Ensuring Business and Service Requirements in Enterprise Mashups

Abstract

During the past few years, mashups have gained wide attention as they utilize Web 2.0 technologies in order to combine data, as well as the functionalities of numerous services, in a simple web application. While developing mashups for simple user-specific needs is not a demanding procedure, this is not the case for value-added services that need to satisfy specific properties and business needs, known as enterprise mashups. As a number of business requirements have to be satisfied, and execution faults are less tolerated compared to user-centric scenarios, a rigorous approach for their development is required. In this work we present such an approach utilizing model checking techniques, provided by the Behavior, Interaction, Priorities (BIP) component framework. In addition, a methodology for the transformation of Business Process Model and Notation (BPMN) models, describing the business logic of a requested mashup, into the corresponding BIP models is proposed. The generated models enable the verification of requested properties.

Keywords: business process; BPMN; enterprise mashups; web services; model checking; BIP component framework.

1 Introduction

Web-based transactions have significantly benefited from the advent of the Service-Oriented Architecture (SOA). More and more enterprises rely on Web Services (WS) in order to fulfill their business needs, both in terms of B2C and B2B transactions. A major advantage that this paradigm offers is the ability to loosely connect heterogeneous systems, which usually results in seamless communications and message exchanges. In addition, the SOA paradigm offers the possibility of WS composition, which enables the rapid development of compelling services, through the merging of WS, which can be handled as composable units.

While SOAP based services are typically composed using BPEL for the orchestration and choreography of services, RESTful compositions are often in the form of Web 2.0 mashups. This term pertains to Web applications or pages that combine data and services from different sources, in order to provide a value-added service, while heavily relying on application programming interfaces (APIs) for the retrieval process. They provide flexibility in creating unified environments, without their construction being significantly effort demanding. Nevertheless, they have mainly been utilized for the creation of simplified projects and user-oriented solutions.

While the abundance of reusable content that can be utilized in order to construct mashups, provide the means for significant business opportunities, such as the low cost development of compelling services, enterprises are generally reliant in reaping those benefits. One of the reasons of this limited interest involves the lack of tools for high-level development based on business and service requirements.

1.1 Business and service requirements

Business requirements refer to the identification and elicitation of the requirements of enterprises, customers and other involved parties in business transactions. They are a primary concern during the development of enterprise information systems and could include the application of execution constraints and the integration of predefined business agreements that should be fulfilled.

Service requirements on the other hand, are requirements that refer both to the functional and non-functional characteristics of the WS that are being used as stand-alone or as a part of a service composition. In more detail functional requirements pertain to those functionalities that are requested by the end-user or the enterprise and focus on the specific operations that the service can perform. Non-functional requirements refer to requirements that are not directly associated to the operations of the service, such as the existence of certain Quality of Service (QoS) characteristics. Such characteristics can play a pivotal role in the selection of a service, between a number of alternatives that provide similar functionalities.

1.2 Definition of emerging mashup types

Mashup types that have recently gained attention and can provide a shift towards the wider adoption of mashups include:

i) Enterprise mashups, which constitute compositions of business-oriented services along with data and information from multiple sources (Pahlke et al. 2010), taking advantage of the conveniences offered by open APIs (Ruhi and Choi 2013). In addition, mashups allow the constant updating of the presented information and content, through the monitoring of incoming events. For example a change in the value of a specific resource (such as a change in the pricing of a specific product) could trigger the instant invocation of another service along with the refreshing of the mashup, in order to keep it up-to-date. These type of mashups provide an important step towards the interoperability between heterogeneous business-oriented systems, a necessity for enterprises, as highlighted in (Zacharewicz et al. 2016).

ii) Physical-Virtual mashups. The advent of the Internet of Things, as well as its extension known as the Web of Things, have given rise to new types of service mashups, which combine functionalities both from traditional WS (virtual services)

and from services based on the functionality of smart devices (physical services). In more detail, Physical-Virtual mashups pertain to the composition of services such as maps, imaging services and business-specific services along with services provided by embedded devices and physical objects (Vesyropoulos and Georgiadis 2013). Typically physical services, allow interactions with end-users or even machine-to-machine (M2M) interactions, through HTTP calls, which give access to the functionality of smart objects, in the form of WS. Value added services can be provided by combining physical and virtual services into a service mashup. An example of such a mashup could be a map, containing information regarding a city's light and humidity measurements in each district.

iii) Business Intelligence mashups, which are web pages or applications that combine an enterprise's internal services along with public and other external WS. In addition, such mashups include the necessary mechanisms that can provide feedback from social media streams, to the enterprise. Such mashups are characterized by their ability to alter or modify the WS involved in a dynamic fashion, in order to respond to the rapidly changing external environment of the enterprise, and to fulfill its needs in the optimal way (Vesyropoulos and Georgiadis 2013).

In this work we attempt to combine the benefits of the aforementioned types, into mashups that include both virtual and physical services while adhering to predefined business rules.

1.3 Motivation

Enterprise mashups comprise of a number of external services, but lack the appropriate mechanisms for their orchestration and the assurances that functional and non-functional properties are satisfied. To tackle this issue, we present a novel approach for the rigorous development of enterprise mashups, by utilizing the BIP component framework. BIP provides a number of powerful model-checking tools that enable the verification of requested properties in complex applications along with a set of execution engines. The framework is analyzed in more detail in a following section. In our proposed methodology we apply a technique for transforming BPMN described scenarios, providing the business logic of a requested composite application, to the corresponding BIP semantics. Thus, the orchestration of involved services and the monitoring of requested properties can be handled by the BIP engine.

The remainder of this paper is organized as follows: section 2 pertains to the presentation of the related work, while in section 3 an introduction to the BIP component framework and to the semantics included is presented. In addition, we elaborate on the application of BIP for the formal definition of architectures. In section 4 we demonstrate the process for transforming BPMN scenarios into BIP components, which can verify the execution order as well as a number of functional and non-functional properties of a requested mashup. Section 5 describes a proposed mashup architecture, while section 6 demonstrates a case study, used as proof of concept for our methodology. Finally, we give some future directions and conclusions stemmed from the application of this modeling approach.

1.4 Business Implications

Businesses require constant communication and interoperation with customers and business partners. Especially, the interoperation between various information systems such as those of suppliers and advertising partners, a vital part of an enterprise's well-being, can be a challenging task. In order to ensure seamless interactions and message exchanges, businesses have often allocated resources into the development of service-based information systems, which utilize well-accepted XML-based communication protocols, while in addition provide WS composition capabilities. Nonetheless, the development and maintenance of such systems requires the constant involvement of an IT department.

By utilizing enterprise mashups, businesses can benefit from the existence of a vast number of publicly available WS whose functionality can be easily combined and

integrated in a Web application or Webpage. Such applications can enable the interoperation with external information systems using various well accepted protocols and file formats, such as HTTP and JSON, accordingly. Such an approach is easier to implement, compared to traditional SOA-based systems, while in addition can lower the costs of an enterprise.

Using the proposed methodology, businesses can provide compelling value-added services to their clients, while limiting the costs required for IT-support, as would be the case in more complex BPEL-oriented service compositions. Since the modeling of business flows in BPMN is also less effort-demanding and their conversion to BIP models can aid the validation of requested properties, businesses can reap the benefits of offering compelling services with low developmental costs and high dependability. Even in scenarios where there is a need of integrating modern IoT-based services, originating from the functionality of devices with embedded sensors, those services can be seamlessly included, as they rely on RESTful calls, constituting them ideal candidates for service mashups.

The enterprise's clients can also benefit from our approach, as it would allow them to have a higher involvement into the composed services and provide them the ability to request specific functional and non-functional characteristics, thus offering a more personalized experience. In addition, as mashups can be in the form of web applications, they are easier to handle and monitor, compared to the more complex SOAP-based compositions.

Finally, software developers can utilize our approach, when developing Web Services, in order to validate specific properties. In more detail, our BIP-based modelling approach and the developed monitor modules can aid in the enforcement of correctness properties, such as lack of deadlocks and interference, during the development of a service, which can also lead to lower testing and software maintenance costs.

2 Related Work

2.1 Mashup Development

Mashup development has been a focal point for researchers for the past few years. A more recent trend, is the research on tools and techniques for the development of business related mashups, as they provide the means for lightweight and on-the-fly value-added solutions that combine services originating from one or more enterprises. In addition, various transformation techniques have also been examined, that convert BPMN models into other model types, in an attempt to automate the composition procedure.

In more detail, authors in (Bozzon et al. 2009) provide a methodology for creating business service mashups, through the transformation of BPMN and WebML flowcharts to rich navigation models. Through the transformation process, an enterprise can acquire enriched models describing their business workflows. This is a promising approach, nonetheless lacks a verification procedure of desired properties, such as the one that can be offered by the BIP component framework. In (Kheldoun, Barkaoui and Ioualalen 2015), a formal model for the verification of complex BPMN models utilizing petri-nets is explained. This approach also takes into consideration the invocation order of BPMN components and formally describes the semantics involved in a BPMN process. Nonetheless, in our approach required properties are both described and embedded into a composite BIP model, which enables the verification of properties into a value-added service mashup.

In (Liu et al. 2011), architecture integration patterns are utilized for a mashup composition, in an approach based on displaying information on map services. This methodology is useful for the rapid development of enterprise mashups, though lacks the means for ensuring business, functional and non-functional properties. In (Xue et al. 2013), an integrated framework for personalized business processes is demonstrated. The framework supports the modeling and monitoring of numerous business processes in an enterprise mashup and in addition allows the automated

execution of the overall composed service. As the focal point of this research is the group of users with limited programming skills that need to compose a mashup based on business processes, no measures are proposed for the validation of business-oriented properties. In (Xu 2013), a modeling approach is described centered on the process collaboration of end-users. This proposal adopts BPMN notions in order to describe control and data flows in enterprise mashups, and relies on the correct use of synchronization patterns on behalf of the end-users, in order to model the behavior of value-added systems. While currently limited validation capabilities are offered, the authors propose the use of a process modeler, based on the aforementioned approach, which will assist users in the modeling process. This is a promising solution, as it could enable the enforcement of desired properties.

In (Hobel et al. 2013), the use of a platform for enterprise mashups is presented, in which security concerns are of the highest priority. The platform enables the definition of security rules and the monitoring of submitted mashups in order to detect security threats, through the evaluation of compliance with predefined policies. As mashups developed on-the-fly by customers can handle sensitive data, the need for imposing security rules and monitoring the composed complex process is considered mandatory, giving value to this research approach.

2.2 Application of model checking and formal methods in virtual and physical WS

In (Said et al. 2015), BIP has been used in order to secure WS compositions, by ensuring the non-interference property. In addition an approach for the validation of time requirements utilizing BIP have been proposed in (Guermouche and Dal Zilio 2012). Other approaches regarding the formal modeling and analysis of WS-based applications are described below.

In (Kil and Nam 2013) authors apply three individual model-checking techniques in order to identify optimal service compositions, though their applicability is limited in scenarios that lack semantic descriptions for the involved WS. A method for conflict detection in value-added services, based on model-checking, is presented in (Kim et al. 2013). This is a promising approach, while differentiated to ours, as BIP utilizes property-preserving transformations which allow the verification of non-interference and deadlock-freedom in composite models (Sifakis 2014).

While the development of RESTful applications can be assisted by REST-related frameworks, limited support is provided for the validation of adherence to the REST constraints and service profiles. In order to overcome this, formal methods have been used in literature, such as the work presented in (Decker et al. 2009), where authors use Petri-nets in order to formally describe the execution of RESTful processes. REST constraints, such as the uniform interface requirement, were also included in the proposed model. In addition, in (Wu et al. 2012), the REST architecture is formally modeled using the CSP process algebra. In this manuscript CSP processes are used to describe RESTful components while the constraints introduced in the REST paradigm, are also being modeled. Through model checking, the authors proceed to the validation of the client-server and cashable constrains.

Similarly to RESTful projects, formal methods are often applied in order to validate specific properties of IoT applications. Such applications are often being developed by utilizing a number of available domain-specific operating systems, such as the Contiki OS and Riot. As constrained devices are used in IoT scenarios, more and more efforts are made to optimize the development procedure. In (Glombitza et al. 2010) FSM are applied as a means for a model-driven approach in developing WSN service applications. In addition a domain specific language is introduced. Developed applications can be utilized in IoT scenarios due to their service-based nature. Finally in (Vörtler et al. 2015), authors introduce a verification framework for Contiki-based IoT applications. This approach relies on CBMC, a model checking tool and the corresponding modeling of the interrupts that occur in the Contiki OS. The authors focus on the verification of IoT applications, while limitations occur especially in cases where a large number of interrupts have been modeled.

3 The BIP component framework

Our suggested approach focuses on enabling a rigorous design of Enterprise Mashups, using the BIP component framework and its associated set of tools. BIP provides the means for designing and validating composite applications, through the utilization of its component-based architecture and rigorous semantics. While initially developed for heterogeneous embedded applications, it has been successfully applied on numerous fields. The semantics of the BIP engine are formally described in detail below, based on the work presented in (Basu et al. 2013).

Through BIP models, the *behavior* of complex systems can be described in hierarchically structured models, which consist of a set of atomic components, along with their corresponding interactions and interfaces. BIP atomic components are transition systems extended with a set of ports and variables. Atomic components AC, are in the form of a tuple (Q, X, P, T), where Q defines the control locations, X the set of variables, P the communication ports and T the available transitions. When transitions occur, the variables that describe characteristics of the modeled system can receive new data. Such transitions τ are represented as (q, p, g, f, q'). In τ the variables $q, q' \in Q$ pertain to control locations and in more detail to the initial and the new location after the transition. The variables p and g correspond to a port and guard value, while f is the user defined function that controls the transition and updates the variable values. This is the case as the BIP framework allows the execution of C and C++ code, in the form of functions, during transitions.

While the transitions model the *behavior* of the atomic components, *interactions* model the necessary synchronization between such components and the corresponding data exchanges. Such interactions are handled by connectors, which define the synchronization rules of the corresponding ports. The BIP semantics define two types of ports: i) *synchrons*, where transitions can take place only if all connected components can simultaneously perform the corresponding transitions or ii) *trigger ports*, which enable a broadcast-based synchronization, as all possible interactions which include the transition of the trigger port can be executed. An interaction a is defined as the triple (P_a, G_a, F_a) where these values correspond to ports utilized in an interaction, the guard value and the specific function that is enabled and executed during the interaction.

In addition, *priorities* refer to the scheduling of the modeled system, which can be accomplished through the definition of rendezvouses between a set of ports. In a set of γ interactions a priority π signifies a partial order $\pi \subseteq \gamma \, x \, \gamma$, where $\alpha_1 \, \pi \, \alpha_2$ demonstrates a higher priority in interaction α_1 compared to interaction α_2.

To conclude, BIP enables the composition of atomic components into composite ones $CC_i = \{(Q, X, P, T)\}_{i=1}^{n}$ where we assume that for each given two AC the joint of ports and the joint of variables equals to zero. By constructing architectures in BIP, thus applying formal specifications, it is possible to enforce and to compose properties while developing composite components. Property enforcement is achieved by applying restrictions to the behavior of a modeled architecture. Property composability, which pertains to the combination of architectures, is accomplished by formally defining the components that the architecture is consisted of, along with a glue operator (Sifakis 2014).

When composing architectures, an important goal is to maintain a global property. As shown in (Bliudze and Sifakis 2008), utilizing glue operators in component compositions enables the preservation of the properties enforced in the standalone components. In (Lekidis et al. 2015) formal architectures that enable the rigorous development of physical services are presented, while in (Stachtiari et al. 2014), virtual service architectures (SOAP-based and RESTful) are deployed, aiming at the enforcement of requested properties. In addition, an elaboration on the utilization of BIP for the composition of service architectures is also being presented.

4 Transformation of BPMN models

Complex business-related requirements, as well as the execution order of the involving services in a business scenario, are often modeled using BPMN due to its provided

graphical notations. BPMN 2.0 provides the means for the serialization of BPMN model and the extraction of the corresponding XML schemes.

In this work, a methodology is presented for the rigorous development of enterprise mashups, which adhere to business-related requirements that have been documented as BPMN processes.

4.1 BPMN components

A BPMN process can consist of the following basic elements:

i) Flow objects: including events, activities and gateways

ii) Connecting objects: which can be sequence flows, message flows and associations

iii) Swim lanes: divided to pools and lanes

iv) Artifacts: such as data objects, groups and annotations.

A formal definition of the core elements of a BPMN process include:

<bpmn_process>	::=	<pool>*
<pool>	::=	<lane>*
<lane>	::=	<event>* <activit>* var_def* <gateway> <connect>
<event>	::=	start_ev | inter_ev | end_ev
<activit>	::=	task | subprocess | transctn | call_act
<gateway>	::=	excl | ev_based | parll | incls | exlc_ev | exl_ev | incls_ev | comlx | parll
<connect>	::=	seq | msg_fl | association

In more detail, a process contains the declaration of variables, the definition of events and the included activities, gateways and connections. Events are handled by the corresponding handlers and can be of three types: starting, intermediate and ending. Activities included in a process can be tasks, subprocesses, transactions and activity calls. A number of different types of gateways are defined in BPMN such as exclusive, event based and parallel gateways while finally connections can be sequential, message flow connections or associations.

4.2 BPMN to BIP transformation

In order to achieve the requested business requirements we translate BPMN models to their corresponding BIP models, based on a predefined set of rules explained below, utilizing a developed transformation tool. Model to model mappings are often applied in business scenarios, when strict properties need to be ensured (Wang, Truptil and Benaben 2016). The proposed transformation of an XML-based language to a corresponding BIP model is a feasible process, as presented in (Stachtiari et al. 2012). We have selected the following rules for the transformation procedure:

i) Pools are transformed into composite BIP modules, while lanes are components of a module.

ii) The communication between components, or even between modules, is achieved through the utilization of ports.

iii) While serial executions in a lane are converted to serial states of a module, parallel executions (originating from the existence of gateways) result to components executing simultaneously and synchronizing through rendezvous.

iv) Finally, restrictions in the invocation order of components in the BPMN model can be handled by the priorities declared in the corresponding BIP model. To enable the monitoring of specific properties, appropriate modules can also be integrated into the generated model, as will be shown in the section 5.3.

The transformation process is demonstrated below, through a low-scale transformation example, in the context of e-commerce transactions. Suppose an e-commerce business provides pick-up spots, from where the customer can collect his ordered products, thus avoiding extra shipping costs. Each time an order is received the corresponding pick-up spot handles the overall procedure. A typical order procedure is depicted in the BPMN model presented in figure 1. As seen in the model, upon receiving the order, the pick-up spot sends a query to the warehouse, in order to ensure that the products are available and, most importantly, in good condition. The warehouse responds to the query accordingly. In case one of the requested products is damaged and no other stock is available, the order is cancelled.

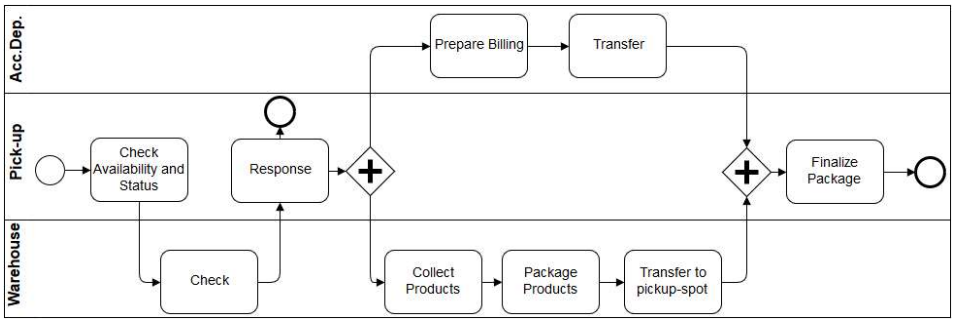

Figure 1 Order procedure in BPMN

If no problems were encountered, two procedures can be initiated in parallel. The warehouse is ordered to collect and package the products and to transfer them to the pick-up spot, while the accountings department can prepare the necessary paperwork. After both processes are completed, the pick-up spot can finalize the order.

The corresponding BIP model, created following the aforementioned rules, is presented in figure 2. As seen in the depicted model, the three lanes of the BPMN scenario produce the same number of atomic BIP components.

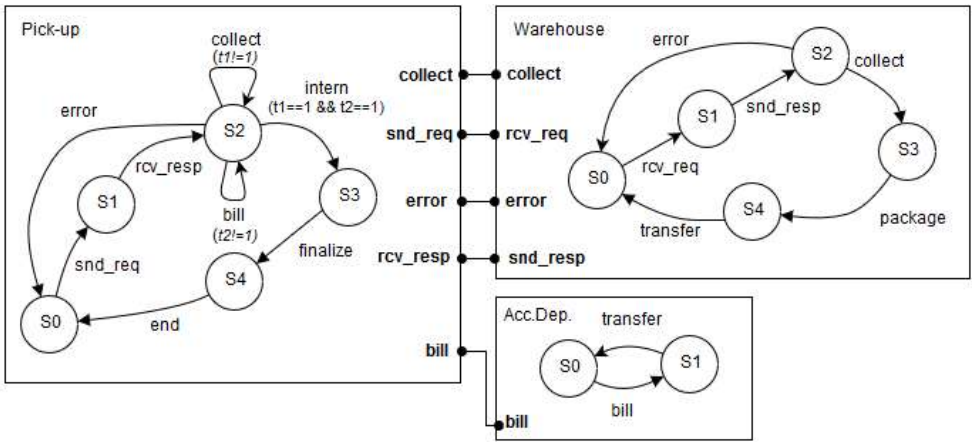

Figure 2 Order procedure in BIP

For communication purposes between these components, a total of ten ports is being deployed. Utilizing these ports the simultaneous execution of parallel transitions is also ensured, as components synchronize through rendezvous. In addition, the invocation order of the BPMN components is maintained, as it is controlled by the declared priorities in the BIP model.

The model can be checked for correctness and ensures that business requirements are satisfied even when the requested behaviors are handled by WS in a complex

value-added service, such as a mashup. In the table 1 the transitions related to the external ports are explained.

Table 1 Explanation of ports in atomic components

Pick-up	
snd_req	sends a request to the warehouse regarding the products
error	receives a detected error regarding availability or status
rcv_resp	receives response regarding the availability and status
collect	issues a collect items request
bill	issues a billing request
Warehouse	
rcv_req	receives an availability and status examination request
error	sends a detected error regarding availability or status
snd_resp	sends a response regarding availability and status
collect	receives a collect items request
Accounting Department	
bill	receives a billing request

5 Mashup Model

In this section, we present a reference architecture for enterprise mashups. The architecture A_{MASHUP} consists of the application, core and service layers, as depicted in figure 3.

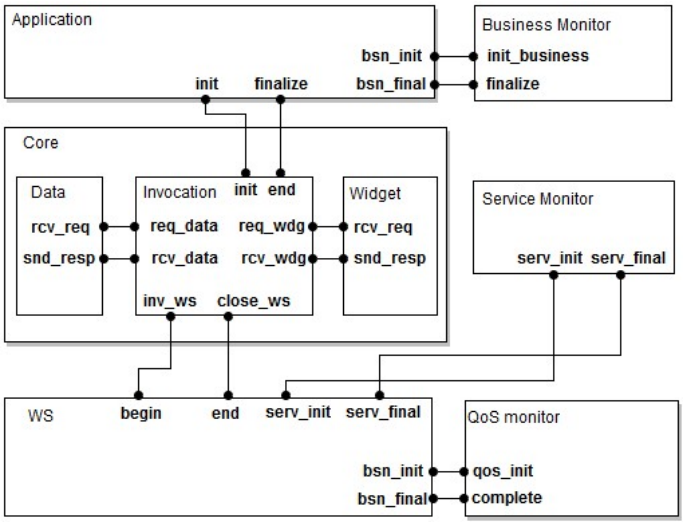

Figure 3 The abstract A_{MASHUP} architecture

The application layer refers to the model originating from the transformation of the BPMN model to the corresponding BIP module. The required business logic is included in this layer and as a result the modeled behavior and interactions are closely related to the needs of the given transactional scenario.

The core layer consists of the module responsible for the *invocation* of requested components, the *widget* module which is responsible for collecting and parsing webpages and the *data* module which collects data from external sources in XML or JSON format.

Finally, the service layer is responsible for the interoperations with the requested internal and external WS. This layer enables the examination of architecture specific properties, such as RESTful architectural constraints. In the overall architecture model, monitoring modules are also included as a means for the verification of business, service and QoS requirements. For the verification of business requirements the corresponding module is connected with appropriate ports to the application layer,

where the business workflow is modeled. The modules that examine functional and nonfuctional characteristics of the involved services are connected to the WS layer. More information regarding the architecture's port are given in table 2.

Table 2 Definition of the architecture's ports

Application	
init	Signals the initiation of the mashup, according to the business logic of the application
finalize	Signals the finalization of the mashup
bsn_init	Initializes the business monitor for observing the possible violation of predefined business rules
bsn_final	Signifies that the business monitor finished its without errors
Core	
Data	
rcv_req	Receives a request for parsing of an external source's data
snd_resp	Returns the data to be included in the mashup
Invocation	
init	Receives a request for the initiation of a mashup
end	Ends the mashups operation
req_data	Requests the parsing of external data
rcv_data	Receives a response from the data module
req_wdg	Requests data in the form of a widget
rcv_wdg	Receives a response from the widget module
inv_ws	Requests the invocation of a specific WS operation
close_ws	Receives a signal regarding the termination of an operation
Widget	
rcv_req	Receives a request for including external information, without modifications, in the form of a widget
snd_resp	Returns data in an appropriate format for a widget
WS	
begin	Receives a request for initialization and commences its operation
end	The WS terminates its operation
serv_init	Initializes the service monitor for observing the possible violation of service properties in a specific WS
serv_final	Signifies that the service monitor finished its without errors
bsn_init	Initializes the business monitor for observing the possible violation of predefined business rules
bsn_final	Signifies that the business monitor finished its without errors
Business monitor	
init_business	Signifies the initialization of the business monitor for observing the possible violation of predefined business rules
finalize	Finalizes the business monitor without errors, as if such an event occurred a deadlock would be detected by the BIP engine
Service monitor	
serv_init	Signifies the initialization of the service monitor for observing the possible violation of service properties in a specific WS
serv_final	Finalizes the service monitor without errors, as if such an event occurred a deadlock would be detected by the BIP engine
QoS monitor	
qos_init	Initializes the QoS monitor for observing the possible violation of a predefined non-functional characteristic of a WS
complete	Finalizes the business monitor without errors, as if such an event occurred a deadlock would be detected by the BIP engine

In case no specific data parsing or widget support is required in a given business transaction scenario, the following abstract architecture can be simplified as seen in the case study in section 6 where we focus on the application layer and the monitoring modules integration. While the application layer can be utilized as a standalone component for the verification of specific properties, benefits can rise from embedding it to the overall architecture model. Such an integration enables the validation of additional properties (e.g. architectural properties through the service layer) and the handling of data by the BIP engine (through the core layer).

While the development of BIP models requires some basic modeling knowledge, an important aspect of the proposed architecture is that the core and service layers, as well as the QoS monitor module, only need to be modeled once. When modeling a new business scenario, the only modules that need to be developed are the application module (as it is based on the required business logic) and the business and service monitors, that can examine application specific properties.

6 Case Study – An e-commerce transaction

As a proof-of-concept regarding our mashup development workflow, we present a BIP model describing both B2B and B2C transactions of a flower e-shop. The business logic of this case study is based on a simple execution flow, described in BPMN, which involves service interactions that occur in a typical e-commerce transaction.

After the conversion process, the corresponding BIP model is checked for behavioral correctness, concurrency issues and for the violation of non-functional requirements, through the integration of monitor modules. The mashup should be able to handle the presentation of products, the ordering mechanics, the preparation of the delivery as well as the calculation of the overall costs and the printing of the invoice, as shown below in figure 4.

6.1 Services Involved

The individual services involved are presented in more detail below:

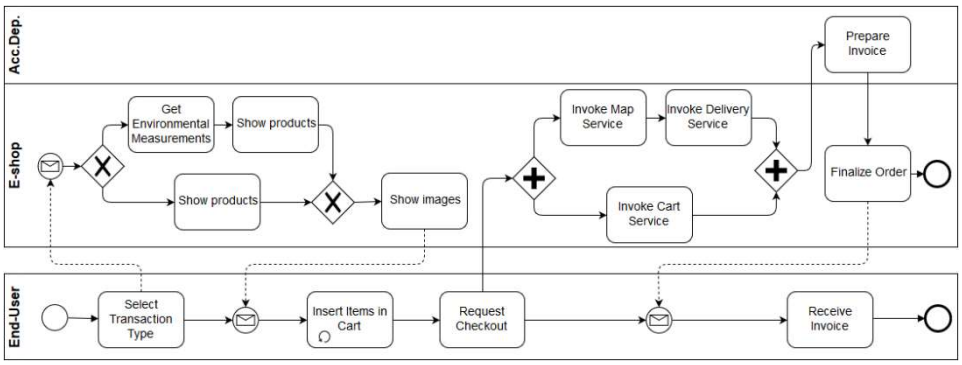

Figure 4 BPMN model describing the execution flow

i) Show products: Presents product description and characteristics. In addition, user ratings and reviews can be included in the overall presentation provided. This service exchanges information with the *show images service* for the projection of images regarding the selected products, which are stored in external image repositories (e.g. in Flickr)

ii) Cart service: The cart service is responsible for the handling of orders. The ordered products list is being transferred in message exchanges and restrictions are applied. Such a restriction is that a product cannot mistakenly be ordered twice.

iii) Invoice service: The invoice operation is responsible for issuing the invoice that includes all the individual charges. This operation needs to exchange information with other services, including the *cart* and *delivery* service, before it can be initiated.

iv) Map service: Is used for the identification of the customer's shipping address and can be one of the popular alternatives offered, such as the Google Maps service offered by the corresponding API.

v) Delivery: The delivery service is responsible for the selection of the appropriate vendor for delivering the requested package to the customer. This service must exchange information with other services, such as the *map* service and the *invoice* service.

vi) Show images: Presents images of products and can exchange information with the show products service. As mentioned above, images are stored in image repositories.

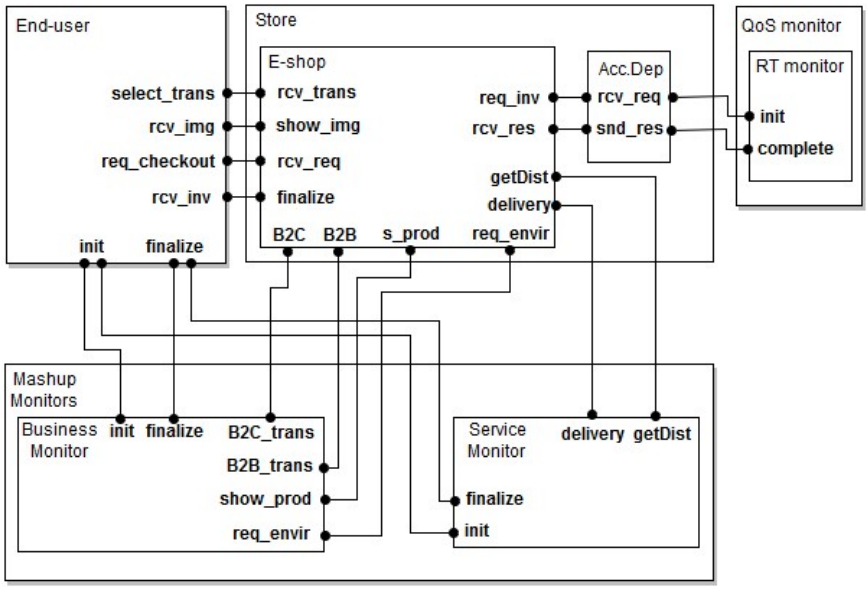

Figure 5 The corresponding abstract BIP model with monitor integration

In case a B2B transaction is initiated the environmental measurement service is also invoked:

vii) Ambient (environmental) measurements service: Physical services enable the usage of the functional characteristics of smart objects, as on-demand services. As flowers are delicate products, in this case study we suppose that the members involved in a B2B transaction, request additional information regarding environmental measurements in order to evaluate the current status of requested products. In more detail the temperature, humidity and light measurements can be returned. This is accomplished by invoking the corresponding sensors and handling their measurements as RESTful resources. For the needs of this case study we have developed this service using the Contiki OS, while the RIOT OS can also be utilized (Isikdag 2015). The integration of this service to the application, demonstrates the feasibility of applying the proposed methodology to validate required properties using both virtual and physical services.

The generated abstract BIP model is presented in figure 5, while the definition of included ports is presented in table 2:

Table 3 Definition of utilized ports

End-user	
select_trans	provides information for the selected transaction type
rcv_img	receives a list of product images
req_checkout	issues a request for the purchase of selected products
rcv_inv	receives the order's invoice
init	signals the initialization of the procedure

finalize	signals the finalization of the procedure
E-shop	
rcv_trans	receives information on the requested transaction type
show_img	sends a list of product's images
rcv_req	receives a request for the purchase of selected products
finalize	informs the user the order is finalized
req_inv	requests the order's invoice
req_resp	receives the order's invoice
B2B	signals the initiation of a B2B transaction
B2C	signals the initiation of a B2C transaction
s_prod	starts the execution of the show products service
req_envir	sends request for environmental measurements
getDist	sends request for the client's distance from the e-shop
delivery	signals the execution of the delivery service
Accounting Department	
rcv_req	receives an invoice request
snd_resp	sends the corresponding invoice
RT monitor	
init	synchronizes with the initiation of the invoice service
complete	synchronizes with the completion of the invoice service
Business monitor	
init	signals the initialization of the procedure
finalize	signals the finalization of the procedure
B2C_trans	receives the initiation event of a B2B transaction
B2B_trans	receives the initiation event of a B2C transaction
show_prod	synchronizes with the execution of the show products service
req_envir	synchronizes with the request for environmental measurements
Service monitor	
init	receives the initiation event of the procedure
finalize	receives the finalization event of the procedure
delivery	synchronizes with the execution of the delivery service
getDist	synchronizes with the request for the client's distance from the e-shop

6.2 Business and service properties

We selected a number of service and business properties (SP and BP accordingly), that can be validated through the generated BIP model. We have included both functional (SP1) and non-functional (SP2) requirements.

SP1: The *Delivery* service must receive as input the distance (as calculated by the *Map* service, based on the information provided by the store) and then calculate and return the delivery cost.

SP2: Involved services are reliable. With the term reliable, we refer to WS tasks that can be invoked on demand and complete their functions in a reasonable time frame and without interferences. While reliability is also heavily linked with the correct transmission of messages, we solely focus on the completion of tasks as described in (Cardoso et al., 2004). This is the case, as the nature of idempotent actions in RESTful services can lead to the prevention of message-based errors.

BP1: The *Ambient measurements* service can only be initiated in B2B transactions.

BP2: The *Show products* service must receive information, regarding the nature of the transaction, on invocation and before the presentation of products and prices, as those differ in B2B and B2C scenarios.

6.3 Evaluation

By converting the BPMN model to the corresponding BIP model, we can ensure that invocation order of the involved services, originally described in the BPMN model is preserved, aiding the fulfillment of requested properties. Nonetheless, in order to monitor complex transactions, when more services are included and more message exchanges occur, and to ensure that all requested requirements are satisfied, atomic monitor components can be applied. Such an approach has been presented in (Stachtiari et al. 2012)

Figure 6 presents a proposed business monitor component for the verification of properties BP1 and BP2. The component terminates the overall procedure if B2B transaction occurs in a B2C scenario.

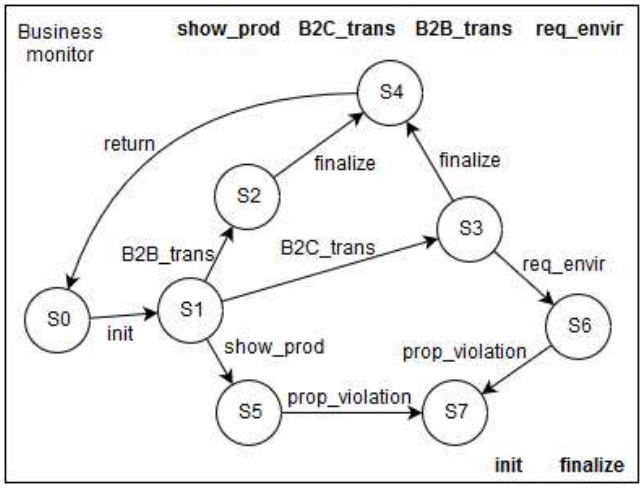

Figure 6 The developed business monitor

Aiming to validate property SP1, we have developed the service monitor component presented in figure 7. The component reaches a deadlock in case a service does not complete its requested functionality.

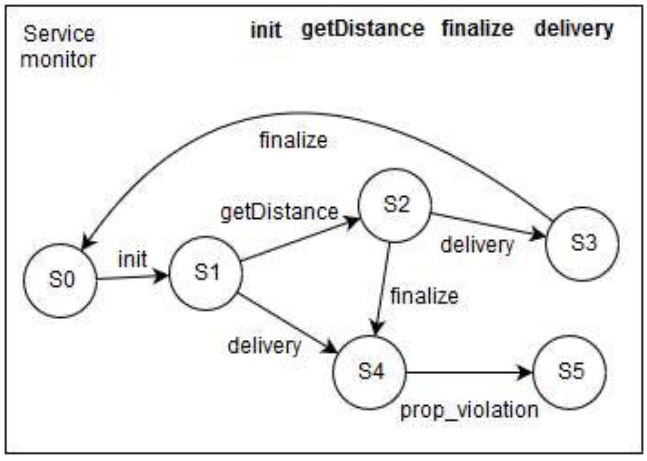

Figure 7 The developed service monitor

Using the tools provided by the BIP component framework, we examined all execution seeds of the mashup model to validate the fulfillments of requested requirements and verify the deadlock freedom property. As seen in figure 8, the validation of the model provides insight on the satisfaction of requested properties. In the depicted state, the mashup model has two monitoring modules integrated, the business and service monitor.

```
tem --explore
[BIP ENGINE]: BIP Engine (version 2015.04-RC7 )
[BIP ENGINE]:
[BIP ENGINE]: initialize components...
[BIP ENGINE]: computing reachable states:.. found 33 reachable states,
0 deadlock, and 0 error in 0 state
```

Figure 8 State exploration of the generated BIP model

In case an erroneous transition, which enables the invocation of the delivery service prior to the map service, exist or the execution of the involved services is not restricted by the BIP tool, this would result into a violation detected by the service monitoring module.

```
[BIP ENGINE]: state #11: 1 interaction:
[BIP ENGINE]:   [0] ROOT.customer.reqCheckout
[BIP ENGINE]: -> choose [0] ROOT.customer.reqCheckout
[BIP ENGINE]: state #12: 2 interactions:
[BIP ENGINE]:   [0] ROOT.eshop.getDistance
[BIP ENGINE]:   [1] ROOT.eshop.delivery
[BIP ENGINE]: -> choose [0] ROOT.eshop.delivery
[BIP ENGINE]: state #13: deadlock!
```

Figure 9 Detected deadlock signifying property violation

The same would apply in transitions that are observed by the business monitor. For example, the invocation of the environmental measurements service in a B2C scenario would result in a violation detected by the aforementioned monitor.

Finally, in order to evaluate the non-functional service property SP2, regarding the QoS property reliability, we have developed the monitor module depicted in figure 10.

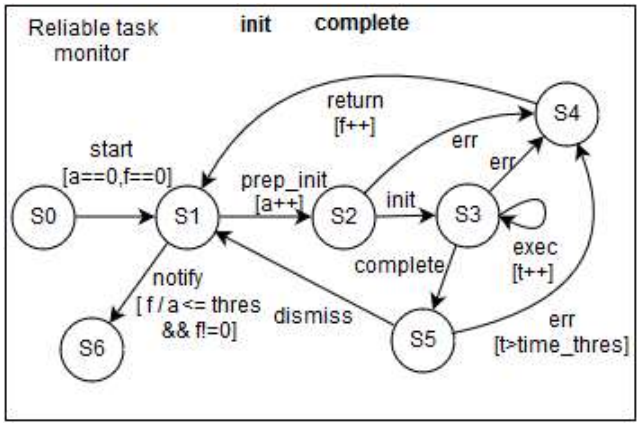

Figure 10 The reliability monitor

The monitor counts the overall attempts to execute a specific service, as well as the number of recorder failures during its invocation or execution. In our scenario, the reliable task monitor is connected to the invoice service. Two threshold values are used, one pertaining to the execution time of a service and another related to the ratio of the overall invocation attempts to those that were not successful. If the reliability property is not fulfilled the monitor returns a corresponding message.

Regarding this property, we have executed a number of REST APIs publicly available in the programmable web site (www.programmableweb.com), for handling invoices. In more detail, we took a random sample of 25% of the available invoice services that the corresponding search query returned (approximately 51 out of 204 services). Using strict values for the time threshold and the failures to attempts ratio threshold, there were occurrences where the reliability monitor returned a violation message, as seen in figure 11 and in table 4.

```
[BIP ENGINE]: state #25: 1 internal port:
[BIP ENGINE]:   [0] ROOT.QoSReliable.return
[BIP ENGINE]: -> choose [0] ROOT.QoSReliable.return
[BIP ENGINE]: state #26: 1 internal port:
[BIP ENGINE]:   [0] ROOT.QoSReliable.notify
[BIP ENGINE]: -> choose [0] ROOT.QoSReliable.notify
[BIP ENGINE]: state #27: deadlock!
```

Figure 11 Detected deadlock triggered by the reliability monitor

By loosening those threshold values, which is equivalent to lowering the required QoS values by the end-user, the BIP model can be validated and no deadlock occurs as can be seen in figure 12.

Table 4 Results of the alteration of threshold values

Alterations attempt	Time threshold	Failures to attempts ratio	Detected deadlocks
1	0.5s	0.1	29.41%
2	0.7s	0.2	25.49%
3	0.9s	0.3	19.61%
4	1.2s	0.4	19.61%
5	1.5s	0.5	11.74%
6	1.8s	0.6	5.88%
7	2.1s	0.7	1.96%
8	2.4s	0.8	1.96%

This can be used as a mean to validate the accuracy and trueness of the model. In the following figure all three developed monitoring modules are integrated.

```
tem --explore
[BIP ENGINE]: BIP Engine (version 2015.04-RC7 )
[BIP ENGINE]:
[BIP ENGINE]: initialize components...
[BIP ENGINE]: computing reachable states:.. found 40 reachable states,
0 deadlock, and 0 error in 0 state
```

Figure 12 Deadlock freedom as a result of threshold adjustment

7 Conclusion, business implications and future work

We have demonstrated a rigorous approach for the development of enterprise mashups, based both on virtual as well as on physical services. Based on the BIP component framework, we provide a mean to enforce business requirements as well as functional and non-functional properties. Through the proposed design flow an enterprise mashup model is demonstrated as a proof-of-concept, while the services that comprise it, have been examined for the correctness of required properties. Our contribution's focal point is the rigorous methodology for developing and testing Web mashups, as there is a lack of such approaches in recent literature.

Enterprises could use the proposed methodology to model their workflows using the well-known BPMN format and easily transform them into formal BIP based models, thus ensuring that the value-added services they provide to their clients and business partners maintain predefined business and service properties. The process of developing BPMN models is significantly less challenging a task as it creating the corresponding BPEL models, along with the necessary orchestration and choreography rules that are typically required in SOAP-based service compositions. In addition, through the automation of the BPMN to BIP transformation process, highly reliable enterprise mashups can be easily constructed. As a result, businesses can benefit from the adoption of our approach, since it can ease the workload of their IT department,

lower the cost of developing and maintaining complex compositions and enable the rapid development of dependable value-added services. Furthermore, enterprise mashups enables clients to receive more personalized services, as they can be more involved into the development process.

Future work involve the enrichment of the automated tool for the transformation of BPMN model to the corresponding BIP modules as well as the integration of SOAP-based services into enterprise mashups. Finally we aim at the extended monitoring of the overall value-added service, through the dynamic validation of additional QoS requirements on run-time, as they constitute an important factor in the selection of WS and highly influence the overall satisfaction level of end-users. Finally, due to the fact that when evaluating the reliability of a service, the execution time needed is of vital importance, we aim to enhance the reliability task monitor. To achieve this we aim at utilizing a differentiated version of BIP (timed-BIP), which promises higher precision in time measurements.

References

Basu, A., Bensalem, S., Bozga, M., Bourgos, P., Maheshwari, M., Sifakis, J (2013) Component Assemblies in the Context of Manycore. In: Beckert, B. et al (eds.) Lecture Notes in Computer Science. Springer, Berlin Heidelberg, pp 314–333

Bliudze, S., Sifakis, J (2008) A notion of glue expressiveness for component-based systems. In: Proc. 20th International Conference on Concurrency Theory (CONCUR), pp 508-522

Bozzon, A., Brambilla, M., Facca, F. M., Carughu, G. T (2009) A conceptual modeling approach to business service mashup development. In: Proc. IEEE International Conference on Web Services, (ICWS), pp 751-758. doi: 10.1109/ICWS.2009.24.

Cardoso, J., Sheth, A., Miller, J., Arnold, J., Kochut, K (2004) Quality of service for workflows and web service processes. Web Semantics: Science, Services and Agents on the World Wide Web. 1: 281-308. doi: 10.1016/j.websem.2004.03.001

Decker, G., Lüders, A., Overdick, H., Schlichting, K., Weske, M (2009) RESTful Petri Net Execution. In: Bruni, R., Wolf, K. (eds.) Web Services and Formal Methods. Springer-Verlag, pp 73-87

Glombitza, N., Pfisterer, D., Fischer, S (2010) Using state machines for a model driven development of web service-based sensor network applications. In: Proc. ACM Workshop on Software Engineering for Sensor Network Applications (ICSE), pp 2-7

Guermouche, N., Dal Zilio, S (2012) Towards timed requirement verification for service choreographies. In: Proc. 8th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), pp 117-126

Hobel, H., Heurix, J., Anjomshoaa, A., Weippl, E (2013) Towards security-enhanced and privacy-preserving mashup compositions. In: Janczewski, J. et al (eds.) Security and Privacy Protection in Information Processing Systems. Springer, Berlin Heidelberg, pp 286-299

Isikdag, U (2015) Internet of Things: Software Platforms. In: Enhanced Building Information Models, pp. 55-70. Springer International Publishing

Kheldoun, A., Barkaoui, K., Ioualalen, M (2015) Specification and Verification of Complex Business Processes - A High-Level Petri Net-Based Approach. In: Motahari-Nezhad, R.H., Recker, J., Weidlich, M. (eds.) Business Process Management. Springer International Publishing, pp 55-71

Kil, H., Nam, W (2013) Semantic web service composition via model checking techniques. International Journal of Web and Grid Services. 9: 339-350. doi: http://dx.doi.org/10.1504/IJWGS.2013.057466

Kim, Y. S., Shin, D. H., Jeon, H. B., Lee, K. H., Cho, K. S., Park, W (2013) Conflict detection in composite web services based on model checking. International Journal of Web and Grid Services. 9: 394-430. doi: http://dx.doi.org/10.1504/IJWGS.2013.057470

Lekidis, A., Stachtiari, E., Katsaros, P., Bozga, M., Georgiadis, C. K (2015) Using BIP to reinforce correctness of resource-constrained IoT applications. In: Proc. 10th IEEE International Symposium on Industrial Embedded Systems (SIES), pp 1-10

Liu, Y., Liang, X., Xu, L., Staples, M., Zhu, L (2011) Composing enterprise mashup components and services using architecture integration patterns. J. Syst. Software. 84: 1436-1446. doi:10.1016/j.jss.2011.01.030

Pahlke, I., Beck, R., Wolf, M (2010) Enterprise mashup systems as platform for situational applications. Business & Information Systems Engineering, 2, 305-315. doi: 10.1007/s12599-010-0121-9

Ruhi, U, Choi, D (2013) Enterprise Mashups for Knowledge Management. In: Proc. 1st International Conference on Information and Communication Technology Trends (ICICTT), pp 159-168

Said, N. B., Abdellatif, T., Bensalem, S., Bozga, M (2016) A Robust Framework for Securing Composed Web Services. In: Braga, C., Ölveczky, P.C. (eds.) Formal Aspects of Component Software. Springer International Publishing, pp 105-122

Sifakis, J (2014) Rigorous system design. In: Proc. Symposium on Principles of distributed computing, pp 292-292. ACM

Stachtiari, E., Mentis, A., Katsaros, P (2012) Rigorous analysis of service composability by embedding WS-BPEL into the BIP component framework. In: Proc. 19th IEEE International Conference on Web Services (ICWS), pp 319-326

Stachtiari, E., Vesyropoulos, N., Kourouleas, G., Georgiadis, C. K., Katsaros, P (2014) Correct-by-Construction Web Service Architecture. In Proc. 8th IEEE International Symposium on Service Oriented System Engineering (SOSE), pp 47-58

Vesyropoulos, N., Georgiadis, C. K (2013) Web of things: understanding the growing opportunities for business transactions. In: Proc. 6th Balkan Conference in Informatics, pp 267-274

Vörtler, T., Höckner, B., Hofstedt, P., Klotz, T (2015) Formal Verification of Software for the Contiki Operating System Considering Interrupts. In: 18th IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS), pp 295-298

Wang, T., Truptil, S., Benaben, F (2016) An automatic model-to-model mapping and transformation methodology to serve model-based systems engineering. Information Systems and e-Business Management, 1-54. doi: 10.1007/s10257-016-0321-z

Wu, X., Zhang, Y., Zhu, H., Zhao, Y., Sun, Z., Liu, P (2012) Formal modeling and analysis of the REST architecture using CSP. In: Beek, M., Lohmann, N. (eds.) Web Services and Formal Methods. Springer-Verlag, pp 87–102

Xu, L., de Vrieze, P., Phalp, K., Jeary, S., Liang, P (2013) Interoperative end-user process modelling for process collaborative manufacturing. International Journal of Computer Integrated Manufacturing. 26: 990-1002. doi: 10.1080/0951192X.2012.685107

Xue, S., Wu, B., Chen, J (2013) An End-User Oriented Approach for Business Process Personalization from Multiple Sources. In: Aditya Ghose et al. (eds.) Service-Oriented Computing–ICSOC 2012 Workshops. Springer-Verlag Berlin Heidelberg, pp 87-98

Zacharewicz, G., Diallo, S., Ducq, Y., Agostinho, C., Jardim-Goncalves, R., Bazoun, H., Wang, Z., Doumeingts, G (2016) Model-based approaches for interoperability of next generation enterprise information systems: state of the art and future challenges. Information Systems and e-Business Management, 1-28. doi: 10.1007/s10257-016-0317-8