PerfBPEL: a graph-based approach for the performance analysis of BPEL SOA applications

Enrico Barbierato
Dipartimento di Informatica
Università di Torino
Torino, Italy
barbierato@mfn.unipmn.it

Mauro Iacono
Dipartimento di studi Europei e Mediterranei
Seconda Università di Napoli
Caserta, Italy
mauro.iacono@unina2.it

Stefano Marrone
Dipartimento di Matematica e Fisica
Seconda Università di Napoli
Caserta, Italy
stefano.marrone@unina2.it

***Abstract*—Non-functional properties evaluation in Service Oriented Architecture (SOA) is still mostly an open challenge. Although this is a problem that has been already partially explored with some success, there is lack of consolidated results for more complex SOA applications based on services composition. This paper presents a contribution to performance evaluation of SOA-based applications integrated by BPEL. The evaluation technique is based on a performance-oriented reinterpretation of the BPEL specification as a performance modeling language within a multiformalism framework. The approach is based on automatic translation of PerfBPEL into Markov chains and it is implemented by means of SIMTHESys modeling and analysis framework to enable the interaction with other performance oriented formalisms.**

I. Introduction

Performance evaluation of complex systems is a key issue for the development of QoS aware applications and systems. The field of business oriented services requires a high degree of flexibility and the ability to react timely to changes: this motivation has led to methodological and technological solutions based on the Service Oriented Architecture (SOA) paradigm. SOA provides the means to encapsulate and publish worldwide atomic functionalities as services. The given abstraction allows automatic integration of services in wider applications, described as business processes and automatically implemented by means of languages like BPEL. This approach enables designers: i) to rapidly compose the applications needed by the customers, ii) to easily reconfigure them by changing the set of involved services and iii) to add the required integration logic. The availability of complex distributed architectures, such as the ones exploiting cloud computing technologies, complicates scenarios but amplifies the benefits of the approach, maximizing reuse possibility, scalability, competition on the market and cost optimization.

The involvement of third-party, geographically distributed services in an application complicates the design. While standard tools have been implemented to support the automatic inclusion of services and significant research results are available for the verification of correctness of the resulting application, a little has been done for the prediction of the performances of such applications. The possibility of evaluating overall performances of a SOA based application since the early phases of the design cycle covers an important role. Firstly, it simplifies the choice between different solutions with different costs and secondly it allows the implementation of QoS policies. Moreover, the transformations that can derive by the exploitation of cloud technologies require the analysis to be performed on a timescale that must be proportional to the rapidity of the changes. The evaluation should be performed automatically on demand, by exploiting the description of the workflow, freeing the developer from the burden of manually generating the models.

This paper presents an automatic performance oriented modeling approach aiming to minimize the paradigm shift from designing and modeling SOA applications, by directly using BPEL workflows to generate performance models. The approach is based on the development of a modeling formalism, namely PerfBPEL, that mimics BPEL constructs and execution semantics, enriching it with performance oriented attributes.

The main contributions of this approach are: minimal syntactical difference from BPEL workflows to lower the cultural gap between modelers and designers; automatic generation of PerfBPEL models from BPEL workflows; possibility of use of different formalisms to model the execution architecture, according to modelers' needs; faithful representation of the effects of BPEL semantic details.

After this introduction, Section 2 gives the reader references about SOA and related modeling approaches relevant for this work; Section 3 motivates the work by presenting PerfBPEL methodological perspective; Section 4 presents PerfBPEL details and the related model solution process, applied to a cloud-based case study in Section 5. Finally, conclusions and future works close the paper.

II. Related Works

SOA is an architectural model for the support of widely distributed software application based on the concept of service. According to the Service Oriented Computing paradigm, a service is an autonomous, platform-independent computational entity that can be used independently by the platform. SOA leverages the potential of Internet to allow software applications (in turn, usable as complex services) that use services available in a network, thus representing the widest accepted model to design geographical distributed systems. It promotes

VALUETOOLS 2012, October 09-12, Cargèse, France
Copyright © 2012 ICST
DOI 10.4108/valuetools.2012.250340

loose coupling between software components, improving their reusability [11] in dynamic business processes. One of the most promising research trend in this field is the evaluation of non functional features of Composed Web Services as stated in [26].

A crucial aspect for the design of SOA applications is service composition, both for what is related to correctness and performance (QoS) aspects. Besides the technical support offered by standards, verification of the correctness of compositions is a widely and successfully explored problem in literature. Correctness verification has been studied by means of different kinds of formal methods (pi calculus, logic, model checking) but the most relevant technique for the purposes of this Section is given by Petri Nets, since Petri Nets have also been used in some performance evaluation work in literature. A detailed description of the problem of correctness in terms of workflows can be found in [29], that surveys this argument, and many other works of the same author. A model checking oriented transformational approach is presented in [16], related to the development of automatic transformation between BPEL (that is a standard de facto for the implementation of SOA applications) and Petri Nets.

On the other hand, the evaluation of QoS in Composed Web Services is still an important open issue. It involves the study of functional and non-functional attributes of a service as performance, availability, security, reconfigurability and so on. The vastity of this issue, cumulated with the need to satisfy related design specifications in a non-controllable, complex environment such as the Internet, represents a grand challenge for researchers. Web Services performances are studied by means of direct measuring and statistical techniques (see e.g. [27] that introduces time management extensions).

The focus of this paper concerns formal methods based techniques. Following this direction, literature presents different studies using Performance Evaluation Process Algebra [15], timedCCS Process Algebra [24], Timed Automata in [10], and of course Petri Nets [20], [9], [31], [17] (the latter presents a clear introduction to the problem, describing a method to evaluate BPEL workflows - in its BPEL4People extension - by analyzing them with GSPN models).

Performance and performability measures have been successfully performed taking advantage of multiformalism modeling techniques. Many experiences in supporting and developing multiformalism modeling techniques and frameworks are reported in literature (Sharpe [28], SMART [4], [5], Mobius [8], [6], [7], OsMoSys [30], [13], [25], [14], SIMTHESys [19], [2], [1], [3], [18]). In this paper SIMTHESys is used to support the development of our approach.

III. Motivation

SOA is advocated to enable business agility [21], due to the principles on which it is inspired: reuse, coarse granularity, modularity, composability, componentization and interoperability. Coarse grain atomic software composition is used as an agile alternative to traditional software development cycles approaches. This approach allows more efficient and consistent results although it requires a higher complexity in the design and more implementation efforts.

The flexibility of SOA is comparable to the richness of capabilities offered by the cloud computing paradigm that allows computing resources i) to be used from the Internet on a on-demand basis and ii) to be seamlessly ran on heterogeneous, (geographically) distributed architectures.

The typical scenarios involve frequent reconfigurations of (high-level) software and hardware architectures and composition of applications by using alternative third-party existing services, differently from typical distributed systems, in which software and hardware are usually designed as quasi-stable, interdependent entities. Moreover, the professional background required in this field is usually different from the typical background of modeling experts.

The change of scenarios, the little time spent on development and the short life cycles of these systems make performance evaluation a difficult and crucial factor for success. The availability of an automated modeling technique, directly representing (BPEL) workflows and fit to the first steps of the development cycle would lower the cultural gap between workflow and model design. As a consequence, designers could exploit their experience on the base of performance evaluations without the need for a radical change of perspective.

Since the most spread approach for the development of SOA applications exploits BPEL, this paper considers a BPEL representation as a starting point for the software level of SOA. The typical scenario is thus given by a distributed software layer, composed of intercommunicating BPEL workflows, and a distributed architecture layer. The latter is composed by different application servers (hardware and application server software) and the computer network to which they are connected.

A. Why PerfBPEL

BPEL is a complex language. Modeling BPEL, as seen, has been the goal of several works. Given the modeling power of Petri Nets and the fact that they have been used for both correctness and performance analysis of BPEL, two main Petri Nets based references have been considered while developing this paper: [23] and [17]. The first presents a comparison of two automatic verification approaches, that give complete and detailed descriptions of all aspects of the language, by articulated modular nets. The second presents a method to describe by a GSPN a given workflow, for performances evaluation, taking in account the main aspects of the language. Although such approaches are very sound and well designed, using the first as a basis for a performance oriented extension would produce very complex nets for common workflows, thus complicating the analysis and affecting its complexity; the second seems not sufficiently detailed to capture all the execution semantics of BPEL (e. g. the execution of fault handling, quite frequent in BPEL) and allow an automatic translation. Moreover, detailed Petri Nets representations of workflows appear as syntactically distant from BPEL descriptions, and are difficult to handle and understand for the average BPEL designer.

In this paper a dedicated modeling formalism, namely PerfBPEL, is introduced. PerfBPEL is designed to represent all performance related aspects of BPEL while keeping the structure of models as similar as possible to BPEL workflows. BPEL constructs are rendered with one to one PerfBPEL equivalents, that capture the same semantics and enrich it with performance related annotations. This allows the direct translation of BPEL workflows into PerfBPEL models, easy to be handled by designers and ready to be completed with performance parameters (not allowed by pure BPEL standard workflows). Such a tool allows a model-based analysis of an application in the first phases of design, with a minimal alteration of the development cycle that supports design choices since its beginning.

In the approach proposed in this paper, PerfBPEL is used to describe the SOA software layer and is implemented as a (graph-based) formalism for the SIMTHESys framework [19], [2], [1], [3], and the architecture layer is described by a dedicated formalism, that describes the deployment of services. Using SIMTHESys enables the use of other formalisms too, such as Generalized Stochastic Petri Nets (GSPN), Queuing Networks (QN) or Fault Trees (FT), to represent services or other architectural components for the convenience of designers.

The main focus of this paper is on PerfBPEL. The rest of this Section is dedicated to more detailed considerations about its concept, its main aspects and its solution method.

B. Main issues

BPEL offers different paradigms and features like concurrency, synchronization, message-based communication, exception handling and the use of variables. Concurrency and synchronization paradigms suggest (as seen in the related works) that formalisms like GSPN or PEPA could be used as an intermediate modeling step. Even if the BPEL synchronization mechanism has a complex semantics[1] (represented by Petri Nets [22]), in the general framework of the approach presented here this would require a completely automatic generation of complex nets, that would be then analyzed to generate an equivalent solution model (e.g. a Markov chain). The generation of a Markov chain from BPEL minimizes the state space and simplifies the modeling of BPEL paradigms, so PerfBPEL models are solved by generation and analysis of equivalent Markov chains.

The virtually unlimited set of possible values of variables is collapsed in relevant subsets, each one determined according to the combination of conditions present in a given workflow. In a case-by-case basis, this approach allows reducing the original number of possible values of a variable during the execution of the workflow to a finite and generally small number of states.

Note that stochastic characterization of the states of variables is required when a deterministic value is not available or is not generated during the execution of the workflow. This can be derived by profiling analogous workflows, analyzing the specifications or setting proper hypotheses. Both the aspects of variables characterization should be taken into account, to enhance the quality of the performance model by letting correlations emerge from the model itself.

C. PerfBPEL models evaluation

PerfBPEL models evaluation is implemented by the SIMTHESys modeling framework, that supplies the SIMTHESysER solvers generation tool. This choice lets PerfBPEL leverage all characteristics available in it. SIMTHESys is a multiformalism modeling approach that allows the development of modeling formalisms by specifying the static and dynamic aspects of its modeling elements. SIMTHESys allows: the automatic generation of solvers for models written in formalism compositions, by using solving engines based on elementary general solution algorithms; and models composition and formalisms extension by means of the behavioral interfaces.

The proposed approach supports message-based communication and fault handling by exploiting the features of the framework. Since the use of the framework natively enables multiformalism modeling, the architectural layer of the systems to be analyzed is specified by means of a separate ancillary formalism, designed to describe how system components are deployed and connected. Moreover, by exploiting multiformalism the modeler can represent the behavior of some components by using submodels written in other well-known formalisms (such as GSPN, QN and FT), to consider peculiar effects of the dynamics of the architecture or of third-party services, or to let modelers evaluate more complex phenomena (such as performability characteristics or effects of the application of user-defined mechanisms).

D. Current implementation and limitations

For the purposes of this paper, besides the development of a dedicated tool to transform BPEL workflows in PerfBPEL models, SIMTHESysER has been used to generate a multiformalism solver. This is based on GSPN, QN, FT and CTMC solving engines, capable of analyzing SOA system models relying on PerfBPEL and the ancillary formalism. Note that the case study presented in the paper does not explicitly show the use of exception handling BPEL constructs due to space restrictions (the use of exception handling primitives [3] in the case study would have complicated the description of the structure). Variables are specified as for the previous subsections with reference to state reduction and stochastic characterization, with the exclusion of correlation management. Currently, synchronization inside Flow constructs is not covered by the implementation. Finally, message-based com-

[1]The Link clauses within Flow constructs are subject to a complex enabling logic with mutual consequences, namely the Dead Path Elimination [12], that involves variable and logic clauses evaluation and influences the result of the execution of the whole Flow construct.

munication is enabled by the use of the ancillary formalism to integrate the submodels forming the architecture.

IV. The PerfBPEL approach

The PerfBPEL formalism is based on the reference implementation WS-BPEL 2.0 [12]. BPEL is a XML-based language designed to represent orchestration of services as a workflow. BPEL constructs are divided in Basic Activities and Structured Activities. Basic activities are atomic operations of the language, while Structured Activities describe its control flow.

A. BPEL constructs

A brief description of BPEL Basic activities is given in Table I. WS-BPEL supports both one-way and two-ways communications by a correlation mechanism used with Invoke, Receive and Reply: correlation allows the use of the communication primitives to obtain synchronous and asynchronous service invocations. A brief description of BPEL Basic activities is given in Table II.

TABLE I
BPEL basic activities

NAME	DESCRIPTION
Invoke	Synchronous or asynchronous invocation of a service offered by a partner
Receive	Waits for a message from a partner (used to complete the protocol in case of asynchronous invocation of a service)
Reply	Sends a message to a caller (used to complete the protocol in case of asynchronous invocation of a service)
Assign	Assigns a value to a variable
Validate	Validates the values of process variables against their associated XML and WSDL data definitions
Wait	Waits for a timer to expire
Empty	Null action
Throw	Launches an exception
Rethrow	Propagates an exception to the outer scope
Exit	Immediately terminates a BPEL process with no termination handling, fault handling or compensation behavior
Compensate	Handles the actions needed in case an activity cannot be completed
Compensate scope	Handles the actions needed in case a scope cannot be completed
Extension activity	Hooks for the extension of the available set of activities with a personalized activity

B. Designing PerfBPEL

The SIMTHESys modeling framework offers a sound approach to formalism development. It prescribes each formalism to be defined in terms of language elements, each of which is characterized in terms of properties (that define their structure), and behaviors (that define their execution semantics). As a result, BPEL has been analyzed according to the SIMTHESys approach by bearing in mind that PerfBPEL should be as much as possible similar to BPEL.

The analysis has been guided by some considerations. Firstly, the goal is to isolate the business logic from choreography details as much as possible, in order to increase

TABLE II
BPEL structured activities

NAME	DESCRIPTION
Scope	Defines inner activities as an atomic group
Process	Defines the main BPEL workflow
Fault handler	Defines a group of actions launched in case of fault in the execution of an activity
Compensation handler	Defines a group of actions launched in case an activity cannot be completed
Termination handler	Defines a group of actions launched in case of termination of the workflow
Event handler	Defines a group of actions launched in case an event happens. Used for asynchronous execution of parts of the workflow
Flow	Executes activities in parallel and waits for their completion. Internal synchronization between parallel execution branches is possible by using Link
If-Else	Executes alternative actions depending on the value of a logic condition
Pick	Executes alternative actions depending on the arrival of one between a set of messages or the expiration of a timer between a set of timers
Sequence	Executes a sequence of activities one after the other in the specified order
While	Executes repeatedly an activity until a condition keeps true (the activity is not executed if the condition is false)
Foreach	Executes repeatedly in a sequence or in parallel an activity N times, and terminates successfully if M execution are successful

reusability and allow early application in the SOA development cycle. Secondly, the details of real service invocations (not necessarily available in the first phases of the cycle) have been ignored in favor of a simplified and generalized approach. Thirdly, value-based choices and messaging have been modeled as probabilistic, since the effective details about variables, types, values and called services are not known at design time. Finally, a static analysis-like quantification can give information about how the application will behave in the average case.

Each BPEL construct corresponds to a PerfBPEL element, including the behaviors that mimic the proper execution semantic and a set of properties. Additional elements have been added to connect the BPEL-equivalent elements in a graph structure and implement the interactions that represents the possible paths of the BPEL execution flow. The aspect related to performances have been implemented as additional properties describing the parameters needed to evaluate execution time (in terms of deterministic and exponentially distributed duration).

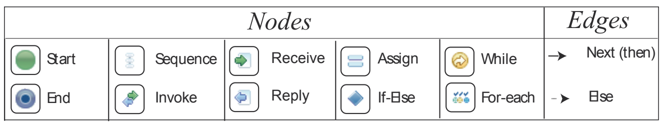

Fig. 1. The elements of PerfBPEL

Fig. 1 presents the BPEL primitives currently considered in PerfBPEL. Future work will incorporate other BPEL activities. Following the SIMTHESys approach, a model is defined by a graph structure. The BPEL model has been transformed into a graph by adding an arc (the *Next* arc in Fig. 1) that connects each primitive to the following one. Since some the structured activities can have more than one following action, we have introduced another type of arc (the *Else* arc). Following the SIMTHESys methodology, the properties that define an element can be classified into three sets: *static* properties, that define the parameters of the model; *dynamic* properties, that are used to identify the state of the model; and *measures* that specify the performance metrics that can be computed for an element. Moreover, the entire model can have its own properties. The following introduces the single nodes composing the formalism and their properties:

BPEL. It represents the entire model, and contains all the other primitives. Its static properties are: *variables*, an array of labels that represents all the variables used in the model, *valueSet* that defines a set of ranges of values that the variables can assume (i.e. the variable types), and *variableType* that associates to each variable one of the available types. Its dynamic properties are: *currentActivity*, that points to the activity that is currently being performed by the model, and *waitingForEvent* used to wait for external events. This property is empty if the system is performing an internal action. Otherwise, if the system is waiting for an external event, it specifies a label for it. The last dynamic property is *variableValues* that stores the current value of all the variables.

Start and **End**. They define the beginning and the end of the model. They have no properties and they must be unique in the entire model. The former cannot have any incoming arcs, while the latter cannot have any outgoing arcs.

Sequence. This element has no properties and represents the start of a sequence of actions. Since a PerfBPEL model is a graph of actions, it is not strictly necessary (the sequence is already implemented by the *Next* arcs). However, it has been included to support the future extension of the formalism to support management of concurrent execution flow inhibition.

Assign. It is characterized by two static properties: *variable*, that represents the variable set by the activity, and *value* that contains the value that it is going to be assigned. When this activity is executed, the given variable is set to the specified value.

Invoke. It is used to model the invocation of an external service. It has a static property called *service* that includes the name of the service that it is going to be called. It also has a boolean dynamic property called *waiting* that defines the state of the element. As soon as the element is considered, it sends the service request corresponding to a label contained in the static properties and it goes to a wait state (by setting *waiting* to *true*). The name of the service is also inserted in the *waitingForEvent* property of the model (the `BPEL` element) to block the execution until the required event is received. The `Invoke` element has also a static property called *rate* that specify a (possibly 0) mean exponential waiting time to represent the time required to issue the request.

Reply and **Receive** are similar to the `Invoke`, as they generate respectively an event or wait for an event to occur. Both have the static property *service* that includes the name of the service that is due to be performed. `Receive` has the dynamic property called *waiting* with a behavior similar to the receiving part of the `Invoke` element. In the same way, `Reply` has a *rate* property that specify the mean waiting time before sending the event.

If-Else. It has two mutually exclusive static properties: *condition* and *probability*. If *condition* is present, it corresponds to a boolean expression involving some variable. If *probability* can be specified as a real value in the range $[0, 1]$ that determines the probability that the condition will be true. An `If-Else` element must have exactly two outgoing arcs: a *Next* arc that represents the direction of the flow of action to be followed when the condition is true, and an *Else* arc that has to be chosen when the condition is *false*. The choice of the next action is either deterministic (if *condition* has been specified), or probabilistic (when *probability* is present).

While. It behaves exactly as the `If-Else` element, with the only extra feature that the component flow is required to return the `While` element to perform a loop.

For-each. It has two static properties: *variable* that represents the variable used to identify the iterations, and *set* that represents the different values that the variable will assume.The value of the considered variable will be used to determine which service has to be called during each iteration. The element also has a dynamic property *currentValue* that stores the current value of the iteration. A `For-each` element has two outgoing arcs: one of *Next* type and one of *Else* type. When there is a new value in the set that the variable can assume, the execution flow continues along the *Next* direction. As soon as all the values of the set have been considered, the execution follows the *Else* arc. Note that this is currently just an approximation of the actual BPEL Foreach structured activity, where actions can be performed either in series or in parallel.

C. The bridge formalism

In the SIMTHESys methodology, submodels written in different formalisms are logically connected by arcs in an external formalism (the *bridge formalism*) that defines the interactions among the various primitives. The ancillary formalism for PerfBPEL is a bridge formalism.

To model the intercommunication among BPEL processes, a new arc called *Connection* is added to the formalism. This arc has a static property called *service* that contains the label of a service request. The *Connection* arcs joins the requesting process with the replying one. Specifically, service requests produced by the element from which the edge starts, and that match the label of the arc, are sent to the process at the other end. If that process is blocked waiting for that particular event, it allows the process to continue to its next state. The same *Connection* arc can also start from elements that defines the `push` behavior and end at elements implementing the `setOccupancy` behavior; moreover, arcs that implement the `isActive` behavior can also terminate on a BPEL submodel:

these features are useful to create multi-formalism models and will be covered in more detail in the following section.

D. The solution process

The solution process is based on the automatic generation of a Markov chain for the model. The generation is obtained by applying the activation rules of a PerfBPEL model, emulating the behavior of a BPEL workflow engine. The state of the engine is obtained from the composition of the distributed state described by the properties of the PerfBPEL elements in the model.

All the exponential and immediate events based SIMTHESys formalisms operate by defining a behavior called `initEvents`. The purpose of this behavior is to find which events (either characterized by an exponentially distributed firing time, or zero time and event probability) are enabled and can occur in a given state. With respect of the BPEL elements, `initEvents` first checks if the process is enabled by looking for all the arcs that end on the considered sub-model and which implement the `isActive` behavior. If at least one of this arcs returns *false*, then the component is considered to be inactive. This is used for example to stop an activity using a place of a GSPN and an inhibitor arc. If the BPEL sub-model is active, then the property *waitingForEvent* is checked to see if the process is blocked waiting for a reply. If the property is not empty, then the execution terminates since the component is waiting for an external event to continue. If *waitingForEvent* is empty, the execution continues by reading the current activity from the property *currentActivity*. Depending on the specific activity, `initEvents` decides which events can occur and schedules them. `Start` and `Sequence` schedule an immediate event that changes the current action to the one found along its exiting arc (that must be present and must be unique for the model to be valid). `End` resets all the variables of the component, and returns the current action to the `Start` element. `If-Else` and `While` schedule, if their *probability* property is set, two immediate events corresponding to the system choosing the *then* or the *else* path. If the *condition* property is set, the condition is evaluated and only one immediate event is generated, along the path corresponding to the value of the test. `Assign` sets the variable to the value defined in the corresponding two properties, and schedules an immediate transition to the next activity. `For-each` assigns the current value to the variable, and then sets its *currentValue* property to the next element of the set (defined in the corresponding property). It schedules an immediate transition along the *Next* arc if there are other elements in the set, or along the *Else* arc if there are no new values to be assigned. As outlined earlier, the `Receive` element sets the *waitingForEvent* property of the BPEL submodel, and schedules no action. When the event is received, it also sets its dynamic property *waiting* to resume the operation along the path that starts from this element. When a *Connection* arc of the bridge formalism executes a `push` behavior, it checks if the sub-model at the destination of the arc is waiting for the event specified in the *service* property of the arc. If it matches, the transition to the next state in the BPEL submodel is scheduled as an immediate event to release the process and continue to the next activity. Since the `push` behavior is very general (it is used for example by arcs exiting from GSPN transitions and FCQN queues to move tokens and customers among their elements), it could be used to create complex inter-formalism interactions, by allowing for example the firing of a GSPN transition to define the completion of a service. A `Reply` activity schedules either an immediate or a timed (exponentially distributed) event, depending on whether the *rate* property is present or not. When this event is executed, it looks for all the arcs that start from the BPEL submodel in which the element is contained. For all the arcs whose *service* property matches the one contained in the corresponding property of the `Reply` element, it executes a `push` behavior to send this event to other sub-models. It also changes the *currentActivity* property of the BPEL sub-model to the next activity connected to the `Reply` node. The `push` behavior of the *Connection* arc checks (assuming that it is not connected to a BPEL sub-model) that its destination element supports the `setOccupancy` behavior. If this is the case, it uses this behavior to increase of one unity the occupancy of the destination element. This can be used in multi-formalism models, for example to increase the number of tokens in a GSPN place, or the jobs in a FCQN queue, when a specific event occurs. Finally, the `Invoke` activity is executed by calling first a `Reply` and then a `Receive` activity on the same *service*. Fig. 2 shows the events that are scheduled, and the time in which the process is stopped waiting for synchronization, obtained for the PerfBPEL model of Fig. 7 (that is the equivalent of the BPEL process given in Fig. 4).

V. Case study

A. Description

The application proposed as case study is related to the topic of resource management and service deployment in a cloud computing context. In particular, the need to have a sustainable computing implies the necessity to improve corporate IT infrastructures efficiency even by means of outsourcing and externalization of such facilities. In other words a user can be invited to use external service from the Cloud rather then expanding owned data center. This choice can be mediated by means of SOA paradigm where BPEL-based services can hide the real location of services implementation (if they are deployed on “real” data centers or on the Cloud).

As it is depicted in Fig. 3, this system is composed of five different web applications:

- *Broker*: it is an application deployed on the user access portal used to determine where the service invoked by the user would be executed. This application will be better described later since it is at the core of this study;
- *Negotiator*: it runs on the cloud providers front-ends, catching requests from the Service Client. If the provider cannot satisfy user’s requests, it propagates the request to other cloud providers;

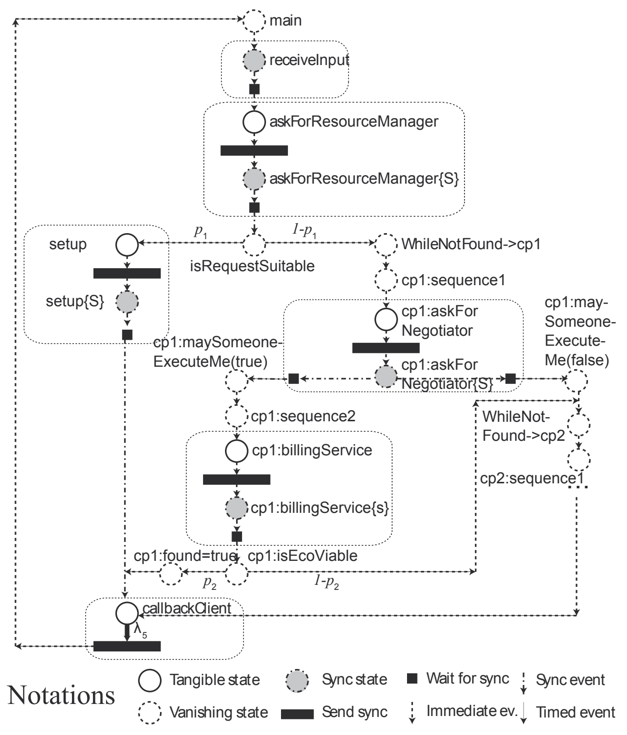

Fig. 2. Equivalent Markov chain for Broker BPEL program

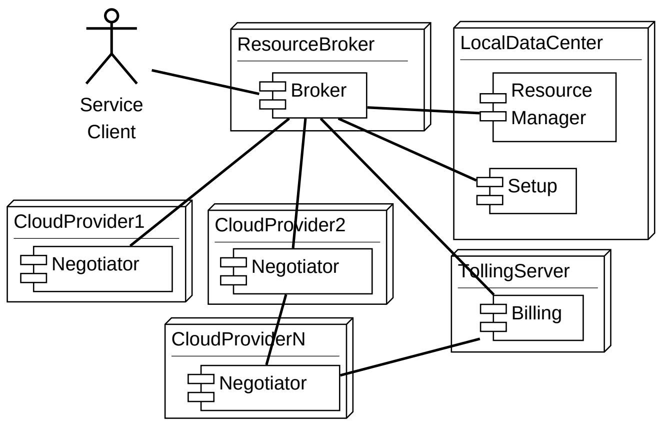

Fig. 3. Service negotiation in Cloud context

- *Resource Manager*: this is a “in-house” service that confirms if the owned data center can take the Service User request;
- *Setup*: it allows the data center to prepare the execution of requested service;
- *Billing*: this is a service hosted on a Tolling Server that is responsible to determine to economical viability of the operation when cloud user/provider check agrees to externalize/accept the requested service.

The interaction starts with the request of a user that activates the *Broker* service. This service asks for *Resource Manager* on its data center first. If it is not possible to execute service “in-house”, it asks for service execution from cloud providers invoking the *Negotiator* services. If one cloud provider answers positively, the phase of economic viability starts by means on the invocation of the remote service *Billing*. If, on the other hand, service can be executed by owned data center (meaning that *Resource Manager* has provided a positive response), *Broker* activates the *Setup* procedure. If a cloud provider cannot satisfy the request, it propagates it to other cloud providers.
We make the hypothesis that the decision to accept or not a user request is taken by a Cloud Provider Administrator (CPA). In this context, the last two services are regarded as an external black-box while the formers are considered two proper BPEL applications. They are depicted respectively in Fig. 4 and Fig. 5.

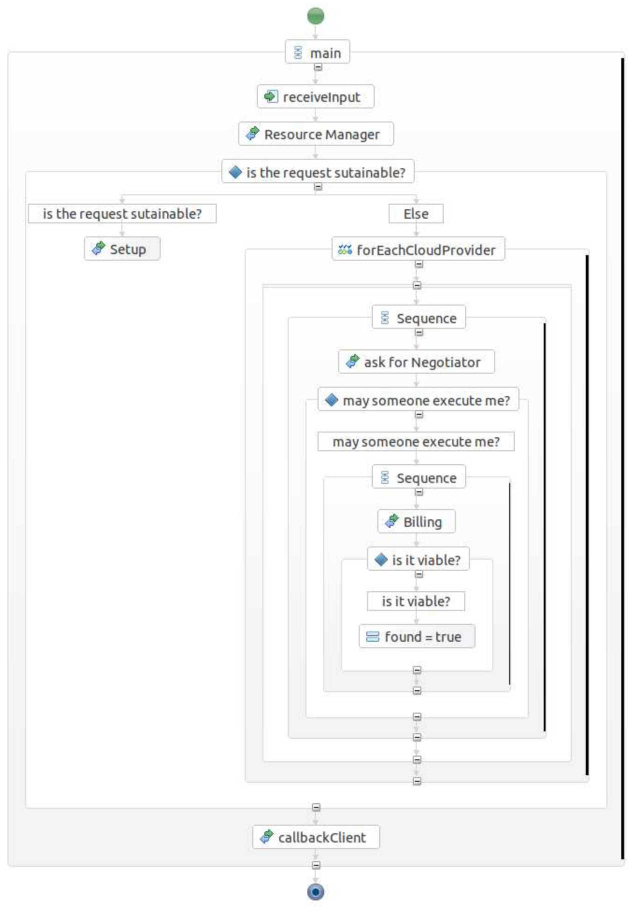

Fig. 4. Broker BPEL workflow

B. Modeling

The model of the system is in Fig. 6. To simplify the presentation, a single edge with arrows on both ends corresponds to two edges connecting the same elements in opposite directions.

The evaluated configuration has two cloud providers (1 and 2) that are directly known by the *Broker* and a third provider (N) that is known by the second provider. These two CWS are represented by PerfBPEL and are detailed in Fig. 7 and in Fig. 8. Being connected by a network, the

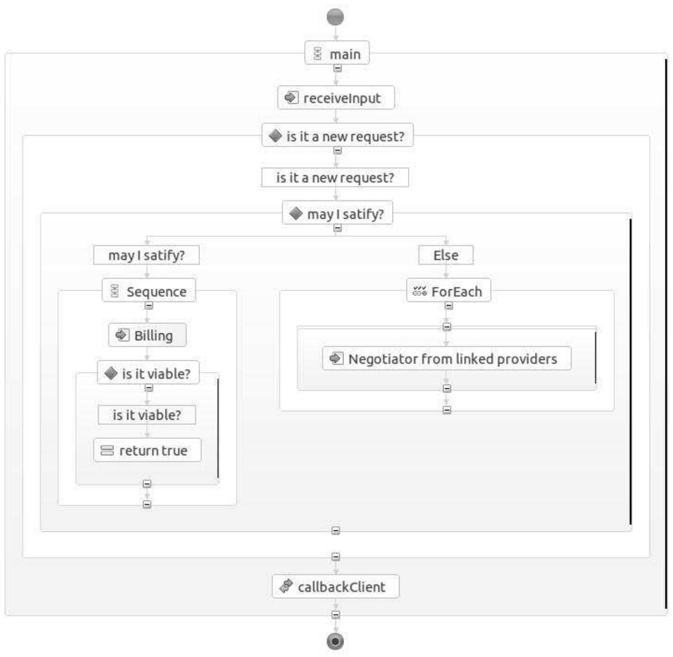

Fig. 5. Negotiator BPEL workflow

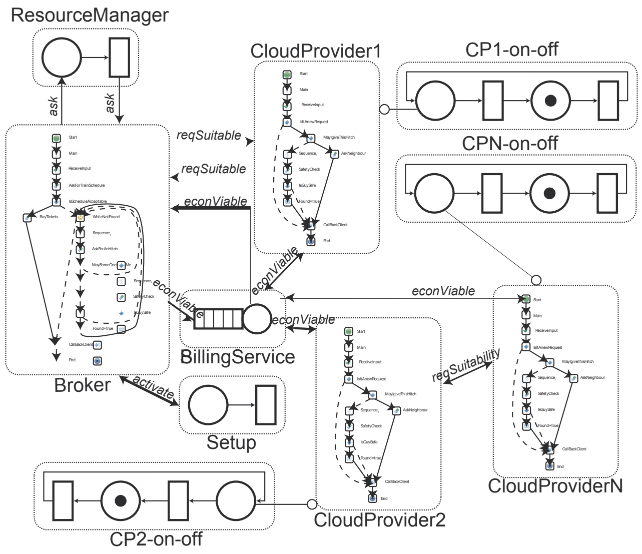

Fig. 6. The multiformalism model of the system

three providers can be available or not: this is obtained by modeling this condition by means of a GSPN model, that enables/disables the related providers model with an inhibitor arc of the ancillary formalism (that extends GSPN inhibitor arcs). Third party services *Resource Manager* and *Setup* are modeled by simple GSPNs, as the only relevant aspect of their presence in the model is related to the delays that they may introduce in the system, while *Billing* is modeled by a QN, as the queuing effect impacts on all the PerfBPEL CWS submodels, resulting in a mutual influence on their response time.

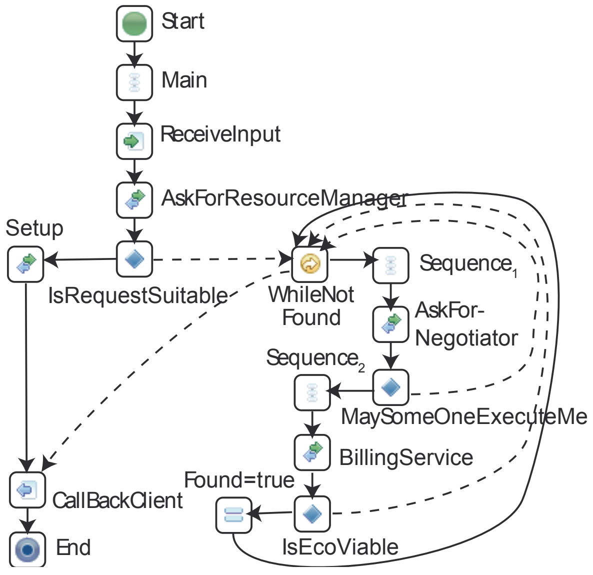

Fig. 7. Broker PerfBPEL description

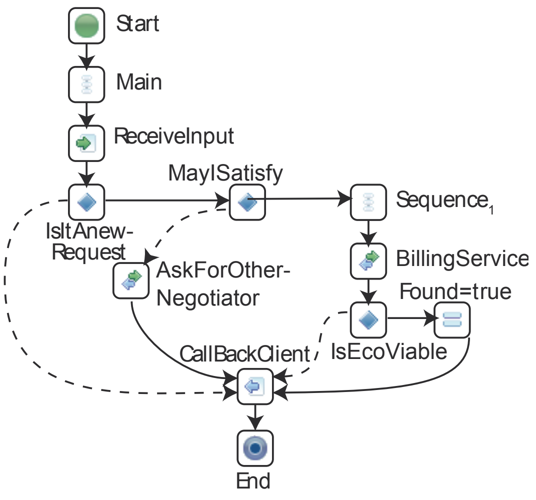

Fig. 8. Negotiator PerfBPEL description

C. Results

Table III shows the parameters used in the model, their meaning and nominal values chosen for the quantitative analysis.

The model has been analyzed in order to obtain two different metrics: the probability to get request accepted with respect of the variation of the *probBLOK* parameter (Fig. 9) and the probability to not have a (even negative) response with respect to the time the CPA spends away from terminal (Fig. 10). The probability of a positive response from the *Billing* is an upper bound to the probability of getting a request accepted: this ensures that the system is working correctly and that there is not the possibility of accept not economically viable requests. The response probability in Fig. 10 accounts for the cases in which all the providers are not available. As expected, the probability of not receiving an answer increases

TABLE III
PARAMETERS OF THE APPLICATION

Name	Meaning	Value
checkTime	time interval at the end of which CPA periodically checks external requests	1800 sec
busyRatio	fraction of time the CPA spends in back-office practice (and so is far from terminals)	1
decisionTime	time in which the CPA takes a decision about satisfying external requests	30 sec
billTime	time in which the *Billing* service computes a request	10 sec
rmTime	time in which the *Resource Manager* service computes a request	10 sec
stTime	time in which the *Setup* service computes a request	10 sec
netTime	generic transmission time of a message	3 sec
probRMOK	probability the *Resource Manager* server gives a good response	0.1
probBLOK	probability the *Billing* service gives a good response	0.7
probSAT	probability according to which the CPA decides to accept user request	0.7

with portion of time CPA spends away from the terminal. However it is interesting to see that the protocol is still capable of limiting the negative effect: with a large off period (240s) the probability is just around 30%.

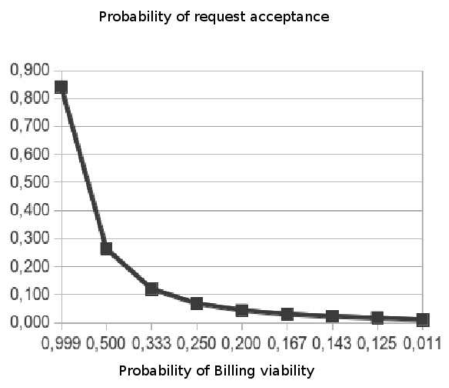

Fig. 9. Acceptance vs probBLOK

VI. CONCLUSIONS AND FUTURE WORK

The approach proposed in this paper provides an original contribution to the performance evaluation of systems based on SOA. This claim is motivated by the capacity of designing models including the effects of third-party services and other external factors, like users behavior. Although the work presented does not cover yet mutual influences of concurrent execution in Flow constructs, the scope considered is sufficient to evaluate real workflows. This approach benefits of the flexibility offered by the SIMTHESys modeling framework. Based on the possibility of defining the execution semantics of the elements of a formalism by means of the behavior mechanism, the PerfBPEL formalism will be extended to cover the missing aspects of BPEL execution semantics. Moreover, future work includes the application of different solution techniques in addition of a Markov chain based solving engine.

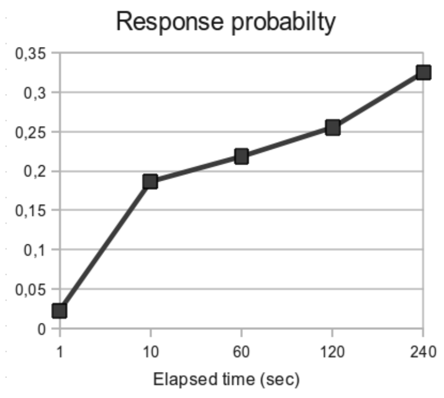

Fig. 10. Acceptance vs checkTime

VII. ACKNOWLEDGEMENT

This work has been partially funded by MIUR by PRIN 2008PXNBFZ-002 Cloud@home grant.

REFERENCES

[1] E. Barbierato, M. Gribaudo, and M. Iacono. Defining formalisms for performance evaluation with SIMTHESys. *Electr. Notes Theor. Comput. Sci.*, 275:37–51, 2011.
[2] E. Barbierato, M. Gribaudo, and M. Iacono. Exploiting multiformalism models for testing and performance evaluation in SIMTHESys. In *Proceedings of 5th International ICST Conference on Performance Evaluation Methodologies and Tools - VALUETOOLS 2011*. ICST, 2011.
[3] E. Barbierato, M. Gribaudo, M. Iacono, and S. Marrone. Performability modeling of exceptions-aware systems in multiformalism tools. In *18th International Conference on Analytical and Stochastic Modelling Techniques and Applications - ASMTA 2011*, pages 257–272, 2011.
[4] G. Ciardo, R. L. Jones, III, A. S. Miner, and R. I. Siminiceanu. Logic and stochastic modeling with smart. *Perform. Eval.*, 63:578–608, June 2006.
[5] G. Ciardo and A. S. Miner. Smart: The stochastic model checking analyzer for reliability and timing. *Quantitative Evaluation of Systems, International Conference on*, 0:338–339, 2004.
[6] T. Courtney, S. Derisavi, S. Gaonkar, M. Griffith, V. Lam, M. McQuinn, E. Rozier, and W. H. Sanders. The Möbius modeling environment: Recent extensions - 2005. *Quantitative Evaluation of Systems, International Conference on*, 0:259–260, 2005.
[7] T. Courtney, S. Gaonkar, K. Keefe, E. Rozier, and W.H. Sanders. Möbius 2.3: An extensible tool for dependability, security, and performance evaluation of large and complex system models. In *DSN*, pages 353–358, 2009.
[8] D. D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster. The Möbius framework and its implementation.
[9] X. Deng, Z. Lin, W. Cheng, R. Xiao, L. Fang, and L. Li. Modeling web service choreography and orchestration with colored petri nets. *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, ACIS International Conference on*, 2:838–843, 2007.
[10] G. Diaz, J. J. Pardo, M. E. Cambronero, V. Valero, and F. Cuartero. Verification of web services with timed automata. *Electron. Notes Theor. Comput. Sci.*, 157(2):19–34, May 2006.
[11] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.
[12] A. Alves et al. Web services business process execution language version 2.0. oasis standard, 2007.

[13] G. Franceschinis, M. Gribaudo, M. Iacono, S. Marrone, N. Mazzocca, and V. Vittorini. Compositional modeling of complex systems: Contact center scenarios in OsMoSys. In J. Cortadella and W. Reisig, editors, *ICATPN*, volume 3099 of *Lecture Notes in Computer Science*, pages 177–196. Springer, 2004.

[14] G. Franceschinis, M. Gribaudo, M. Iacono, S. Marrone, N. Mazzocca, and V. Vittorini. Compositional modeling of complex systems: Contact center scenarios in OsMoSys. In Cortadella, J. and Reisig, W., editor, *Applications and theory of Petri nets 2004, Proceedings*, volume 3099 of *LNCS*, pages 177–196, HEIDELBERGER PLATZ 3, D-14197 BERLIN, GERMANY, 2004. SPRINGER-VERLAG BERLIN.

[15] S.T. Gilmore and M. Tribastone. Evaluating the scalability of a web service-based distributed e-learning and course management system. In *WS-FM*, pages 214–226, 2006.

[16] S. Hinza, K. Schmidt, and C. Stahl. Transforming BPEL to Petri Nets. In *Proceedings of the 3rd Int'l Conference on Business Process Management (BPM 2005)*, pages 220–235. Springer Verlag, 2005.

[17] H.J.A. Holanda, J. Merseguer, G. Cordeiro, and A.B. Serra. Performance evaluation of web services orchestrated with WS-BPEL4People. *International Journal of Computer Networks & Communications*, 2:18, 11/2010 2010.

[18] M. Iacono, E. Barbierato, and M. Gribaudo. The SIMTHESys multiformalism modeling framework. *Computers and Mathematics with Applications*, (0):–, 2012.

[19] M. Iacono and M. Gribaudo. Element based semantics in multi formalism performance models. In *18th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems - MASCOTS 2010*, pages 413–416. IEEE, 2010.

[20] M. Iacono and S. Marrone. A performance driven modelling approach for soa based application. In *ESM '08: Proc. of European Simulation and Modelling Conference (ESMc)*, Le Havre, France, 2008. EUROSIS.

[21] C. Koch. A new blueprint for the enterprise. *CIO Magazine*, 2005.

[22] N. Lohmann. A feature-complete Petri net semantics for WS-BPEL 2.0. In M. Dumas and R. Heckel, editors, *Web Services and Formal Methods, Forth International Workshop, WS-FM 2007, Brisbane, Australia, September 28-29, 2007, Proceedings*, volume 4937 of *Lecture Notes in Computer Science*, pages 77–91. Springer-Verlag, 2008.

[23] N. Lohmann, H.M.W. Verbeek, C. Ouyang, and C. Stahl. Comparing and evaluating Petri net semantics for BPEL. *International Journal of Business Process Integration and Management*, 4(1):60–73, 2009.

[24] F. Martinelli and I. Matteucci. Synthesis of web services orchestrators in a timed setting. In *Proceedings of the 4th international conference on Web services and formal methods*, WS-FM'07, pages 124–138, Berlin, Heidelberg, 2008. Springer-Verlag.

[25] F. Moscato, F. Flammini, G. Di Lorenzo, V. Vittorini, S. Marrone, and M. Iacono. The software architecture of the OsMoSys multisolution framework. In *ValueTools '07: Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*, pages 1–10, 2007.

[26] M.P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: A Research Roadmap. *International Journal of Cooperative Information Systems*, 17(02):223+, 2008.

[27] A. Tahamtan, C. Oesterle, A.M. Tjoa, and A. Hameurlain. BPEL-TIME - WS-BPEL time management extension. In *ICEIS (3)*, pages 34–45, 2011.

[28] K. S. Trivedi. Sharpe 2002: Symbolic hierarchical automated reliability and performance evaluator. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, page 544, Washington, DC, USA, 2002. IEEE Computer Society.

[29] W.M.P. van der Aalst. The application of Petri Nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[30] V. Vittorini, M. Iacono, N. Mazzocca, and G. Franceschinis. The OsMoSys approach to multi-formalism modeling of systems. *Software and System Modeling*, 3(1):68–81, 2004.

[31] J. Xia and C.K. Chang. Performance-driven service selection using stochastic CPN. In *J. V. Atanasoff Int. Symposium on Modern Computing (JVA 2006)*, pages 99–104, Sofia, Bulgaria, 2006. IEEE.