Inf Syst E-Bus Manage
DOI 10.1007/s10257-017-0363-x

ORIGINAL ARTICLE

Ensuring business and service requirements in enterprise mashups

Nikolaos Vesyropoulos[1] · **Christos K. Georgiadis**[1] · **Panagiotis Katsaros**[2]

Received: 22 November 2016 / Revised: 24 July 2017 / Accepted: 1 September 2017

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 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

✉ Nikolaos Vesyropoulos
nvesyrop@uom.edu.gr

Christos K. Georgiadis
geor@uom.edu.gr

Panagiotis Katsaros
katsaros@csd.auth.gr

[1] University of Macedonia, 156 Egnatia Street, 54006 Thessaloníki, Greece

[2] Aristotle University of Thessaloniki, Thessaloníki, Greece

Published online: 06 September 2017

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, such as enhanced maps with embedded information regarding the location of certain points of interest. These mashups are produced by combining a map service (e.g. the Google maps API) with data extracted from a secondary source, such as, for example, a listing of available museums in a specific area. Such a solution can aid an end-user in retrieving information, rapidly, through a unified visual representation, though has limited application value in a business context. A more compelling project is described in Polymerou et al. (2014), where a combination of a sentiment analysis service, a map service and a video repository can provide an interactive map that reveals the emotions that a video can invoke in different regions, by processing the users' comments in the video's corresponding section. Through the utilization of this mashup, peoples' attitudes regarding specific issues can be examined.

The abundance of reusable content that can be utilized in order to construct mashups, provides the means for significant business opportunities. In addition, the cost of developing compelling services using mashups can be significantly lower than the cost associated with building these solutions from scratch. As a result, mashups are currently gaining more attention from developers and end-users. Nevertheless, many enterprises still seem hesitant about adopting this methodology. One of the reasons for this limited interest involves the lack of tools for high-level development based on business and service requirements. It has to be noted that during the past decade service compositions were linked with SOAP-based services and BPEL, as RESTful services have only recently gained wider adoption. Nonetheless, in recent years a number of API repositories (e.g. Programmable Web and Mashery) have emerged that provide the means for creating enterprise mashups and, in addition, can assist the developing process by providing detailed documentation, users' feedback and online support. Furthermore, various researchers have recently proposed novel solutions for the construction of such

value-added services, while commercial suits have begun integrating tools for mashup development (e.g. Oracle's WebCenter and IBM's Mashup Centre) (Paredes-Valverde et al. 2015). These suites can provide flexible interfaces and business environments that can enable an enterprise to rapidly develop or customize web applications that can satisfy emerging business needs, thus, also highlighting the benefits of utilizing mashups. While this constantly increasing support indicates a shift towards a higher adoption of enterprise mashups, there is still a need for solutions that can verify or enforce specific 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 pertain to the behavior that is expected from business processes and external services, in order to satisfy the enterprise's predefined business needs (Kazhamiakin et al. 2004). Instead of focusing on the actual implementation of such processes and services in a transaction, they are closely connected to the way those can be utilized in order to achieve specific goals as well as to their mutual dependencies (Pistore et al. 2004). They are a primary concern during the development of enterprise information systems and could also include the application of execution constraints and the integration of predefined business agreements that should be fulfilled. An example of a business requirement could be a restriction to the execution order of external services that would prohibit the invocation of a certain service before the completion of another.

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 services or as a part of a WS composition. They are closely connected to the expected behavior of a WS in a certain environment (Wang et al. 2009). 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 (Chen et al. 2014). For example, a functional requirement of a map service could be that the service can calculate the distance between two given locations. Non-functional requirements refer to requirements that are not directly associated with the operations of the service. Such requirements could include the existence of certain quality of service (QoS) characteristics (El Kassmi et al. 2016). These characteristics can play a pivotal role in the selection of a service, between a number of alternatives that provide similar functionalities, and can include its response time, reliability and availability (Chen et al. 2014). If, for example, a given service is not always available for user to consume and has a relatively long response time it is not considered an ideal candidate for participating in a value-added service.

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 services, internal and external to the enterprise, that handle business processes (Leymann et al. 2002) 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 services that handle business processes 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 manner, 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 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 functionalities can be easily combined and integrated in a Web application or Web page. 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 is associated with lower developmental costs.

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 (Hoyer et al. 2011). Since the modeling of business flows in business process model and notation (BPMN) schemas is also less effort-demanding and their conversion to behavior, interaction, priorities (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 for 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.

In addition, mashups enable the simultaneous monitoring of up-to-date data originating from various sources. Furthermore, they provide the means for an enterprise to control the operations of its individual departments through a unified environment (Pahlke et al. 2010). As a result, they constitute ideal solutions for the development of information systems with relatively low costs. Nevertheless, the satisfaction of certain properties is critical in such systems and, as a result, a rigorous approach for their development, is of immense importance. Through our proposed methodology, businesses can ensure the compliance with pre-requested properties in mashup-based information systems.

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 monitoring modules can aid in the enforcement of correctness properties, such as the lack of deadlocks and interference, during the development of a service, which can also lead to lower testing and software maintenance costs.

1.4 Motivation and research contributions

As already mentioned, enterprise mashups consist of a number of internal and external services that are combined into a value-added service. They provide the opportunity to enterprises to monitor and handle information regarding the operation of their various departments and, additionally, enable users to interoperate with the enterprise in novel ways (Ghiani et al. 2016; de Vrieze et al. 2011). Nevertheless, enterprise mashups lack the appropriate mechanisms for their orchestration and the assurances that functional and non-functional properties are satisfied (Wilson et al. 2011; Zahoor et al. 2008). As the compliance with such properties is of significant importance in business-oriented scenarios, where faults cannot be tolerated, enterprises are often unable to reap the benefits provided by the utilization of mashups (Hoyer et al. 2011). 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.

The research contributions of our work can be summarized as follows: We demonstrate a methodology for transforming BPMN described scenarios, providing the business logic of a requested composite application, to the corresponding BIP semantics. Thus, we enable the handling of the value-added service's orchestration and the monitoring of requested properties by the BIP engine. In addition, we provide a novel architecture in BIP that can assist the rigorous development of enterprise mashups that comply with requested business and service properties. For the purpose of observing the compliance with the aforementioned properties, a number of monitoring modules, which can be integrated into the architecture, are also being demonstrated.

As BPMN models can be relatively easy to construct and comprehend, they can be ideal for describing an enterprise's business flows and do not require high programming skills for their development. Thus, through our proposed transformation, an enterprise can acquire a methodology for the rigorous development and orchestration of mashups without the need of extensive engagement from an IT department (Ghiani et al. 2016), as well as a means for the verification of required properties.

While the focal point of this manuscript is the construction of enterprise mashups that comply with predefined requirements, its theoretical contributions can be utilized for the development of various solutions based on the principles of software component design (Lau and Wang 2007). For example, organizations can model their individual business processes as a BPMN schema and, through the proposed transformation methodology, utilize BIP in order to develop an enterprise information system (Panetto and Cecil 2013). By modeling existing internal applications as atomic BIP components, and by integrating them in more complex compound components, they can validate the satisfaction of properties, the lack of deadlocks and the seamless information exchange between various business departments. This could lead to higher interoperability, improved productivity

and lower costs. The process of applying the BIP component framework for the development of software, based on the principles of component design, is feasible, as demonstrated in Basu et al. (2013).

The remainder of this paper is organized as follows: Sect. 2 pertains to the presentation of the related work, while in Sect. 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 Sect. 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 Sect. 6 demonstrates two case studies, used as proof of concept for our methodology. Finally, we give some future directions and conclusions stemmed from the application of this modeling approach.

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 procedure for the verification of desired properties, such as the one that can be offered by the BIP component framework. In Kheldoun et al. (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 et al. (2013), a modeling approach is described, which is centered on the process collaboration of end-users. This proposal adopts BPMN notations 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. (2016), 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, but 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 the aforementioned 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 cachable 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 being 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 modeling of 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.

2.3 Model-based and component-based software development

The BIP component framework has a powerful set of tools and engines that enable the development of software based on the principles of model-based and component-based development. An important advantage of BIP is the fact that it is not only a verification tool but an execution engine as well (http://www-verimag.imag.fr/BIP-Tools-93). In other words, the transitions described in a modeled system can be executed, as the engine enables C/C++ functions to be added to the model. As a result, the engine verifies properties in an executable system, thus also providing orchestration capabilities. Compared to other technologies and frameworks, BIP offers a number of significant advantages. Authors in Paredes-Valverde et al. (2015) discuss alternative tools, languages and methodologies for the development of composite mashups. Their comparison highlights the fact that a number of approaches support the synchronization between components of a mashup and the introduction of simple business rules and restrictions (such as Apatar and IBM Mashup Centre) but these tools do not provide verification capabilities, and thus cannot ensure the compliance of the resulting mashup with certain requested properties. Furthermore, authors in Zahoor et al. (2008) and Kil and Nam (2013) describe the application of model-based approaches that enable the verification of certain requirements, but lack the capabilities of orchestrating the resulting mashup, due to the lack of an execution engine.

As already mentioned in Sect. 1.4, the BIP component framework can be utilized in order to develop software, based on the principles of component design. Following those principles, authors in Basu et al. (2013) use the framework in order to design parallel applications in manycore devices. Through the modeling of the application software, they ensure the validation of specific functional properties and analyze the performance of the modeled system. Utilizing BIP, their proposed methodology also offers automated code generation and deployment on the targeted platforms, thus highlighting another significant advantage of the BIP component framework.

In addition, in Lekidis et al. (2015) authors apply the same principles, in order to propose a workflow for the rigorous development of IoT applications in the Contiki OS. By modeling the architecture of Contiki OS and the behavior of a requested application, using BIP, they propose a methodology for the development of IoT software and the enforcement of required properties.

Finally, in Mavridou et al. (2016), authors discuss the utilization of BIP for the development of the on-board control software of the CubETH satellite, through a series of steps. Initially, authors demonstrate the modeling of architecture styles for on-board software, a process followed by the analysis and formalization of requirements. Authors also describe the development of atomic components for the various functionalities of the modeled system and finally validate the model for the lack of deadlocks. Through, this process they ensure that the resulting software is correct-by-construction, a necessity in cases of software that handle critical operations, such as those performed in a satellite system.

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 and processes 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 | parll_ev | incls_ev | comlx
<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*, s*ubprocesses*, *transactions* and *activity calls*. A number of different types of gateways are defined in BPMN such as *exclusive*, *inclusive* 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. 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 by authors in Stachtiari et al. (2012). We have selected the following basic 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 Sect. 6.1.3.

The mapping of individual components of the BMPN model is explained, in more detail, below, while it is also schematically presented in Fig. 1.

For the transformation to the BIP model to be completed, individual events, activities and connected objects also have to be mapped. As already mentioned, there are three kinds of events in BPMN regarding their position in the overall business flow: *starting*, *intermediate* and *ending* events. The *starting* event of a BPMN process corresponds to the initial node in the BIP model, from where the execution will commence. *Intermediate* events are modeled in the form of transitions between intermediate nodes, while the *ending* events are mapped as transitions that are pointed towards the initial node and terminate the execution of the BIP model. Regarding their functionality, events can be in the form of *timer*, *message* or *conditional* events or can even lack a specific operation (such events are known as *none events*, which are commonly used as starting or ending events). *Timer* and *conditional* events are modeled as looping transitions, as they begin and end in the same node. Both types have a condition that ends the loop. In the case of *timer* events every loop increases the value of a variable by a certain time unit, until a threshold is reached. In *conditional* events a certain condition must be met in order for the execution to proceed to the next transition.

Examples of *conditional* events are presented in Fig. 2 below. Finally, *message* events can be modelled as simple transitions, as BIP enables the transfer of data and the allocation of new values in the model's variables with each transition.

Activities can be in the form of *tasks*, *subprocesses*, *transactions* and *call activities*. After the transformation process, *tasks* are depicted as individual nodes of a BIP component. They are included into the atomic component that models the lane they originally belonged to, in the BPMN process. As *sub-process* can contain a number of connected *tasks* and events, each *task* included takes the form of an individual node of the BIP model, while each event is also modeled based on the principles described above. The same notions are applied for *transaction* activities, but since those are handled as a whole by BPMN engines, a compensation mechanism is necessary in case one or more of the included *tasks* fail to complete. In this case, all actions performed by the involved *tasks* must be revoked. To model this behavior in BIP an additional node is created, that can handle this compensation and each individual node of the transaction can transfer the execution to this node through an appropriate transition. A differentiated activity type is the *call activity* type. It pertains to a global process that can be invoked by different *tasks* in the overall business flow. When this process is invoked, the execution is transferred to it. Such a process can contain a number of events and tasks, spanning across different lanes. As a result, in the produced BIP model it is mapped as a compound component.

BPMN also offers a number of gateways in order to describe an overall business workflow. The most commonly utilized gateways are the *parallel*, *exclusive* and *inclusive* gateways. All three types can be triggered by the completion of the task before them or by a specific event. In the latter case they can be characterized as

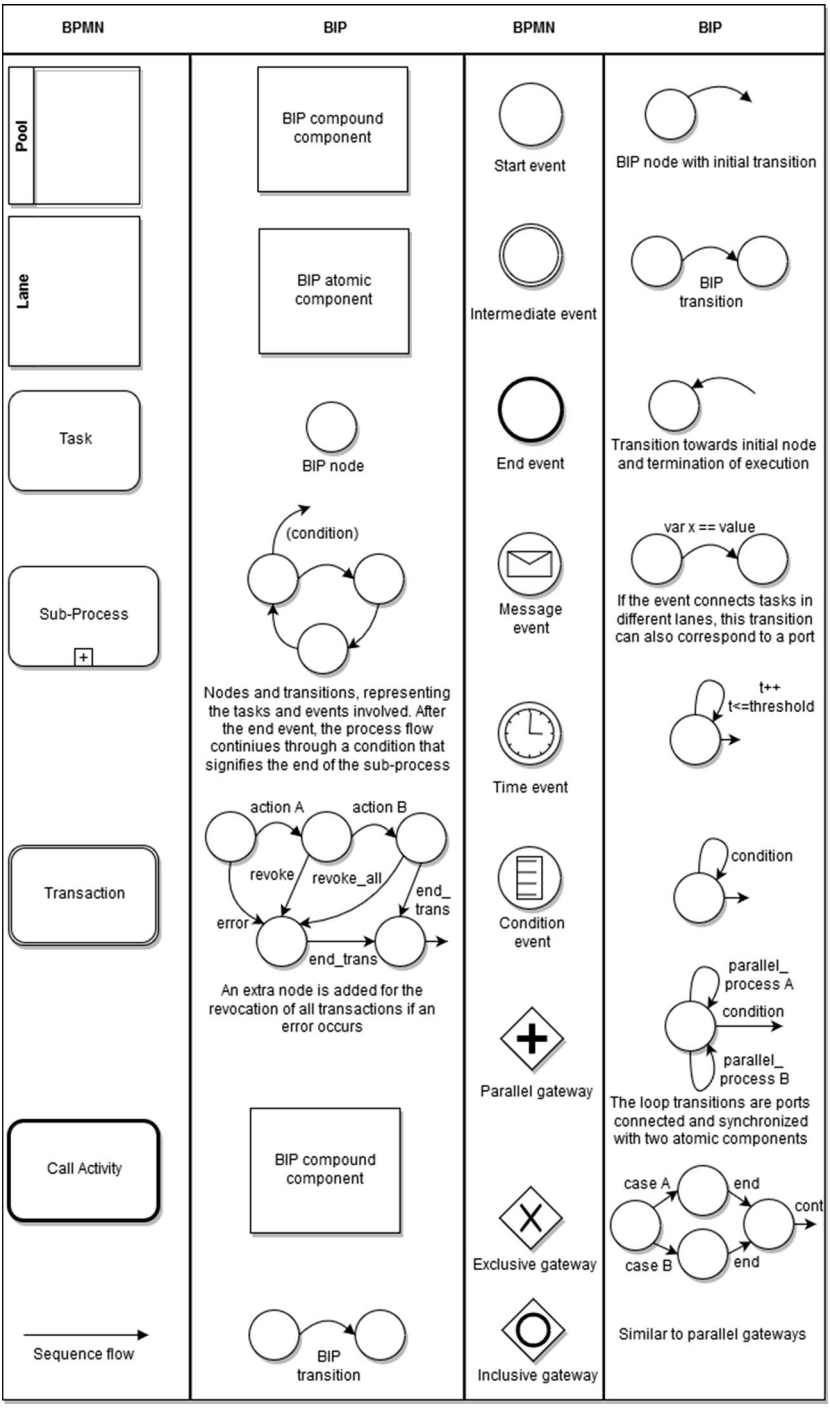


Fig. 1 Mapping of BPMN components to BIP

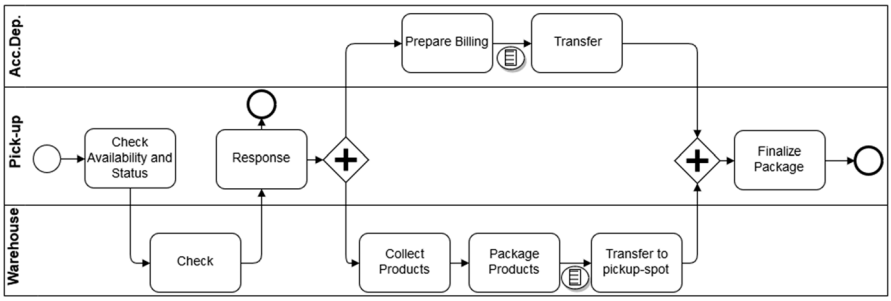


Fig. 2 Order procedure in BPMN

event-based gateways. *Parallel* gateways describe the concurrent execution of two or more *tasks*. In case these *tasks* are placed in different lanes, their mapping in BIP is accomplished by synchronizing atomic components with rendezvous. In more detail, the atomic components that describe the parallel *tasks* are connected with ports that trigger their parallel execution. When both *tasks* are completed a condition can enable the execution flow to continue to the next *tasks*. An example of the mapping of such a parallel gateway is presented in Fig. 2 where the *bill* and *collect* ports enable the synchronization of three atomic components and a condition enables the transfer of the execution to the next *task*.

In case the involved tasks are placed in the same lane, there is no need for the creation of additional atomic components, as parallel executions are possible in an atomic component and the transfer of the execution to the next task can be achieved with the utilization of an internal condition.

The mapping of e*xclusive* gateways is simpler, as those set mutual exclusive execution paths. As a result, they are modeled as multiple available transitions, beginning from the same node and leading to distinct destination nodes. Finally, *inclusive* gateways can lead to a serial execution flow, excluding the alternative paths, or to a concurrent execution of two or more paths, based on specific conditions. As a result, they are treated in a similar manner as parallel gateways, where the execution flows are modeled into distinct atomic components and in case two or more paths are to be executed in parallel, those are synchronized by rendezvous. While BPMN also supports the *complex* type gateways, users are generally recommended to utilize simpler gateways for the modelling of their business workflows. We opted not to develop a specific rule for the transformation of *complex* gateways, as those should be handled differently in each given scenario.

Regarding the relevancy of the mapping process, it must be noted that not all of the tasks described in the BPMN model have to be RESTful services. These tasks can also correspond to internal business processes. In both cases, ports are utilized in the generated BIP model in order to invoke a process or service and to pass the values of required variables. Especially in the case of RESTful services, ports are utilized in order to invoke specific operations of the service and to return information regarding the outcome of the operation.

As it has already been noted, BIP is not only a model verification tool but it also provides an execution engine (http://www-verimag.imag.fr/BIP-Tools-93). As a result, the BIP model that is generated after the transformation process, provides the ability to call existing RESTful services during its transitions. Furthermore, as BIP allows the execution of C/C++ functions, the transformation process can also provide a template for a RESTful service, where operations from an open source service can be integrated, or alternatively those can be developed from scratch according to the enterprise's needs. The feasibility of this approach has been demonstrated by authors in Stachtiari et al. (2014).

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 Fig. 2. 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.

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 accounting 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 Fig. 3. As seen in the depicted model, the three lanes of the BPMN scenario produce the same number of atomic BIP components.

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

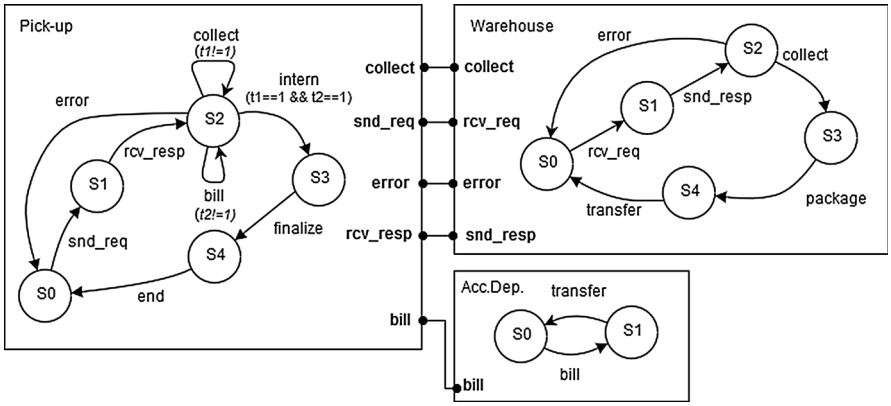


Fig. 3 Order procedure in BIP

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.

5 Mashup model

Enterprise mashups consist of WS that are external to an organization and a number of internal services and processes as well. They offer a unified environment that can help an enterprise or its clients to monitor data from multiple sources and complete various operations rapidly. As a result, they can be utilized for the development of information systems or web applications in which clients can interact with the enterprise. In addition, enterprise mashups can also contain widgets and data parsed from external sources, such as websites or databases (López et al. 2009; Hoyer et al. 2008; de Vrieze et al. 2011). For this reason, any given solution should provide the capability of invoking both services and widgets, while also allowing the integration of parsed data from various resources. To comply with those requirements, in this section, we present a reference architecture for enterprise mashups, modeled using the BIP component framework. Such an architecture is necessary as it describes the handling of the aforementioned operations. Furthermore, by modeling an architecture, global coordination properties can be enforced, thus enabling the development of correct-by-construction composite systems (Mavridou et al. 2016). Combined with the novel transformation presented above, it can aid the rigorous development of mashups, which adhere to predefined functional and non-functional requirements and can be orchestrated through the utilization of BIP's execution engine. The

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 a 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

architecture A_{MASHUP} consists of the application, core and service layers, as depicted in Fig. 4.

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 non-functional characteristics of the involved services are connected to the WS layer. More information regarding the architecture's port are given in Table 2.

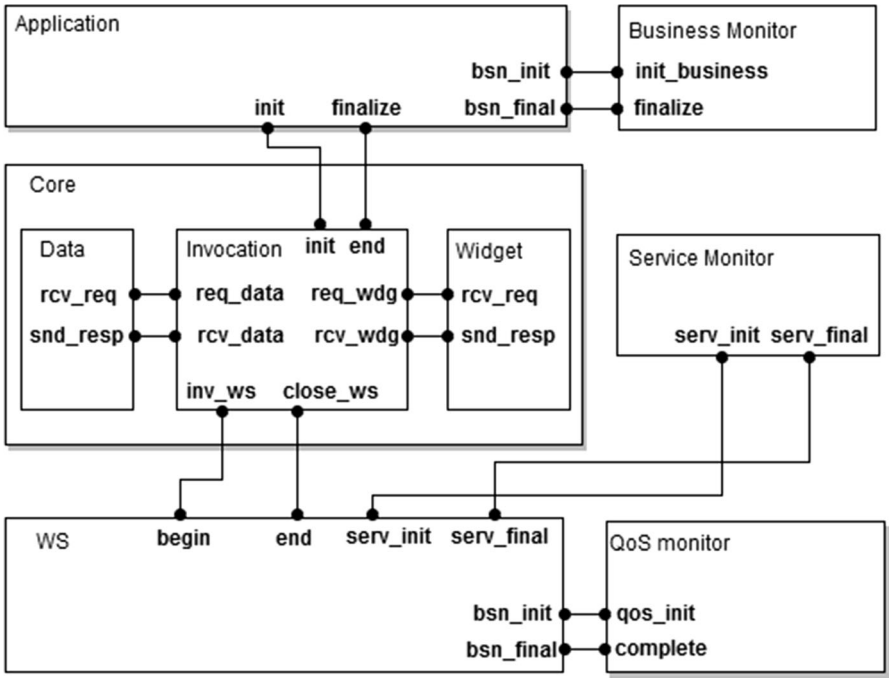


Fig. 4 The abstract A_{MASHUP} architecture

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 monitoring 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 first case study in Sect. 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 widgets and data by the BIP engine (through the core layer, as seen in the second case study in Sect. 6).

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 properties related to a given application.

6 Case studies

For the purpose of evaluating the proposed transformation methodology as well as the proposed architecture, we utilize two case studies. The evaluation objective is to determine whether the generated BIP models can identify the violation of a requested property, inform the user and terminate the overall procedure. In more detail, our interest is focused both on properties that are described by the BPMN model, such as properties that relate to the orchestration of the mashup as well as properties that are linked with the performance of the involved services. For this reason, we introduce specific business and service properties for each case study. In addition, by examining properties that relate to the overall orchestration, we are also able to determine the validity of the transformation process. As the requested business flow and the orchestration of the involved services are being described in the BPMN model, any differentiation during the execution of the mashup should result in the issuing of a violation message by the BIP engine. Thus, the two case studies also enable the evaluation of the transformation process, through the examination of the operation of the various atomic components.

The first case study is simpler and focuses solely on the application layer and the monitoring modules of the proposed mashup architecture. Thus it provides more insight on the transformation process and the way the BIP engine can identify and inform the user of possible deviations from the requested orchestration, originally described in the BPMN model. The second case study is more complex as it demonstrates various levels of integration of the application layer into the mashup architecture, utilizes a number of real world WS and follows a scenario-based approach in order to highlight the robustness of our approach. Thus, both case studies can shed light on the applicability of our approach and the benefits associated with its adoption by enterprises.

6.1 Case study A: 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 Fig. 5.

6.1.1 Services involved

The individual services involved are presented in more detail below:

(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.

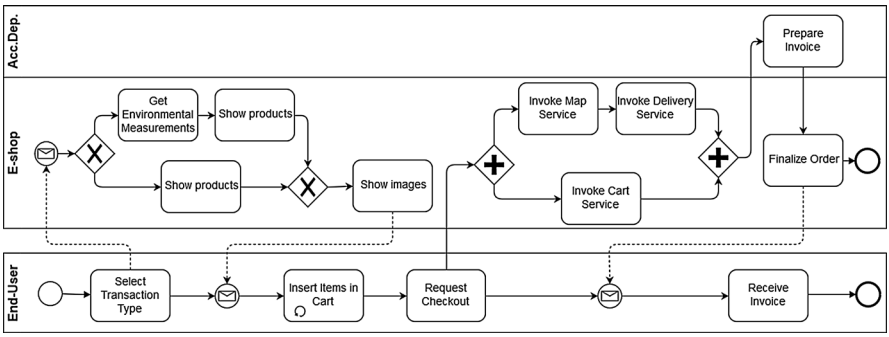


Fig. 5 BPMN model describing the execution flow

(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.

In case a B2B transaction is initiated the environmental measurement service is also being 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 into the application, demonstrates the feasibility of applying the proposed methodology to validate required properties using both virtual and physical services.

Based on the transformation rules described in Sect. 4.2, two compound components are initially created in the BIP model, as two pools exist in the BPMN model. The pool that corresponds to the store contains two lanes, one describing the process flow of the e-shop and another one that contains a process by the accounting department. As a result, the store compound component contains two atomic components. The starting transition of the BIP model leads to the execution of the "select transaction type" task from the end-user's lane. The message event originating from this task is modeled as a port (the *select_trans* port that is connected to the *rcv_trans* port) that synchronizes the "End-user" and "E-shop" atomic components. Similarly all events that require the existence of rendezvous between components are modeled as ports and their functionality is explained below. In order to ensure the compliance with a number of properties that are introduced in the following section, we have also integrated three monitors in the BIP model that are analytically presented in Sect. 6.1.3. The generated abstract BIP model is presented in Fig. 6, while the definition of included ports is presented in Table 3.

6.1.2 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.

6.1.3 Evaluation

By converting the BPMN model to the corresponding BIP model, we can ensure that the invocation order of the involved services, originally described in the BPMN

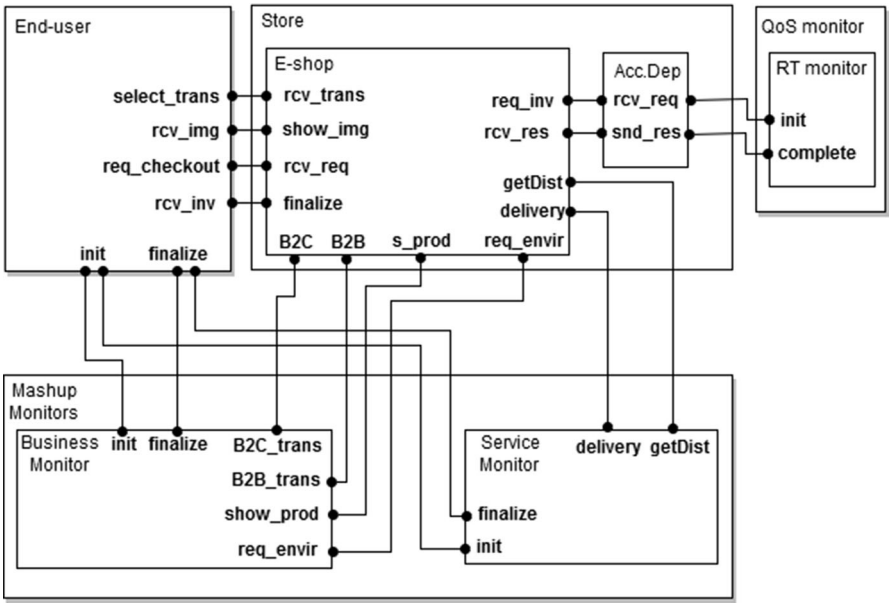


Fig. 6 The corresponding abstract BIP model with monitor integration

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 that 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

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

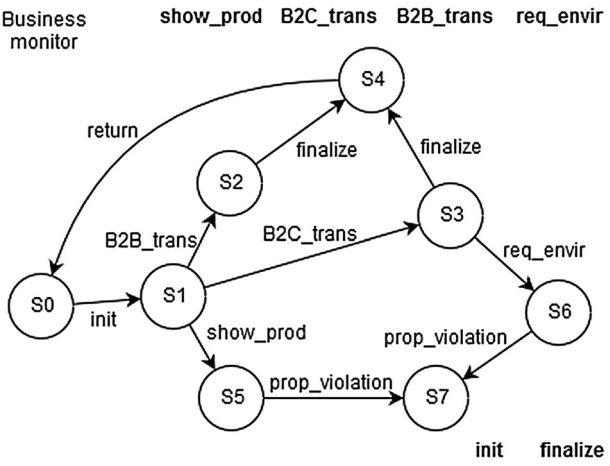


Fig. 7 The developed business monitor

satisfied, atomic monitor components can be applied. Such an approach has been presented in Stachtiari et al. (2012).

In this section, we aim to investigate whether the BIP engine can detect a violation in one or more of our requested properties that are associated with the orchestration described in the BPMN model. In more detail, if during the execution of the mashup a deviation from the business flow presented in the BPMN model occurs, the BIP engine should reach a deadlock state and terminate the execution of the overall process. In order to achieve this, two monitor modules are integrated into the BIP model, which can examine the fulfillment of properties SP1, BP1 and BP2. In addition, the detection of such errors by the BIP engine also validates the accuracy of the transformation process.

Using the tools provided by the BIP component framework, we examined all execution seeds of the mashup model to validate the fulfillment of requested requirements and to verify the deadlock freedom property.

Figure 7 presents the 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.

Aiming to validate property SP1, we have developed the service monitor component presented in Fig. 8. The component reaches a deadlock in case a service does not complete its requested functionality.

In case an erroneous transition, which enables the invocation of the delivery service prior to the map service, exists 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.

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.

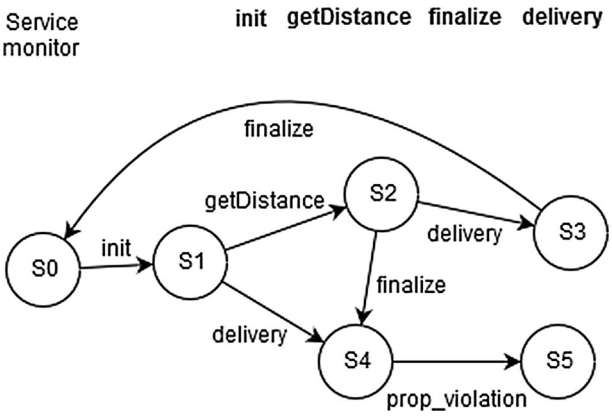


Fig. 8 The developed service monitor

```
[BIP ENGINE]: state #11: 1 interaction:
[BIP ENGINE]:   [0] ROOT.customer.reqCheckout: C.outReq(binding=2;
addressTo=;runtime=0;error=0;) ES1.prepReq(binding=2;addressTo=;ru
ntime=0;error=0;)
[BIP ENGINE]: -> choose [0] ROOT.customer.reqCheckout: C.outReq(bi
nding=2;addressTo=;runtime=0;error=0;) ES1.prepReq(binding=2;addre
ssTo=;runtime=0;error=0;)
[BIP ENGINE]: state #12: 2 interactions:
[BIP ENGINE]:   [0] ROOT.eshop.getDistance: ES1.ReqDis(binding=2;
addressTo=;runtime=0;error=0;) S1.DistReq(binding=2;addressTo=;run
time=0;error=0;)
[BIP ENGINE]:   [1] ROOT.eshop.delivery: ES1.ReqDel(binding=2;add
ressTo=;runtime=0;error=0;) S1.DelReq(binding=2;addressTo=;runtime
=0;error=0;)
[BIP ENGINE]: -> choose [1] ROOT.eshop.delivery: ES1.ReqDel(bindi
ng=2;addressTo=;runtime=0;error=0;) S1.DelReq(binding=2;addressTo=
;runtime=0;error=0;)
[BIP ENGINE]: state #13: deadlock!
```

Fig. 9 Detected deadlock signifying property violation

As highlighted in Fig. 9, the BIP engine detects deviations from the intended business flow and orchestration and, as a result, terminates the overall process. This behavior demonstrates the applicability of our approach for the development of enterprise mashups that adhere to predefined properties and that do not enable the execution of seeds that violate those properties. The fact that the monitor described above returns an error message in case the execution of the mashup deviates from the business flow described in the BPMN model, validates the robustness of our transformation methodology. Nonetheless, the benefits from the utilization of our approach are not only restricted to the identification of the fulfillment of functional properties. Thus, in order to evaluate the non-functional service property SP2, pertaining to the reliability property, we have developed the monitor module depicted in Fig. 10.

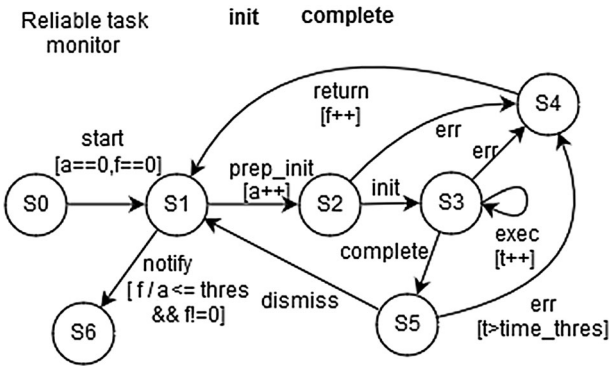


Fig. 10 The reliability monitor

The monitor counts the overall attempts to execute a specific service, as well as the number of recorded 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 associated with 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 Fig. 11 and in Table 4.

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 Fig. 12.

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.

While in this section we also utilized state space exploration in order to demonstrate the validity and trueness of the model, this is not a necessity for developers that apply our methodology when creating a mashup. BIP offers a wide selection of tools for validation purposes and while its state space exploration capabilities are not significantly effort-demanding, developers can alternately use the simpler interactive execution of the model in order to examine the available transitions step-by-step. This way they can easily detect deadlocks and identify parts of the execution process in which potential flaws may exist.

6.2 Case study B: an enterprise mashup for a hotel chain

In the second case study we present the development of a mashup that contains four real-world WS and two of an enterprise's internal services. It pertains to the

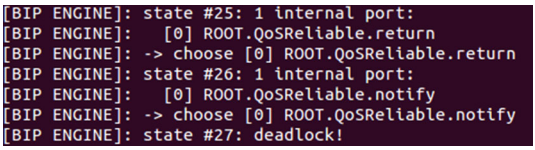


Fig. 11 Detected deadlock triggered by the reliability monitor

Table 4 Results of the alteration of threshold values

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

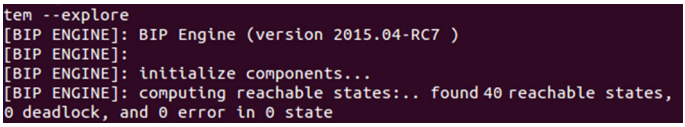


Fig. 12 Deadlock freedom as a result of threshold adjustment

business logic of a hotel chain that enables users to make a booking in one of their hotels and additionally presents multimedia information regarding the selected hotel's city. The business logic of this case study is again based on a simple execution flow, described in BPMN. The description of the individual internal and external services involved is given below. For the deployment and execution of the mashup, the Apache web server is utilized, since it enables the interaction with BIP, through CGI scripts as a way to dynamically add content to a webpage.

The BPMN model is converted to the corresponding BIP model, based on the methodology described in Sect. 4.2. In order to assess the robustness of the approach we utilize three case scenarios that involve different levels of integration of the application layer to the overall A_{MASHUP} architecture.

6.2.1 Services involved

The individual services involved are presented in more detail below (Fig. 13):

(i) Booking service: Based on the principles adopted by online booking applications, we have developed a booking service that can be used internally by the hotel management. This internal service receives

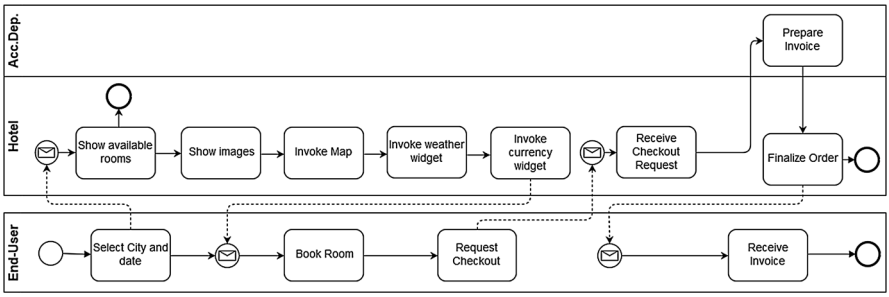


Fig. 13 BPMN model describing the execution flow of an online booking

information regarding the city that the user wants to visit, the date of his arrival and departure, the number of adults that want to stay in the hotel, as well as the number of the requested rooms. The service sends a request to the hotel's database and returns a list of available rooms. When the user selects a room, the room's cost is transferred to the *invoice service*.

(ii) Show images: The *show images* service receives the requested city as an input and returns the first five images retrieved by Flickr's API, when the name of the city is given as input and the results are ranked by their interest. By utilizing this real world API, the enterprise mashup is enriched by content generated by the city's visitors.

(iii) Map service: By utilizing the Google Maps' API, the address of the hotel, as recorded in the hotel's database is visually presented in a map. Thus, using the *map service* the end-user can locate points of interest near the hotel's proximity.

(iv) Weather widget: Using the widget module, segments of a website or widgets that are already developed by third parties can be embedded into the mashup. In this enterprise mashup, the requested date of arrival is used as input in a weather forecast service. Since widget applications are available by a number of weather services (e.g. https://www.accuweather.com/en/free-weather-widgets/current), those can be embedded into the value-added application. Utilizing this widget, clients can rapidly receive valuable information regarding the weather conditions that are expected during their arrival.

(v) Currency widget: As potential clients may use different currencies in their countries than the one preferred by the hotel, a widget that can instantly convert money values can be embedded in the mashup. By utilizing the widget module, a pre-developed service (https://www.exchangeratewidget.com/currency-converter-widget/) is added to the mashup.

(vi) Invoice service: For the needs of this case study we have developed a service that receives the user's personal information and the room's charge and creates a printable invoice. This service can be used as an internal service by the hotel chain and can be added to the enterprise mashup.

Taking into consideration the rules described in Sect. 4.2, the corresponding BIP model is developed. In more detail, two compound components are created since two pools exist in the BPMN model. The pool that corresponds to the hotel contains two lanes, one describing a service executed by the accounting department and the other describing the execution of all other services and widgets involved in the booking process. As a result, the compound component named *enterprise* contains two atomic components.

6.2.2 Requested property

For the needs of this case study we have selected one required service property (SP1) and one required business property (BP1).

SP1: Involved services are reliable. Similarly to the first case study, 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.

BP1: The *Invoice* service should only be invoked after the invocation of all other services. As services are invoked in a serial manner, the business monitor needs only to examine whether the *invoice* service is initiated after the *currency* widget (or the *map* service instead, in case the application layer is not integrated into the A_{MASHUP} architecture as seen in the second case scenario in the evaluation section below).

6.2.3 Evaluation

For evaluating the robustness of the transformation methodology and the correctness of the A_{MASHUP} architecture, a scenario based approach has been selected. Such an approach has been adopted by researchers in various fields. For example, authors in de Bruin and Van Vliet (2001) applied this methodology to assess quality characteristics in a set of generated software architectures. This approach has also been applied in order to evaluate how software architectures affect the fulfillment of business requirements when developing enterprise information systems (Niu et al. 2013) and to examine behavioral system properties such as the existence of deadlocks (de Bruin 2000).

In the first case scenario, the application layer is used as a standalone module and is not integrated into the A_{MASHUP} architecture. In addition, it is not connected to any monitor modules. As a result, the generated mashup cannot handle widgets and the end- user cannot monitor the fulfillment of the required properties SP1 and BP1. For example, if the *invoice* service is invoked before the completion of all other services and such a transition is enabled in the BIP model, the end-user will not be notified that the requirement BP1 has been violated. The same limitation is present in cases where a service does not complete its functionality in reasonable time frames, thus violating the requirement SP1. We examined a number of possible execution seeds and encountered instances of this behavior, which highlights the need for a higher degree of integration of the application layer into the proposed architecture.

In the second case scenario, the application layer is not integrated into the A_{MASHUP} architecture, nonetheless a number of monitor modules are being utilized. In more detail, the reliability monitor presented in the previous case study as well as a business monitor are being connected to the application layer. A limitation observed in this scenario is that the reliability monitor can only be connected to a single service, as was also the case in the previous case study. This limitation occurs due to the fact that the A_{MASHUP} architecture is not being utilized. Regarding the fulfillment of property BP1, the developed monitor terminates the overall process in case the *invoice* service is being invoked before the invocation of the *map* service. We examined various execution seeds and encountered such occurrences, where the end-user is being notified of a property violation. While this approach enables the monitoring of requested properties, as well as the deadlock-freedom property, it has some limitations. First of all, since the application layer is not integrated into the A_{MASHUP} architecture, the reliability monitor can only be connected to a single service. In addition, the generated BIP model cannot handle the invocation of widget modules. This signifies that benefits can be reaped by the integration of the model to our proposed architecture, as seen in the final case scenario.

In the third, and final, case scenario, the application layer is connected to the A_{MASHUP} architecture model, thus the overall mashup can also handle widgets, while the reliability monitor can observe the behavior of all involved services. The enterprise mashup that is developed as a result of the overall process is presented in Fig. 14.

The BIP model that resulted after the transformation process corresponds to the application layer of the architecture. The core layer contain the widget module, that is responsible for the integration of widgets, and the invocation module that is responsible for the invocation of involved WS. Finally, in the WS layer the service

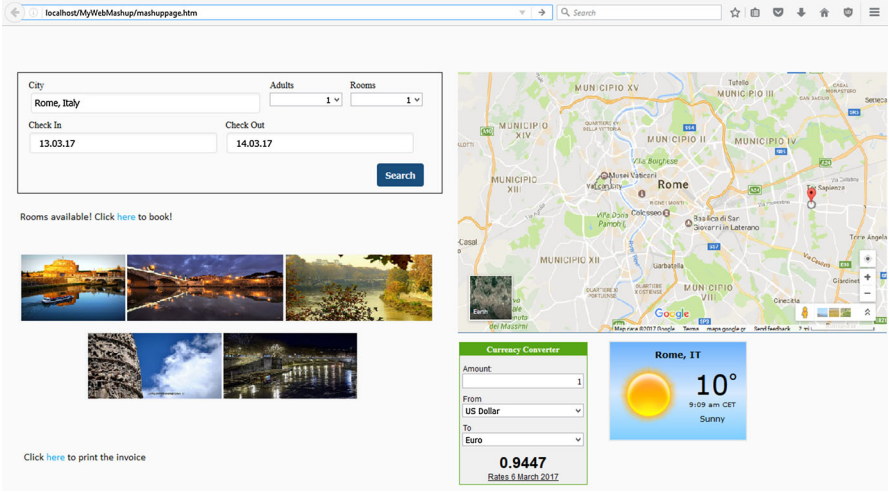


Fig. 14 An enterprise mashup for a hotel chain

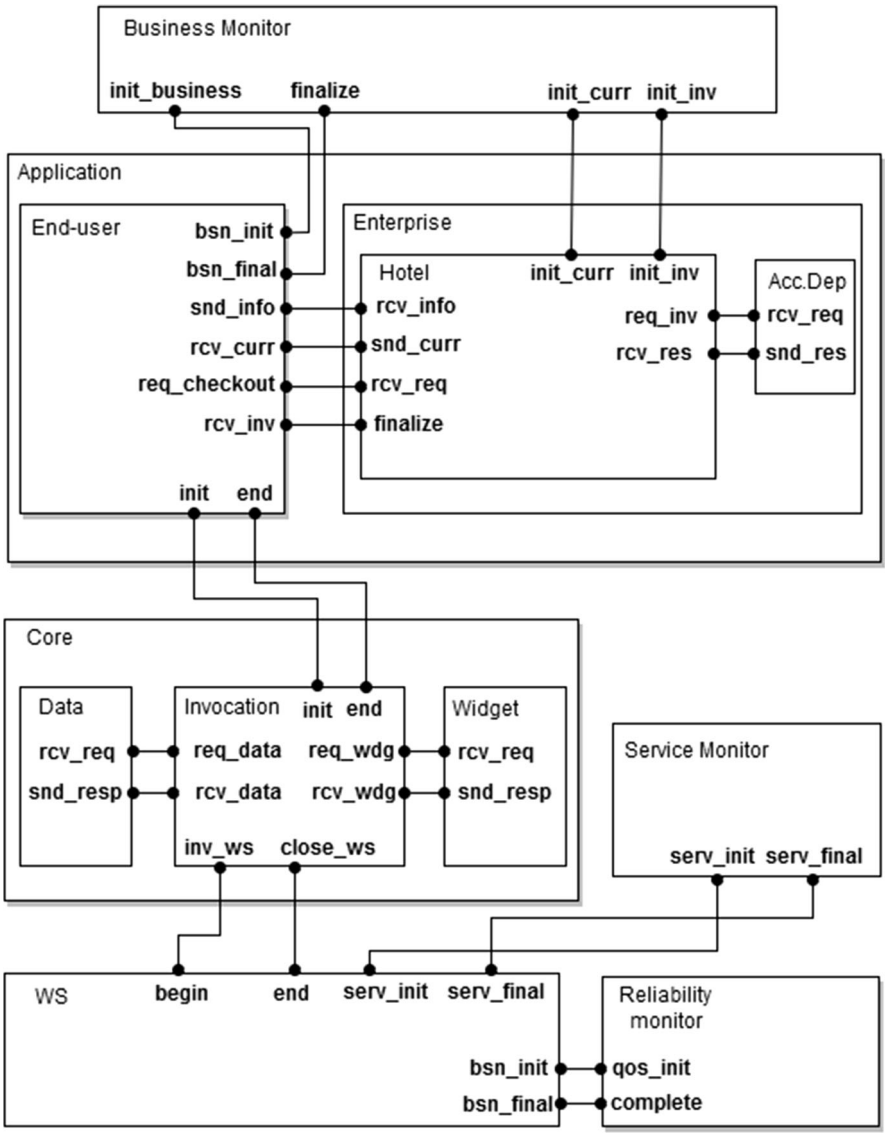


Fig. 15 Abstract BIP model integrated into the A_{MASHUP} architecture

interactions are described and the connection with the reliability and service monitors is accomplished.

The generated abstract BIP model is presented in Fig. 15, while the definition of included ports in the application layer is presented in Table 5. In addition, Fig. 16 presents the transitions that are included in the invocation and widget modules.

Through the transformation of the BPMN model to the corresponding BIP model and its integration to the A_{MASHUP} architecture, the BIP engine can handle the

Table 5 Definition of utilized ports in the application layer

End-user	
bsn_init	Initializes the business monitor
bsn_final	Finishes the synchronization with the business monitor
snd_info	Sends the booking information to the hotel
rcv_curr	Receives notification that the currency widget is invoked
req_checkout	Issues a payment request for the booking
rcv_inv	Receives the order's invoice
init	Signals the initialization of the procedure
end	Signals the finalization of the procedure
Hotel	
rcv_info	Receives information on the requested booking
snd_curr	Signals a notification that the currency widget is invoked
rcv_req	Receives a request for the payment of selected rooms
finalize	Informs the user that the order is finalized
req_inv	Requests the order's invoice
rcv_res	Receives the order's invoice
init_curr	Signals the initiation the currency widget
init_inv	Signals the initiation of the invoice service
Accounting department	
rcv_req	Receives an invoice request
snd_resp	Sends the corresponding invoice

orchestration of the resulting enterprise mashup and the fulfillment of requested properties SP1 and BP1. As stated above, the business monitor does not allow the invocation of the *invoice* service before the invocation of other services. In this case study, and as the mashup can also handle widgets, we opted to notify the end-user in case the *invoice* service is being invoked before the invocation of the *currency* widget. In case such an erroneous transition occurs, a violation is detected by the engine. In more detail, a deadlock is reached in case the user attempts to initiate the *init_inv* port in the modeled system, resulting into the termination of and the overall execution. The business monitor's detailed behavior, is depicted in Fig. 17.

Using the tools provided by the BIP component framework, we examined a number of possible execution seeds of the mashup model.

As seen in Fig. 18, there were instances where the business monitor detected the violation of property BP1 and terminated the overall execution.

The findings of the three case scenarios highlight the need for applying the proposed A_{MASHUP} architecture and, in parallel, show the robustness of our approach. Firstly, through the execution of possible execution seeds in all three case scenarios we validated that the mashup adheres to the business flow originally presented in the BPMN model. Secondly, through the utilization of our transformation methodology and the integration of the generated model into the overall

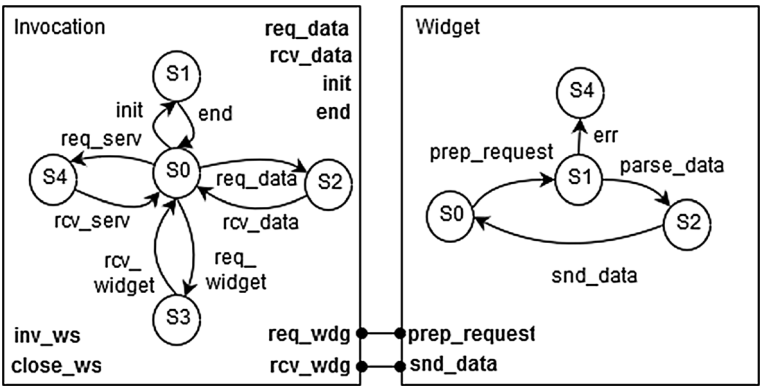
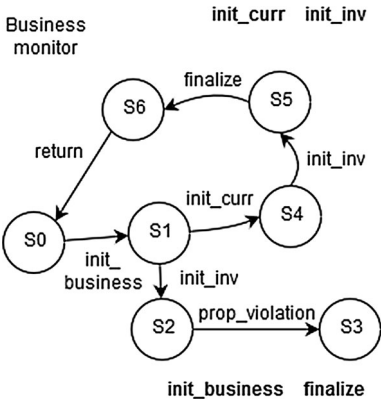


Fig. 16 The invocation and widget modules of the core layer

Fig. 17 Component monitoring the invocation order of services




```
[BIP ENGINE]: state #36: 2 interactions:
[BIP ENGINE]:   [0] ROOT.BusinessMonitor.invokeCurr: B.CurReq(binding=2;
addressTo=;runtime=0;error=0;) ES1.ReqCurr(binding=2;addressTo=;runtime
=0;error=0;)
[BIP ENGINE]:   [1] ROOT.BusinessMonitor.invokeInvoice: B.InvReq(binding
=2;addressTo=;runtime=0;error=0;) ES1.ReqInv(binding=2;addressTo=;runtim
e=0;error=0;)
[BIP ENGINE]: -> choose [1] ROOT.BusinessMonitor.invokeInvoice: B.InvReq
(binding=2;addressTo=;runtime=0;error=0;) ES1.ReqInv(binding=2;addressTo
;runtime=0;error=0;)
[BIP ENGINE]: state #37: 1 internal port:
[BIP ENGINE]:   [0] ROOT.BusinessMonitor.propviolation
[BIP ENGINE]: -> choose [0] ROOT.BusinessMonitor.propviolation
[BIP ENGINE]: state #38: deadlock!
```

Fig. 18 Execution reaches a deadlock due to the violation of property BP1

proposed architecture, we demonstrated how an enterprise mashup can be developed. In addition, we presented how, using this approach, the orchestration of the value-added service can be controlled by the BIP engine. Finally, we pointed

towards the fact that, through the application of our methodology, the generated mashup adheres to predefined functional and non-functional properties and can notify the end-user in case of a property violation.

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 means 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 is 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 enable clients to receive more personalized services, as they can be more involved in the development process.

Future work involves the development of an automated tool for the transformation of BPMN models to their corresponding BIP modules. This is a feasible approach, as tools that generate BIP models through a transformation process have been demonstrated in Stachtiari et al. (2012) and Chkouri et al. (2008), in order to convert BPEL and Architecture Analysis and Design Language (AADL) models, accordingly, to their corresponding BIP models. In addition, we work towards the integration of SOAP-based services into enterprise mashups. Furthermore, 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, pp 314–333

Bliudze S, Sifakis J (2008) A notion of glue expressiveness for component-based systems. In: Proceedings of 20th international conference on concurrency theory (CONCUR), pp 508–522

Bozzon A, Brambilla M, Facca FM, Carughu GT (2009) A conceptual modeling approach to business service mashup development. In: Proceedings of 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 Semant Sci Serv Agents World Wide Web 1:281–308. doi:10.1016/j.websem.2004.03.001

Chen M, Tan TH, Sun J, Liu Y, Dong JS (2014) Veriws: a tool for verification of combined functional and non-functional requirements of web service composition. In: Proceedings of 36th international conference on software engineering (ICSE), pp 564–567

Chkouri MY, Robert A, Bozga M, Sifakis J (2008). Translating AADL into BIP-application to the verification of real-time systems. In: Proceedings of international conference on model driven engineering languages and systems (MODELS), pp. 5–19

de Bruin H (2000) Scenario-based analysis of component compositions. In: Proceedings of international symposium on generative and component-based software engineering. Springer, Berlin, pp 131–148

de Bruin H, van Vliet H (2001) Scenario-based generation and evaluation of software architectures. In: Bosch J (eds) Generative and component-based software engineering. Springer, Berlin, pp 128–139

de Vrieze P, Xu L, Bouguettaya A, Yang J, Chen J (2011) Building enterprise mashups. Future Gener Comput Syst 27:637–642. doi:10.1016/j.future.2010.10.004

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, Berlin, pp 73–87

El Kassmi I, Jarir Z, Obaid A (2016) Non-functional requirements interdependencies in web service composition. In: Proceedings of IEEE international conference on systems of collaboration (SysCo), pp 1–6

Ghiani G, Paternò F, Spano LD, Pintori G (2016) An environment for end-user development of web mashups. Int J Hum Comput Stud 87:38–64. doi:10.1016/j.ijhcs.2015.10.008

Glombitza N, Pfisterer D, Fischer S (2010) Using state machines for a model driven development of web service-based sensor network applications. In: Proceedings of 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: Proceedings of 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, pp 286–299

Hoyer V, Stanoesvka-Slabeva K, Janner T, Schroth C (2008) Enterprise mashups: design principles towards the long tail of user needs. In: Proceedings of IEEE international conference on services computing (SCC), pp 601–602

Hoyer V, Stanoevska-Slabeva K, Kramer S, Giessmann A (2011) What are the business benefits of enterprise mashups?. In: Proceedings of IEEE 44th Hawaii international conference on system sciences (HICSS), pp 1–10

Isikdag U (2015) Enhanced building information models: using IoT services and integration patterns. Springer, Cham

Kazhamiakin R, Pistore M, Roveri M (2004) A framework for integrating business processes and business requirements. In: Proceedings of eighth IEEE international enterprise distributed object computing conference (EDOC), pp. 9–20

Kheldoun A, Barkaoui K, Ioualalen M (2015) Specification and verification of complex business processes—a high-level petri net-based approach. In: Motahari-Nezhad RH, Recker J, Weidlich M (eds) Business process management. Springer, Berlin, pp 55–71

Kil H, Nam W (2013) Semantic web service composition via model checking techniques. Int J Web Grid Serv 9:339–350. doi:10.1504/IJWGS.2013.057466

Kim YS, Shin DH, Jeon HB, Lee KH, Cho KS, Park W (2013) Conflict detection in composite web services based on model checking. Int J Web Grid Serv 9:394–430. doi:10.1504/IJWGS.2013.057470

Lau KK, Wang Z (2007) Software component models. IEEE Trans Software Eng 33:709–724. doi:10.1109/TSE.2007.70726

Lekidis A, Stachtiari E, Katsaros P, Bozga M, Georgiadis CK (2015) Using BIP to reinforce correctness of resource-constrained IoT applications. In: Proceedings of 10th IEEE international symposium on industrial embedded systems (SIES), pp 1–10

Leymann F, Roller D, Schmidt MT (2002) Web services and business process management. IBM Syst J 41:198–211. doi:10.1147/sj.412.0198

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

López J, Bellas F, Pan A, Montoto P (2009) A component-based approach for engineering enterprise mashups. In: Proceedings of international conference on web engineering (ICWE), pp 30–44

Mavridou A, Stachtiari E, Bliudze S, Ivanov A, Katsaros P, Sifakis J (2016) Architecture-based design: a satellite on-board software case study. In: Proceedings of 13th international conference on formal aspects of component software (FACS 2016)

Niu N, Da Xu L, Bi Z (2013) Enterprise information systems architecture—analysis and evaluation. IEEE Trans Ind Inf 9:2147–2154

Pahlke I, Beck R, Wolf M (2010) Enterprise mashup systems as platform for situational applications. Bus Inf Syst Eng 2:305–315. doi:10.1007/s12599-010-0121-9

Panetto H, Cecil J (2013) Information systems for enterprise integration, interoperability and networking: theory and applications (editorial). Enterp Inf Syst 7:1–6. doi:10.1080/17517575.2012.684802

Paredes-Valverde MA, Alor-Hernández G, Rodríguez-González A, Valencia-García R, Jiménez-Domingo (2015) A systematic review of tools, languages, and methodologies for mashup development. Softw Pract Exp 45:365–397. doi:10.1002/spe.2233

Pistore M, Roveri M, Busetta P (2004) Requirements-driven verification of web services. Electron Notes Theor Comput Sci 105:95–108. doi:10.1016/j.entcs.2004.05.005

Polymerou E, Chatzakou D, Vakali A (2014) Emotube: a sentiment analysis integrated environment for social web content. In: Proceedings of 4th international conference on web intelligence, mining and semantics (WIMS14), pp 20–25

Ruhi U, Choi D (2013) Enterprise mashups for knowledge management. In: Proceedings of 1st international conference on information and communication technology trends (ICICTT), pp 159–168

Said NB, Abdellatif T, Bensalem S, Bozga M (2016) A robust framework for securing composed web services. In: Braga C, Ölveczky PC (eds) Formal aspects of component software. Springer, Berlin, pp 105–122

Sifakis J (2014) Rigorous system design. In: Proceedings of 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: Proceedings of 19th IEEE international conference on web services (ICWS), pp 319–326

Stachtiari E, Vesyropoulos N, Kourouleas G, Georgiadis CK, Katsaros P (2014) Correct-by-construction web service architecture. In: Proceedings of 8th IEEE international symposium on service oriented system engineering (SOSE), pp 47–58

Vesyropoulos N, Georgiadis CK (2013) Web of things: understanding the growing opportunities for business transactions. In: Proceedings of 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 Q, Shao J, Deng F, Liu Y, Li M, Han J, Mei H (2009) An online monitoring approach for web service requirements. IEEE Trans Serv Comput 2:338–351. doi:10.1109/TSC.2009.22

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

Wilson S, Daniel F, Jugel U, Soi S (2011) Orchestrated user interface mashups using w3c widgets. In: Proceedings of international conference on web engineering (ICWE), pp 49–61

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, Berlin, pp 87–102

Xu L, de Vrieze P, Phalp K, Jeary S, Liang P (2013) Interoperative end-user process modelling for process collaborative manufacturing. Int J Comput Integr Manuf 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: Ghose Aditya et al (eds) Service-oriented computing—ICSOC 2012 workshops. Springer, Berlin, 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. IseB. doi:10.1007/s10257-016-0317-8

Zahoor E, Perrin O, Godart C (2008) Mashup model and verification using mashup processing network. In: Proceedings of international conference on collaborative computing: networking, applications and worksharing, pp 632–648