}} \mathcal{A}_p$ is an injective function which maps interaction tasks, $\mathcal{H} = \{(a, a') \in \mathcal{A}_p \times \mathcal{A}_{p'} \mid \exists_{p,p' \in \mathcal{P}} conv(a) = a'\}$ is the **conversation** relation, and $\mathcal{L} \subseteq \mathcal{R} \times \mathcal{A}$ is a **process** relation.

Likewise, when mapped to the SCA space, the *services become components*. The processes are executed by service-components that contain the effective implementation artifacts. The inter-component interactions are conceptually similar to the inter-process invocations. Consequently, the conversations between processes can map to the connectors between components. It must to be noticed, that we intentionally hide definition of the rest of BPMN constructs shown in Figure 1 (e.g., events, sequence flows and gateways), since those types have not conceptually equivalent counterpart types in SCA as explained in Section II-B. In order to generate sound component compositions we have introduced in [4] a technique that emphasizes the separation of service modeling concerns between the BPMN design space and the SCA level. This technique, which we called **orchestration process partitioning**, is intended for avoiding the straight mapping of the inter-task dependencies to the component connectors. As shown in Figure 1, it transforms the inter-task control flow of a single orchestration process with different roles, e.g., *retailer*, into a new collaboration between separated processes, with singular roles. Also, it decomposes monolithic "task-driven logic" into functionally equivalent decoupled logics. Each resulting participant, e.g., *order*, *schedule*, *invoice* and *shipment*, contains its own private process with additional synchronization tasks.

Due to the lack of space, we do not give details of the partitioning algorithm. Those processes directly interact with each other with additional conversation. Formally, for any $p \in \mathcal{P}$, the subset of interaction tasks $\mathcal{A}_p$ that have conversations with the same external participant (i.e., service denoted $p'' \in \mathcal{P}$) is denoted $\mathcal{A}_{p \bowtie p''} = \{a \in \mathcal{A}_p^{\triangleright} \cup \mathcal{A}_p^{\triangleleft} \mid \exists a'' \in \mathcal{A}_{p''} a \mathcal{H} a''\}$. By applying the orchestration partitioning algorithm to its process, i.e., denoted $\mathcal{L}_p \subseteq \mathcal{R}_p \times \mathcal{A}_p$, we obtain a set of participants $\mathcal{P}'$ where $\mathcal{P}' = \mathcal{R}_p$ and $\forall_{p' \in \mathcal{P}'} \mid \exists_{p'' \in \mathcal{P}} \mathcal{A}_{p'} = \mathcal{A}_{p \bowtie p''} \wedge \mathcal{R}_{p'} = \{p'\}$. Each obtained participant, i.e., "process partition" denoted p', includes only a subset of the tasks that invoke (resp., provide) operations on a same service provider, i.e. denoted p'' (resp., consumer). Those partitions represent "proxy" orchestrations that are related to each external service. They are fine-grained internal services composing the initial service (i.e., relative to the initial participant p) and they are matched up with p through participant associations, i.e., $\mathcal{E} \subseteq \cup_{p \in \mathcal{P}} \times \cup_{p' \in \mathcal{P}'}$. In this case, we refer to the initial participant as an *associated participant*. The orchestration partitioning ensures a more manageable component architecture and adds a clear architectural view about how fine-grained services are composed, that the BPMN per se fails to capture.

In the SCA, the assembly models logically modularize and compose components in a manner that is decoupled from their code. A component can be a composite (i.e., an assembly of components glued together using some "composition logic") or atomic when it is considered without its inside structure. The components expose their provided ports, i.e., called services, and require (i.e., or discover) other services by means of references. They are configured to interact with the other components through the connectors, i.e., called wires. Also, the callbacks can be defined for services and references when the components play per se both the service consumer and provider roles. Finally, each component contains the implementation with an appropriate technology of the process specified by the mapped participant's orchestration. We define those models to be used for the further formal foundations of the model transformation which are given in the next section.

*Definition 2 (***Target Canonical SCA Assembly Model):** A target canonical SCA assembly model, i.e., assimilated to a composite, is a tuple $t = (\mathcal{B}, \mathcal{M}, \mathcal{N}, \mathcal{C}, \mathcal{O}, \mathcal{Q}, \mathcal{K}, \mathcal{G}, \mathcal{J}, \mathcal{I}, \mathcal{R}, \mathcal{V}, \mathcal{W})$ where $\mathcal{B}$ is a set of **sub-composites** which can be partitioned to disjoint sets of **composite services** $\mathcal{M}$ and **composite references** $\mathcal{N}$, $\mathcal{C}$ is a set of **components** which can be partitioned into disjoint sets of **component services** $\mathcal{O}$ and **component references** $\mathcal{Q}$, $\mathcal{K}$ is a set of **callbacks**, $\mathcal{G} \subseteq (\mathcal{M} \cup \mathcal{N}) \times \mathcal{K}$ is the **composite callback** relation, $\mathcal{J} \subseteq (\mathcal{O} \cup \mathcal{Q}) \times \mathcal{K}$ is the **component callback** relation, $\mathcal{I}$ is the of **implementations**, $\mathcal{R} \subseteq (\mathcal{M} \times \mathcal{O}) \cup (\mathcal{N} \times \mathcal{Q})$ is the **promotion** relation, $\mathcal{V} \subseteq \mathcal{C} \times \mathcal{B}$ is the **implementation as composite** relation, and $\mathcal{W} \subseteq (\mathcal{M} \cup \mathcal{Q}) \times (\mathcal{N} \cup \mathcal{O})$ is the **wire** relation.

B. The BPMN to SCA Conceptual Mapping

Before we delve into the formal details, a look at the general model transformation problem is needed. Given two metamodels $\mathcal{S}$ and $\mathcal{T}$, the *model-to-model transformation*, denoted $trans_{map} : \mathcal{S} \rightarrow \mathcal{T}$, is a partial function that takes a source model s in $\mathcal{S}$ and produces a target model t in $\mathcal{T}$. The source-to-target types correspondences are defined by a partial function, denoted map, which is based on the semantics of the source core BPMN collaboration and the target SCA assembly types. This means that only the source model objects with types that has proper conceptual mapping into target types are transformed. For example, only the tasks that specify service interactions (i.e., send or receive) are transformed. The other constructs such as gateways and events are not relevant for sketching the component assembly, but,

define functional aspects of the component implementation. Moreover, we define an **index** function as follows: $\forall_{p,p' \in \mathcal{P}} \mid \exists_{a \in \mathcal{A}_p, a' \in \mathcal{A}_{p'}}, a\mathcal{H}a'$, $\underline{index(a) = initial^{p'}}$ if $\nexists_{a'' \in \mathcal{A}_p} \mid a'' \in prev^*(a)$ and $\underline{index(a) = final^{p'}}$ if $\nexists_{a'' \in \mathcal{A}_p} \mid a'' \in next^*(a)$. This function is used to differentiate the tasks in the service interaction patterns [9] in the following definition. The functions denoted $prev^*(a) = \{a' \in \mathcal{A}_p \mid a'\mathcal{F}_p^* a\}$ and $next^*(a) = \{a' \in \mathcal{A}_p \mid a\mathcal{F}_p^* a'\}$ give the set of all direct and transitive predecessors and successors of a task a, where $\mathcal{F}_p^*$ is the reflexive transitive closure of $\mathcal{F}_p$. A static analysis of the process flow is considered to distinguish tasks in conversations. We formally define the mapping below.

*Definition 3 (***Collaboration to Composite Mapping):** Let $s = (\mathcal{A}, \mathcal{R}, \mathcal{P}, \mathcal{A}^{\triangleright}, \mathcal{A}^{\triangleleft}, \mathcal{E}, \mathcal{F}, conv, \mathcal{H}, \mathcal{L})$ be a well-formed and well-behaved source core BPMN collaboration model (i.e., given in Definition 1). s can be conceptually mapped to onto a target canonical SCA assembly model, i.e., assimilated to a composite $map(s) = (\mathcal{B}, \mathcal{M}, \mathcal{N}, \mathcal{C}, \mathcal{O}, \mathcal{Q}, \mathcal{K}, \mathcal{G}, \mathcal{J}, \mathcal{I}, \mathcal{U}, \mathcal{V}, \mathcal{W})$ where

- $\mathcal{B} = \bigcup_{p \in \mathcal{P}} \{p \mid \exists_{p' \in \mathcal{P}}\ p\mathcal{E}p'\}$, i.e., **associated participant to composite**,
- $\mathcal{M} = \bigcup_{p \in \mathcal{P}} \{a \in \mathcal{A}_p^{\triangleleft} \mid \exists_{p',p'' \in \mathcal{P}}\ p\mathcal{E}p' \wedge index(a) = initial^{p''}\}$, i.e., **initial receive task to composite service**,
- $\mathcal{N} = \bigcup_{p \in \mathcal{P}} \{a \in \mathcal{A}_p^{\triangleright} \mid \exists_{p',p'' \in \mathcal{P}}\ p\mathcal{E}p' \wedge index(a) = initial^{p''}\}$, i.e., **initial send task to composite reference**,
- $\mathcal{C} = \mathcal{P} \setminus \bigcup_{p \in \mathcal{P}} \{p \mid \exists_{p' \in \mathcal{P}}\ p\mathcal{E}p'\}$, i.e., **participant to component**,
- $\mathcal{O} = \bigcup_{p \in \mathcal{P}} \{a \in \mathcal{A}_p^{\triangleleft} \mid \nexists_{p' \in \mathcal{P}}\ p\mathcal{E}p' \wedge \exists_{p'' \in \mathcal{P}}\ index(a) = initial^{p''}\}$, i.e., **initial receive task to component service**,
- $\mathcal{Q} = \bigcup_{p \in \mathcal{P}} \{a \in \mathcal{A}_p^{\triangleright} \mid \nexists_{p' \in \mathcal{P}}\ p\mathcal{E}p' \wedge \exists_{p'' \in \mathcal{P}}\ index(a) = initial^{p''}\}$, i.e., **initial send task to component reference**,
- $\mathcal{K} = \bigcup_{p \in \mathcal{P}} \{a \in \mathcal{A}_p^{\triangleleft} \cup \mathcal{A}_p^{\triangleright} \mid \exists_{p' \in \mathcal{P}, a' \in \mathcal{A}_p}\ a \in next^*(a') \wedge index(a') = initial^{p'} \wedge index(a) = final^{p'}\}$, i.e., **final receive task or final send task to callback**,
- $\mathcal{G} = \bigcup_{p \in \mathcal{P}} \{(a, a') \in \mathcal{A}_p^{\triangleleft} \times \mathcal{A}_p^{\triangleright} \mid \exists_{p',p'' \in \mathcal{P}}\ p\mathcal{E}p' \wedge a \in prev^*(a') \wedge index(a) = initial^{p''} \wedge index(a') = final^{p''}\}$, i.e., **associated receive task relation to composite callback relation**,
- $\mathcal{J} = \bigcup_{p \in \mathcal{P}} \{(a, a') \in \mathcal{A}_p^{\triangleleft} \times \mathcal{A}_p^{\triangleright} \mid \nexists_{p' \in \mathcal{P}}\ p\mathcal{E}p' \wedge \exists_{p'' \in \mathcal{P}}\ a \in prev^*(a') \wedge index(a) = initial^{p''} \wedge index(a') = final^{p''}\}$, i.e., **associated send task relation to component callback relation**,
- $\mathcal{I} = \bigcup_{p \in \mathcal{P}} \{p \mid \nexists_{p' \in \mathcal{P}}\ p\mathcal{E}p' \wedge \exists_{a \in \mathcal{A}_p, r \in \mathcal{R}_p}\ r\mathcal{L}a\}$, i.e., **process to implementation**,
- $\mathcal{U} = \bigcup_{p \in \mathcal{P}} \{(a, a') \in (\mathcal{A}_p^{\triangleleft} \times \mathcal{A}_{p'}^{\triangleleft}) \cup (\mathcal{A}_p^{\triangleright} \times \mathcal{A}_{p'}^{\triangleright}) \mid \exists_{p',p'' \in \mathcal{P}}\ p\mathcal{E}p' \wedge index(a) = initial^{p''} \wedge index(a') = initial^{p''}\}$, i.e., **associated task relation to promotion relation**,
- $\mathcal{V} = \bigcup_{p \in \mathcal{P}} \{(p, p') \in \mathcal{P} \times \mathcal{P} \mid \exists_{p' \in \mathcal{P}}\ p\mathcal{E}p'\}$, i.e., **participant association to implementation as composite**,
- $\mathcal{W} = \bigcup_{p \in \mathcal{P}} \{(a, a') \in \mathcal{A}_p^{\triangleright} \times \mathcal{A}_{p'}^{\triangleleft} \mid \exists_{p' \in \mathcal{P}}\ index(a) = initial^{p'} \wedge index(a') = initial^{p} \wedge a\mathcal{H}a'\}$, i.e., **conversation to wire**.

Note that we assume the well-formedness of the source models and its well behavedeness. It means they are consistent with the structural and the behavioral requirements of service interaction specification, e.g., defined in [9]. For the minimal correctness requirements on the structure of the processes we follow [10], e.g., there is a single start event, there is one or more end events, and every process task is on a path from the start event to an end event. Due to lack of space, we refer to [11] for detailed structural requirements on the SCA models. Since the BPNM participants map into SCA components, then, after the orchestration partitioning, the obtained participant associations between the new participants map into the implementations as composites. For example, Figure 1 shows the mapping of participants resulting from the *retailer's* process partitioning. The associated participant is mapped to a component. The collaboration obtained after the partitioning maps into the *retailer* composite which is attached to the implementation of the initial *retailer* component. Recursively, each obtained sub-participant, e.g., *order*, *schedule*, *invoice* and *shipment*, is mapped onto a component, and it is placed within the implementing composite. The process of each sub-participant maps to a single component implementation. This allows different technologies to be used for each process implementation. It has to be noticed that we do not make recommendations on the technology usage.

III. BPMN to SCA Model Transformation

In the previous section, we have introduced the conceptual mapping for the transformation of canonical SCA assembly models from core BPMN collaboration models. This section performs a more through investigation on the implementation of the *transformation code*. In order to implement the transformation of the model artifacts, we use the ATL transformation chain which is depicted in Figure 2. The ATL is integrated in the Eclipse development environment and can handle models based on the core Eclipse Modeling Framework (**Ecore** [6]). It also provides support for models using EMF-based UML profiles. In our proof-of-concept prototype, the BPMN and SCA metamodels are created using the Ecore. The BPMN diagrams and the SCA definitions metamodels are specified in the eXtensible Markup Language Metadata Interchange (**XMI**).

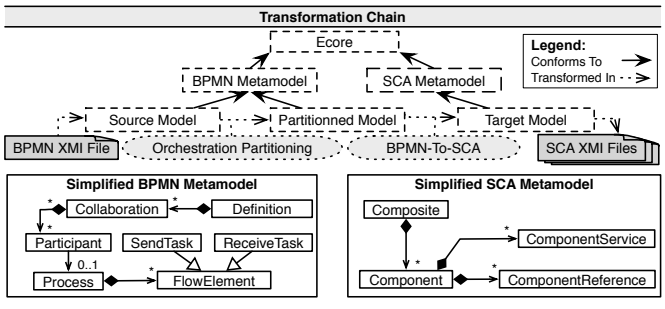

Fig. 2. BPMN-to-SCA transformation ATL chain overview.

Rather than using intermediate model facilities [12], we

advocate a direct transformation chain that starts off from a BPMN XMI file containing the source core BPMN collaboration model. Then, its orchestrations are partitioned. Subsequently, this model is transformed into a target canonical SCA assembly model resulting in XML files which contain the Service Component Definition Language (**SCDL** [3]) of the mapped composites, i.e., the particular ADL of the SCA. An ATL-code is composed of rules describing how to generate the objects of the target models. It is compiled into the ATL bytecode and then executed by the ATL virtual machine [5]. As a simple running example, consider the following *BPMNtoSCA* ATL program, i.e., the module.

```
module BPMNtoSCA;
create OUT : SCA from IN : BPMN;
helper context BPMN!Definitions
    def : CollaborationHasParticipants() : Boolean =
    if (self.getCollaboration().
        participants.oclIsUndefined()) then false
    else true endif;
helper context BPMN!Definitions
    def : getDefinitions() : BPMN!Definitions = self;
helper context BPMN!Definitions
    def : getCollaboration() :  BPMN!Collaboration =
    (self.rootElements -> select (e | e.oclIsTypeOf(
        BPMN!Collaboration))).asSequence().first();
rule CollaborationtoComposite {
from    d: BPMN!Definitions, s: BPMN!Collaboration
to  t: SCA!Composite(name <- s.name,
    component <- BPMN!Participant.allInstancesFrom('IN')
    -> collect(e | if not e.processRef.oclIsUndefined()
        thisModule.ParticipanttoComponent(e,
                    d,e.processRef))}
lazy rule ParticipanttoComponent {
from    d: BPMN!Definitions, p: BPMN!Participant,
    l: BPMN!Process
to  c: SCA!Component(name <-
    if d.CollaborationHasParticipants()=true
        then p.name
    else d.getCollaboration().name endif,
    reference <- BPMN!SendTask.allInstancesFrom('IN')->
    select(e | l.flowElements.includes(e))->
    collect(e | thisModule.SendTaskToReference(e)),
    service <- BPMN!ReceiveTask.allInstancesFrom('IN')
    -> select(e | l.flowElements.includes(e))->
    collect(e | thisModule.ReceiveTaskToService(e)))}
lazy rule SendTaskToReference {
from    sen: BPMN!SendTask
to  ref: SCA!ComponentReference(name <- sen.name)}
lazy rule ReceiveTaskToService {
from    rec: BPMN!ReceiveTask
to  ser: SCA!ComponentService (name <- rec.name)}
```

It defines four rules from the conceptual mapping given in Definition 3. Those rules uses the simplified BPMN and SCA metamodels shown in Figure 2. The ATL is a hybrid model-to-model transformation language that supports both declarative and imperative constructs [13]. The preferred style is declarative, which allows simpler implementation for relatively simple conceptual mappings. However, imperative constructs are provided so that some not trivial mappings to be handled declaratively can still be specified. Roughly speaking, the ATL program transforms a collaboration to a composite and each participant of the collaboration to a component of the corresponding composite. Also, the *rules* map the send and the receive task in a straightforward manner to component services and references. The order of the rule execution is determined automatically, with the exception of *lazy rules*, which needs to be called explicitly. The *helper* functions provide imperative constructs.

In order to ensure the consistent BPMN-to-SCA mapping, we partition the service-enabled business process into smaller decoupled service-components with separate concerns. The orchestration process partitioning is written with imperative constructs (i.e., not presented in the above ATL-code due to lack of space). It provides a better view on the dependencies between the component inside the application being modeled (e.g., *order*, *schedule*, *invoice* and *shipment* within *retailer* in Figure 1) and their relationships with the outside of the system (e.g., *customer*, *scheduler*, *invoicer*, and a *shipper*). Also, it makes the overall component architecture more robust and flexible for the later business process or IT evolutions. Eventually, realizing some change in a specific single business service contract brings into play only a part of the component configuration architecture. Finally, it becomes possible only when looking at this SCA architecture to localize the concerned parts to be changed, without spying on the opaque application code. It is undeniable, however, that the flexibility to adapt the processes in order to respond to evolving business needs is the most relevant trend in the MDE success.

IV. Related work

In order to align the Business and IT capabilities, various organizational considerations need to be taken into account [14], [15]. In this paper, we focus on the Business-IT alignment through the business processes, particularly, on the generation of the software architectures from service-enabled (business process driven) system specifications. Several languages can be used to specify the service interaction models at different levels of abstraction [7]. Using the BPMN 2.0 [2] can provide the sufficient conceptual information needed to specify a complex service-enabled system in a cross-organizational setting. Likewise, the SCA [3] provides a framework for creating software solutions based in a multi-language and multi-platform environment that are based on complex IT environments. As an alternative to the BPMN along with the SCDL, one could use UML Activity Diagrams and Sequence Diagrams, which offer comparable features along with any other ADL. However, as the UML is neither fully service-centered, nor business process-oriented, it is necessary to provide alternative methods to design the business process logic independently of their implementation infrastructure. Thus, the MDE techniques needs to address the gap in existing UML modeling methodologies for the system engineering with a Service Dominant Logic [14], [15].

In [12], the authors present a MDE 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 model [3]. The approach is promising, however, the absence of accurate mappings between the two metamodels makes the model transformation not viable. First, it requires further manual adaptation to reflect the initial process requirements because the BPMN per se neither specifies the roles (e.g., consumer or provider) that participants are expected to play in a business collaboration. Of course, adapting the

generated SCA configurations necessarily leads to their misalignment with the business process layer, and implies design decision losses. Second, the provided automated incremental transformation is fictional, since from the initial mapping, the source constructs are mapped to conceptually different target elements. Thus, providing consistency management over this approach turns out to very challenging endeavor in practice. Furthermore, the *change propagation* is a major use case for the MDE. When a source model is changed and that a transformation has previously generated a corresponding target model, it is necessary to keep the models synchronized. The model transformation languages [13], specifically the ATL [5], cannot work well here. It supports a mode for in-place transformation, called the refining mode [5], but it does not support incremental model transformations: a complete source model is read and a complete target model is created. Thus, the manual changes in the target model are not preserved [16]. Morevover, the Query/View/Transformation (**QVT**) is a standardized language for model unidirectional transformations. Some of the issues for the QVT usage was raised in [17]. The ATL supports only unidirectional transformation code. First, it requires to explicitly write the *synchronization code* to deal with each kind of update on each type of assorted models. Second, the complexity of each combinatorial change mapping is inherently compounded with decisions regarding the potential information loss or gain related to different levels of model's expressiveness. They make the synchronization code much complex, particularly, when not all the source model elements correspond the to target model elements. For example, it is necessary to revise both the SCA definitions and their implementations, just to make a minor change on the service interaction patterns [9] at the BPMN level.

V. Summary and Outlook

In this paper, we have presented a development scenarios for the SOC, where business process specifications drive component architectures that are organized in a SOA. At the design-time, the business analyst focus on the modeling of task-driven and service-oriented logics without being bothered by any implementation-specific configuration information. The introduced conceptual mapping rules enable the transformation of BPMN collaboration models into canonical SCA assembly models. The later models describe and guide the final system deployment architecture and its implementation. Thus, the reuse of some existing software applications as well as creating new ones to be reused in other system parts become more evident for the architects. At the execution-time, the assembly models are instantiated for the SCA runtime environments and benefit from those frameworks for deployment and monitoring. Since there is no mean of control flows between the SCA components, then, it seems that some BPMN behavioral aspects have no counterparts in the SCA. However, other semantics can be defined in the SCA components. Therefore, we want to enlarge our approach by transforming other BPMN constructs to non-standard SCA elements for example for the implementation of non-functional requirements.

The experimental results for the BPMN-to-SCA transformation with the ATL are encouraging, and the MDE approach seems very promising, however, it still needs to be validated in a real-scale case studies. The integration of our ATL chain in the Eclipse SOA Tools Platform [18] is in development. Furthermore, the (full) round-tripping between the business processes and the SOA is essential to rapidly realign the IT to accommodate changing business conditions. Consequently, striving for further alignment via service pattern-based techniques, and making SOA development more tractable to reconfigure architectures without disrupting the functional capabilities of the implementations remain as a future work.

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] *Business Process Model and Notation 2.0, Beta 1*, OMG, May 2009.

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

[4] K. Dahman, F. Charoy, and C. Godart, "Generation of Component Based Architecture from Business Processes: Model Driven Engineering for SOA," in *ECOWS 2010 - The 8th IEEE European Conference on Web Services*, Ayia Napa, Greece, 12 2010.

[5] F. Jouault and M. Tisi, "Towards incremental execution of atl transformations," in *Proceedings of the Third international conference on Theory and practice of model transformations*, ser. ICMT'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 123–137.

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

[7] M. Dumas and T. Kohlborn, "Service-enabled process management," in *Handbook on Business Process Management 1*, ser. International Handbooks on Information Systems, P. Bernus, J. Blazewics, G. Schmidt, M. Shaw, J. v. Brocke, and M. Rosemann, Eds. Springer Berlin Heidelberg, 2010, pp. 441–460.

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

[9] W. van der Aalst, A. Mooij, C. Stahl, and K. Wolf, "Service Interaction: Patterns, Formalization, and Analysis," in *Formal Methods for Web Services*, ser. Lecture Notes in Computer Science, M. Bernardo, L. Padovani, and G. Zavattaro, Eds. Berlin, Heidelberg: Springer Berlin / Heidelberg, 2009, vol. 5569, ch. 2, pp. 42–88.

[10] R. M. Dijkman, M. Dumas, and C. Ouyang, "Semantics and analysis of business process models in bpmn," *Information and Software Technology*, vol. 50, no. 12, pp. 1281 – 1294, 2008.

[11] M. Léger, T. Ledoux, and T. Coupaye, "Reliable Dynamic Reconfigurations in a Reflective Component Model," in *Component-Based Software Engineering*, ser. Lecture Notes in Computer Science, L. Grunske, R. Reussner, and F. Plasil, Eds. Berlin, Heidelberg: Springer Berlin / Heidelberg, 2010, vol. 6092, ch. 5, pp. 74–92.

[12] 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.

[13] M. Biehl, "Literature Study on Model Transformations," Royal Institute of Technology, Tech. Rep. ISRN/KTH/MMK/R-10/07-SE, Jul. 2010.

[14] H.-M. Chen, R. Kazman, and O. Perry, "From software architecture analysis to service engineering: An empirical study of methodology development for enterprise soa implementation," *IEEE Trans. Serv. Comput.*, vol. 3, pp. 145–160, April 2010.

[15] K. Levi and A. Arsanjani, "A goal-driven approach to enterprise component identification and specification," *Commun. ACM*, vol. 45, no. 10, pp. 45–52, 2002.

[16] Y. Xiong, D. Liu, Z. Hu, H. Zhao, M. Takeichi, and H. Mei, "Towards automatic model synchronization from model transformations," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, ser. ASE '07. New York, NY, USA: ACM, 2007, pp. 164–173.

[17] P. Stevens, "Bidirectional model transformations in qvt: Semantic issues and open questions," in *MoDELS*, 2007, pp. 1–15.

[18] "Eclipse soa tools platform project," (accessed 29 March 2011). [Online]. Available: http://www.eclipse.org/stp/