A Framework for Verifying SLA Compliance in Composed Services

Hua Xiao[1], Brian Chan[2], Ying Zou[2], Jay W Benayon[3], Bill O'Farrell[3], Elena Litani[3], Jen Hawkins[3]

School of Computing[1]
Queen's University
Kingston, Ontario, Canada
huaxiao@cs.queensu.ca

Dept. of Electrical and Computer Engineering[2]
Queen's University
Kingston, Ontario, Canada
{ying.zou, 2byc}@queensu.ca

IBM Toronto Lab[3]
Markham, Ontario, Canada
{jayb, billo, elitani, jlhawkin}@ca.ibm.com

Abstract

Service level agreements (SLAs) impose many non-functional requirements on services. Business analysts specify and check these requirements in business process models using tools such as IBM WebSphere Business Modeler. System integrators on the other hand use service composition tools such as IBM WebSphere Integration Developer to create service composition models, which specify the integration of services. However, system integrators rarely verify SLA compliance in their proposed composition designs. Instead, SLA compliance is verified after the composed services are deployed in the field. To improve the quality of the composed services, we propose a framework to verify SLA compliance in composed services at design time. The framework re-uses information in business process models to simulate services and verify the non-functional requirements before the service deployment. To demonstrate our framework, we built a prototype using an industrial process simulation engine from IBM WebSphere Business Modeler and integrate it into an industrial service composition tool. Through a case study, we demonstrate that our framework and the prototype assist system integrators in composing services while considering the non-functional requirements.

1. Introduction

Web services permit enterprises to adapt to rapidly changing business requirements. However, the dynamic discovery and binding of services create great challenges in ensuring that the composed services can achieve various non-functional requirements, such as processing time, availability, and processing cost. Such requirements are derived from Service Level Agreements (SLAs) which stipulate contracts between consumers and providers of services. For example, an on-line ticket service should process requests within 3 seconds (i.e., processing time < 3s), and the cost for processing each request should be less than 1 dollar (i.e., cost < $1).

Developing SOA systems requires the involvement of a large number of individuals, such as business analysts, system integrators and software developers. All these individuals use various tools and industrial standards to specify, compose, and develop web services. In the business domain, business analysts specify *business process models* (or *process models*) and stipulate SLAs using modeling tools, such as IBM WebSphere Business Modeler (WBM) [8]. In the service composition domain, system integrators use service composition languages such as BPEL to create more detailed models for composing services. The detailed *service composition models* (or *composition models*) reflect the design of the *process models* as specified by the business analysts.

Business analysts commonly use a process simulator to verify SLA compliance for a *process model*. Using the simulator, the analysts can explore alternative models using different scenarios till the optimal model is picked. On the other hand, system integrators commonly wait for services to be fully composed and deployed before they can detect and correct SLA violations and design faults. A process monitor is used to monitor the execution of the deployed service and capture events generated from the running service. The problems in the design of the service composition are detected by analyzing the generated events. Such problems should be flagged earlier during the design and development phases of services.

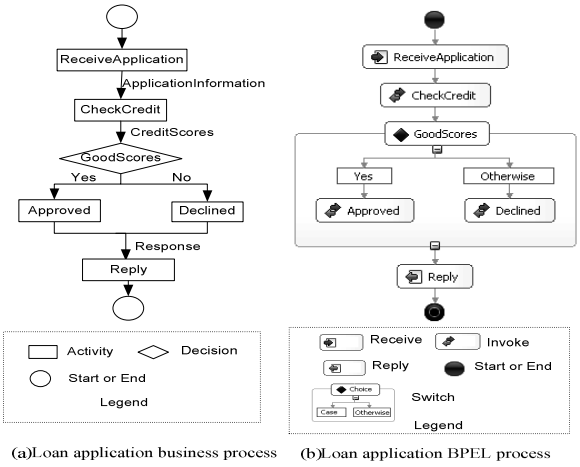
Due to the unavailability of the source code for services, testing techniques for services are limited to black-box testing which examines if a *composition model* is correctly specified and that services are accurately referenced and wired [13]. Although *composition models* (e.g., BPEL processes) are derived from *process models*, *composition models* only capture the functional (i.e., structural) aspects of the *process models* and do not encode the non-functional aspects. Non-Functional Attributes (NFAs) which describe the non-function requirements in the abstract *process models* are rarely relayed to the detailed *composition models* due to the lack of support for describing such attributes in composition languages, such as BPEL.

In this paper, we provide a framework which verifies SLA compliance of composed services. We bridge the gaps between the process modeling domain and the service composition domain to share NFAs and design evaluation tools. The framework uses lightweight techniques to annotate *composition models* with NFAs that are derived from *process models*. The annotated *composition models* are then simulated using process simulators. Such simulators are commonly used to

978-0-7695-3310-0/08 $25.00 © 2008 IEEE
DOI 10.1109/ICWS.2008.26

simulate *process models* instead of simulating *composition models*. Therefore, we have to enhance such simulators to account for the peculiarities of *composition models*. Our work helps in flagging SLA violations of services at design time before the deployment of these services. Using our framework, system integrators can optimize their *composition models* and avoid SLA violations.

The rest of the paper is organized as follows: Section 2 gives an overview of the business processes modeling and service composition. Section 3 discusses our framework for verifying SLA compliance in a *composition model*. Section 4 presents our case study. We evaluate the effectiveness of our prototype using industrial tools. Section 5 reviews the related work. Finally, Section 6 concludes the paper and discusses future work.



(a)Loan application business process (b)Loan application BPEL process

Figure 1. A *Loan Application* process

2. Process Modeling and Service Composition

In this section, we give an overview of business process modeling and service composition.

Business Process Modeling

A *process model* is a set of linked tasks to realize a business objective [18]. *A process model* specifies tasks, control flows, data flows, and resources. Tasks are the operations to be performed in order to achieve business objectives. For example as shown in Figure 1.(a), a *loan application* business process contains tasks, such as *ReceiveApplication*, *CheckCredit*, and *Approved*. A sub-process describes a subset of tasks to be reused in different contexts. Control flows determine execution paths of a process following various control nodes. The control nodes include sequence, loop, merge (i.e., *OR-Join*), join (i.e., *AND-Join*), fork (i.e., *AND-Split),* and decision (i.e., *OR-Split*) [18]. For example, shown in Figure 1.(a), *GoodScores* is an *decision* control node. Data flows present the input and output of tasks. An instance of data flow is the data, *ApplicationInformation*, which travels from *ReceiveApplication* task to *CheckCredit* task. NFAs and the values are annotated in the corresponding task specification.

Process models are often described in proprietary formats used by particular process modeling tools. For example the IBM WBM tool describes *process models* using the Business Object Models languages. Business processes can also be specified using standards, such as XPDL (XML Process Definition Language) [19].

Service Composition

A service composition describes the implementation of a business process through the composition of various services. Service compositions are described using block description and directed graph based description. Block descriptions are inherited from XLANG [16], which does not have explicit control flows but provides structures to describe the flow of control. The directed graph description comes from WSFL [17] and uses a graph to describe the tasks and their interactions.

BPEL is a commonly used standard for describing a service composition. It supports both description techniques. Similar to *process models*, entities in a *composition model* generally includes: activities, data flows, and control flows. Activities describe the fundamental behavior for handling services requests and invoking services. The basic activity includes *receive*, *reply*, *invoke*, *assign*, and *wait*. In Figure 1.(b), *ReceiveApplication* is a *receive* activity which receives requests from an external service. *CheckCredit, Approved* and *Declined* are *invoke* activities which invoke the corresponding services. Similar to sub-processes in a *process model*, *scope* encodes a unit of activities and control nodes. Data flows are messages exchanged among services. The control flows determine the possible execution paths, including *sequence*, *switch*, *while*, *pick* and *flow*. In Figure 1.(b), *GoodSores* is an example of *switch,* which directs the execution flow to one of the possible activities: *Approved* activity or *Declined* activity.

Comparing to *process models* which capture the functional and non-functional requirements of a service, BPEL conveys implementation details. For example, BPEL provides extensions to encapsulate Java code in the service composition. It also enables the declaration of local variables in activities, and sets conditional expressions to control the service execution. However, NFAs of entities are not supported in composition languages. This lack of support limits the ability of integrators in verifying SLA compliance in created *composition models*.

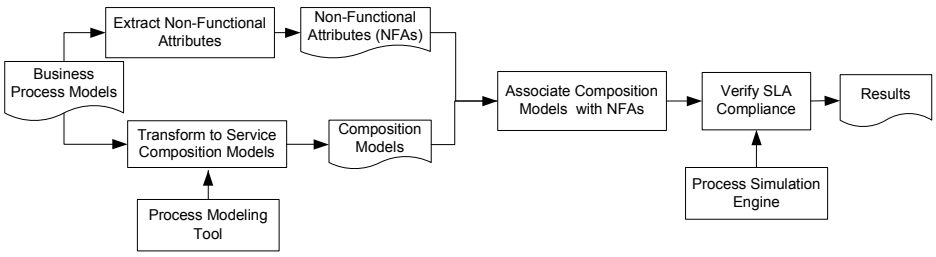


Figure 2. A Framework for verifying SLA compliance in *composition models*

3. Framework for Verifying SLAs in *Composition Models*

We propose a framework that analyzes a *composition model* and uses information from the *process model* to evaluate compliance for SLAs in the *composition model*. The major steps in the framework are specified in Figure 2. A service is directly composed by a system integrator who may have limited knowledge of the NFAs that are critical to measure SLA compliance. To support the verification of SLA compliance, we make use of the NFAs that are annotated by business analysts in the original *process models*.

A *process model* can be automatically transformed to a *composition model* using process modeling tools, such as the WBM. Each task in the *process model* is annotated with the NFAs. To avoid losing NFAs and their default values during the transformations, we provide techniques to maintain the associations between NFAs and their corresponding entities in the *composition model*.

Instead of developing a new tool to analyze the *composition models*, we leverage the powerful capability of commercial business process simulators. However, a process simulator is not designed to analyze *composition models*. We design and develop a wrapper which converts entities in a *composition model* to match with the interface of the process simulator. Meanwhile, the transferred NFAs for a *composition model* are provided as input to the process simulator. As a result, the process simulator generates qualitative results to evaluate the SLAs delivered in the *composition model*. For example, the overall execution time for a composed service and the probability of each executed path are provided to the system integrators. Such qualitative results help the integrator in assessing alternative designs and optimizing them in order to meet the SLAs. In this ever changing business domain, a system integrator needs to frequently update *composition models*. The proposed framework helps ensure that updated models achieve the business objective by providing feedback on the impact of the changes. In the following sub-sections, we discuss the major components of our framework.

3.1 Extracting NFAs

We parse the XML representation of *process models*, and extract the NFAs annotated with each entity in the models. We extract the NFAs and the names of the corresponding tasks. We capture three types of NFAs, which are used to verify the achievement of SLAs: time, cost and resource.

Specifically, time describes the temporal constraints on performing a task. For example, the completion time stipulates the amount of processing needed for accomplishing a task or a process instance (i.e., an instance of a *process model* in the run-time). The wait time represents maximum queuing delay that a task is queued for obtaining a resource (e.g., available service) before the task fails.

Cost specifies the requirements on expenses. For example, the processing cost is applied to each time a task is performed. Idle cost measures the expense when a task is idle and waiting for a resource to become available. Revenue is generated by completing a task.

Resources specify the items (e.g., personnel, equipment, or materials) required for executing a task. For example, *CheckCredit* task is automatically executed. Timetables are used to describe the availability of resources. Other resources describe the quality of the services, such as security levels and reliability of the services.

A *process model* may have multiple execution paths due to the control structures (e.g., decision node). The achievement of SLAs is pertinent to the selection of execution paths in a *process model*. For example, some paths may take longer time to complete than others. To evaluate the NFAs of a process, the execution of a process can be evaluated in three ways: randomly pick a single path to execute, select a path based on the probability of the execution, and select a path using conditional expressions. When a path is frequently executed, it becomes a critical path, which has a significant impact on the overall quality of a composed service. Therefore, we extract the probability and expressions annotated in the control nodes for *process models* and transfer this information to the corresponding entities in the *composition models*.

To improve the analysis accuracy of *composition models*, the actual values of the NFAs could also be gathered from the black-box testing tool in a service composition environment, the historical data collected from the past deployment and run-time logs. For example, the black-box testing tool often provides the possible

values of the variables defined in activities in a *composition model*. The values of the variables could be used to determine the conditional expressions to select possible execution paths. Therefore, we provide a NFA editor in a service composition environment to allow a system integrator to enter the estimated values for the NFAs in a *composition model*.

3.2 Associating *Composition Models* with NFAs

To associate an extracted NFA and its default value to an appropriate entity in the *composition model*, we need to link entities in both models: *process model* and *composition model*. We map process entities to composition entities which have similar semantic meaning. Table 1 shows the mappings between process entities and composition entities.

Table 1. Process models vs. composition models

Process Model Entity	Composition Model Entity	Non-Functional Attributes
Task	Invoke	Cost, Time, Resources, Expression
Data	Variables	Initial value, Range
Sub-Processes	Scope	N/A
Merge, Join, Fork	Empty/Assign	N/A
Sequence	Sequence	N/A
Parallel	Flow	N/A
Decision	Switch-Case	Probability, Expression
While	Loop	Probability, Expression

As shown in Table 1, it is straightforward to map the tasks and control structures (e.g., sequence, parallel) in *process models* to the counterparts in the *composition model*. The mappings transfers the NFAs attached to a process entity to the corresponding composition entity. For example, the mappings between the control structures transfer the probability of different alternative execution paths in a *process model* to the corresponding execution paths in the *composition model*.

In the *process models*, data is explicitly specified in the input and output of a task. For example in the *loan application* as shown in Figure 1.(a), the data, *ApplicationInformation* is specified as an input data and the *creditScores* is the output of *CheckCredit* task. The data types, initial values and value ranges of the data are described by a business analyst and stored in the *process model* specifications. However, such data information is never relayed to the *composition model*. In a *composition model*, some variables are explicitly specified as messages to the subsequent activities. For example, as shown in Figure 1.(b), *creditScores* variable is transferred as the output of *CheckCredit* activity and the input variable of *GoodScores* activity. Other variables could be defined in *CheckCredit* activity, but are not flowed into other activity. To retain the data information in the *process model*, we map the data in a *process model* to the variables which are flowed between the consecutive activities in the converted *composition model*.

If a merge, join or fork node in a *process model* combines several input data into one output data, we interpret such nodes as an *assign* type, which has the equivalent functionality as merging multiple variables into one in the *composition model*. Otherwise, a merge, join or fork node is translated to an *empty* activity which simply distributes input variables to output variables without any computation in the *composition model*.

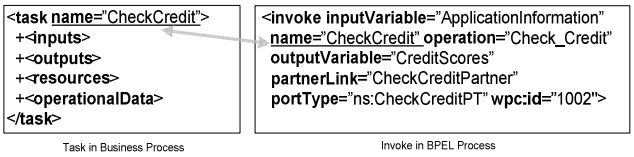


Figure 3. An example mapping

For each process entity, we extract the associated NFAs and store the NFAs in a separate XML document. We want to ensure that the *composition models* conform to standard specification languages which do not support the description of NFAs. To verify SLA compliance in the *composition models*, the NFAs extracted from the process entities need to be assigned to the corresponding composition entities. In a *process model*, each task and control node (e.g., fork, decision, and join) is assigned a unique name. The unique naming allows us to establish a one-to-one mapping from process entities to composition entities. When transforming a *process model* to a *composition model*, the names are transformed as attributes in the *composition model*. For example as shown in Figure 3, the *CheckCredit* task is converted to the equivalent entity, *invoke* activity following the mappings specified in Table 1. The task name (i.e., *CheckCredit*) is retained as the value of "*name*" attribute in the *invoke* activity.

However, the transformed *composition model* can be further customized. For example, the name of an activity can be changed. Therefore, it is not sufficient to rely on naming to attach the extracted NFAs to the composition entities. To address this issue, we extend each entity in a *composition model* with a unique identifier throughout the service composition environment. For example as shown in Figure 3, each activity is extended with an unique identifier attribute (*WPC:id*). Using the initial name mappings between a process entity and a composition entity, we assign the extracted NFAs of a process entity to the composition entity of the same name. The NFAs track the identifier (i.e., *WPC:id*) of the associated entity in the *composition model*. When the *composition model* is edited, we use the identifier to keep the composition entity and the associated NFAs in sync. For example,

when the location of an entity in the *composition model* is changed, it would not affect the related NFAs. When a new entity is added to a *composition model*, a new identifier is assigned to the new entity. If an entity is deleted, its related NFAs are also deleted.

3.3 Verifying SLA Compliance in *Composition Models*

To verify SLA compliance and to detect design faults in a composed service without having to deploy it into the run-time environment, we integrate a commercial simulation engine into a service composition environment. However, the simulation engine is used to evaluate *process models* and therefore does not support service composition languages. In particular, NFAs are not normally encoded in a *composition model*. To bridge the gap between the *composition models* and the interface of the simulator, we design and develop a wrapper that gathers the information from the *composition model* and converts the information to match the expected inputs of the simulator. The architecture of the wrapper is illustrated in Figure 4. Once a process simulator is integrated into a service composition environment, the process simulator can be also used to analyze a new *composition model* created in a service composition environment without an initial *process model*.

In the simulator, a run-time task object is instantiated for each task in a *process model*. Such a task object encapsulates values for the NFAs of that task. A converted entity in a *composition model* contains all the information needed by a task object, but in a different format. The specification for *composition models* organizes the entities in a tree structure where every node in the tree contains specific information for activities in *a composition model*. Moreover, the NFAs for an entity in a *composition model* are kept in a separate file as discussed in Section 3.2. The wrapper sets the input required for the simulation engine and produces the result for the analysts. A wrapper contains four major components:

The *Merger* component extracts NFAs from the NFA files and combines the NFAs with the corresponding activity in the *composition model*. Each activity has its own NFAs, such as execution time and cost. We keep track of the ordering sequence and input/output dependencies among the activities.

The *Dispatcher* component instantiates a run-time task object for the simulation engine and uses individual activity information to initiate the internal state of the task object. For example, the dispatcher invokes the setter methods (e.g., task.setName("Task Name")) defined in the interface of the task object to map the task object to an activity in a *composition model*. Moreover, the dispatcher links the task objects to reflect the data and control dependencies among the activities in the *composition model*. A link connects the output of a task object to the input of the subsequent task object. These links describe the possible execution paths taken by the simulation engine. Once the task objects are linked, a process object is formed as a unit of simulation.

The *Analyzer* component captures events at runtime. For example, the execution time for a process object can be triggered as an event after a trial run of all the task objects in the process object. The *Analyzer* gathers such events to analyze the design of the service composition, and evaluate performance metrics to verify SLA compliance.

The *Result Viewer* component notifies the service composition environment with the result of the analysis. The wrapper ensures that GUI shows the results of the updated analysis.

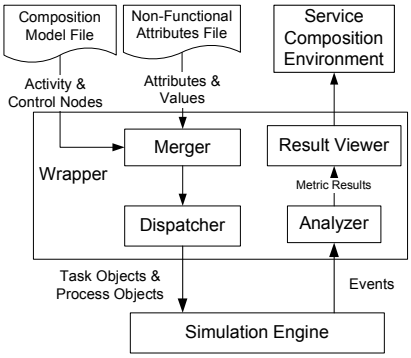


Figure 4. Architecture for verifying SLA compliance

4. Case Study

To evaluate the feasibility of our proposed framework, we design and develop a prototype that analyzes service composition described in BPEL. We use examples to demonstrate that our prototype helps a system integrator verify SLA compliance and optimize a service composition to meet the non-functional requirements imposed by SLAs.

4.1 Prototype Implementation

Process models are created and simulated using the IBM WBM. After simulation, the NFAs for the process entities are annotated into the *process model* specification. WBM supports the automatic conversion of a *process model* to a *composition model* in BPEL. The BPEL process is visualized and edited in the IBM WebSphere Integration Developer (WID), an IBM service composition environment. However, the NFAs for the *process model* are not transformed. We develop a prototype that uses the mapping schema presented in Section 3.2 to associate the NFAs of a process entity to the corresponding composition entity in BPEL. To verify SLA compliance, we migrate the WBM business process simulator and integrate it into the WID as an Eclipse plug in. Figure 5 depicts a screenshot of the prototype as a plug-in in the IBM WID.

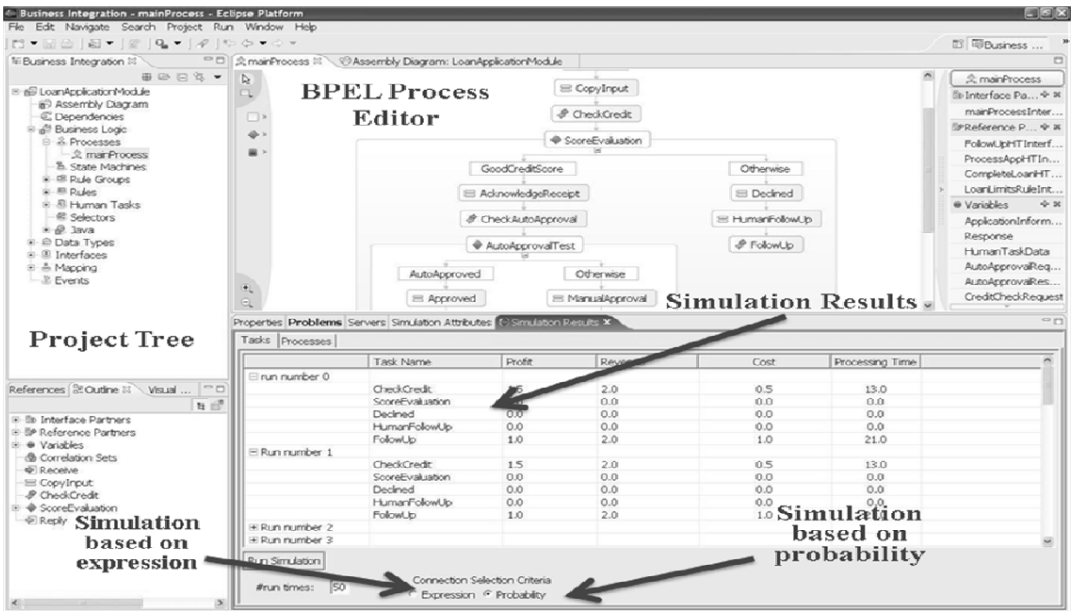


Figure 5. Annotated screenshot for simulating a BPEL process in IBM WID

As shown in Figure 5, the BPEL process editor in the WID visualizes a *composition model* for a system integrator to modify and analyze. We developed a NFA editor to list the extracted NFAs for each composition entity in BPEL. The default value for each NFA is extracted from a *process model* when a *composition model* is transforms from the *process model*. A system integrator can modify the values of NFAs. When a *composition model* is created in the WID, we list all the NFAs extracted from the WBM for each composition entity in WID. The system integrator provides the values for NFAs.

A *composition model* can be analyzed using the probability of execution paths. For instance, in Figure 5, the decision node *ScoreEvaluation* has two output branches: "Yes" branch and "No" branch. Suppose "Yes" branch has 80% probability to be executed and "No" branch has 20% probability to be executed. The simulator would choose an output branch based on those probabilities. Moreover, the decision node could be expressed using conditional expressions with variables. For example, when the decision node uses an expression (i.e. if(CreditScores>1000)), the simulator uses a boolean output to choose an output branch. If the expression "scores > 1000" evaluates to true, then the simulator goes to "Yes" branch; otherwise, it goes to "No" branch. Branches governed by expressions are dynamically selected based on the values of variables while probabilities are statically configured.

4.2. Application Examples

We choose the *loan application* [10] in BPEL provided by the WID to demonstrate the uses of our prototype. The *loan application* process describes the steps for applying for a loan. Once an applicant's information is received, the applicant's credit is checked. If the credit score is low, the process declines the application. Otherwise, the process checks the loan amount to decide whether the application is approved automatically or manually. *Loan application* process has 17 activities and 20 control nodes. We extract six types of NFAs which are defined in the WBM as follows: processing time, processing cost, Revenue, startup cost, waiting time cost; probability.

4.2.1 Evaluating Performance Metrics

We use the prototype to compute four performance metrics for verifying SLA compliance: average processing time, average revenue, average cost and average profit for a process. The average processing time calculates the average time of executing different paths. Similarly, the average profit, the average revenue and the average cost are defined to measure profit, revenue and cost of the process under different conditions.

To obtain the results for the aforementioned metrics, the prototype runs the *loan application* process 50 times using the probabilities provided by a business analyst. The 50 iterations ensure that all paths are considered by the simulator. As shown in Figure 6, there are three possible execution paths. As the result of the simulation, the simulator runs 42 times on path #1, 5 times on path #2, and 3 times on path #3, as listed in Table 2. The results for the four metrics of each execution path are listed in Table 2. To evaluate the average processing time, the probability of each path is considered. For example, the prototype calculates the average processing time of the composed services as follows:

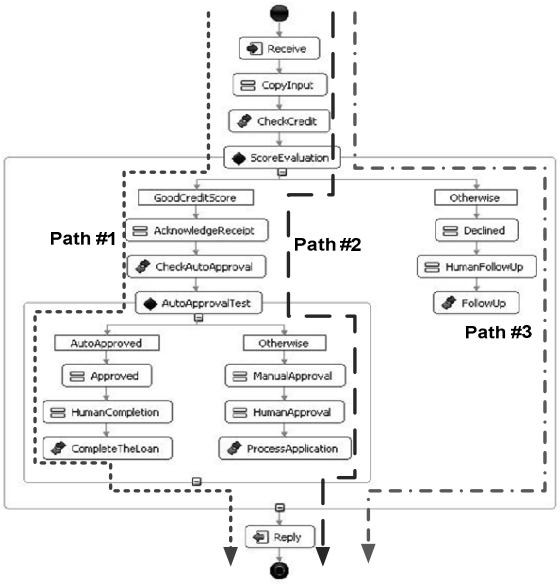


Figure 6. *Loan Application* process

Table 2. Results of performance metrics

Path	Iterations	Prob.	Processing Time (Min)	Revenue (USD)	Cost (USD)	Profits (USD)
1	42	84%	47	9.0	4.0	5.0
2	5	10%	67	11.0	5.5	5.5
3	3	6%	34	4.0	1.5	2.5
Average Result			48	7.9	4	4.9

$$M_{average\_process\_time} = 47 \times 84\% + 67 \times 10\% + 34 \times 6\% \approx 48(Min)$$

Most of composed services involve human tasks, which require humans to manually conduct the activity. For instance, the *loan application* process has three human tasks: *HumanCompletion*, *HumanApproval*, and *HumanFollowUp*. The human task poses great challenges to verify SLA compliance of a composed service in a run-time environment. In particular, to evaluate the overall processing time, a loan manager needs to be involved to manually fulfill the required activity. However, it is a tedious job for the loan manger to conduct the activity tens of times. Our prototype provides a lightweight solution to verify SLA compliance in a composed service. It allows a system integrator to provide estimates on the values of the NFAs of human tasks and modify any NFAs of a composition entity using the attribute editor. The overall performance of the composed service is automatically computed.

4.2.2 Optimizing Service Composition

A system integrator uses the prototype to assess the performance metrics for each alternative and selects an optimal design. Using the qualitative results from the prototype, a system integrator can identify the critical path for executing the composed service, and optimize the design to improve the performance of the critical path.

Critical path is the path with the highest probability of being executed. The performance of the activities in the critical path would greatly impact the general performance of the process. A critical path is derived by analyzing process instances following different execution paths. In Table 2, path #1 is the critical path since it is executed 42 times out of 50. If a system integrator needs to optimize the average processing time of the service, he/she should focus on optimizing the activities in the critical path.

Redundant activities are two or more activities which provide similar functionality in a critical path. The redundant activities in a critical path could lead to a longer proceeding time. For example as shown in Table 2, path #1, as a critical path, takes 47 minutes to complete. Although it is not the longest runtime, ideally a critical path should be as time efficient as possible. In Figure 6 *CheckCredit* activity and *CheckAutoApproval* activity in path 1 both access a user's private information. Therefore, its functionality is redundant and causes additional processing time in the critical path. A more efficient method is to merge the two activities into one in order to reduce the number of times for accessing the private information. Once the credit has the good score, the *AutoApprovalTest* decision is triggered. The optimization reduces one decision node *ScoreEvaluation* and two activities in path #1 (i.e., *AcknowledgeReceipt* and *Check AutoApproval*). After merging, the new average processing time would be reduced by 9 minutes with this optimization.

5. Related work

Research on verifying business requirements in the *composition models* is divided into two major directions. One direction focuses on transforming BPEL processes into various formal models and checks the functional behaviors of the composed services [4]. Qian et al. [15] use the timed automata to verify functional properties such as reachability, and activity dependency in BPEL processes. Foster et al. [6] transform BPEL processes to finite state processes notation. Yu et al. [20] use a language named PROPOLS to describe the temporal logic in a BPEL process. Duan et al. [5] provide a framework to specify the semantics of tasks and control flows defined in BPEL. Mayer and Lubke [13][14] propose a layer-based framework to test BPEL processes. Different from these approaches, our work focuses on verifying SLA compliance of composed services.

Another research direction aims to check the Quality of Services (QoS) of BPEL processes. Koizumi and Koyama [12] propose a performance model using logs to estimate the processing execution time. Kazhamiakin et al. [11] use a finite state machine model to formulize time related properties and analyze temporal requirements, such as the duration of a process. Fung et al. [7] propose a message tracking model to support QoS measurement in

BPEL processes. Barbir et al. [3] discuss the main security requirements for web services, and test the security of web services using web services middleware. These approaches focus on either assessing one type of non-functional requirements (e.g., process execution time) or constructing their tool using the run-time data. These approaches also require the deployment of the services in the field before identifying the violations to the desired QoS. Our work uses a lightweight solution to verify multiple NFAs of the *composition model* without the deployment. Our work improves the productivity for developing web services.

6. Conclusion and Future Work

In this paper, we present a framework that verifies SLA compliance and designs of *composition models*. We bridge the gaps between the process modeling domain and the service composition domain to share NFAs and design evaluation tools. We present a lightweight solution that provides qualitative feedback for a system integrator to optimize the service composition. Through a case study, we demonstrate the feasibility of our approach. Currently, the prototype verifies the SLAs common to *process models* and *composition models*. In the future, we plan to extend the simulation engine to handle more NFAs that are not considered in the process simulator, such as reliability, security and reputation of a service.

Acknowledgment

We would like to thank Mr. Gary Tang and Mr. Casey Best at IBM Toronto Laboratory for their valuable feedback to this research. This research is supported by Natural Sciences and Engineering Research Council of Canada, and IBM Canada Centers for Advance Studies.

Trademarks

References

[1] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, et al., "Business Process Execution Language for Web Services Version 1.1," 2003.

[2] A. Alves, A. Arkin, S. Askary, C. Barreto, et al., "Web Services Business Process Execution Language Version 2.0," 2007.

[3] A. Barbir, Ch. Hobbs, El. Bertino, F. Hirsch, L. Martino, "Challenges of Testing Web Services and Security in SOA Implementations," In book *Test and Analysis of Web Services*, Springer, 2007, pp. 395-440.

[4] F. van Breugel and M. Koshkina, "Models and Verification of BPEL," *Working Paper,* 2006.

[5] Z. Duan, A. Bernstein, P. Lewis, and S. Lu, "Semantics Based Verification and Synthesis of BPEL4WS Abstract Processes", in *Proc. of the IEEE International Conference on Web Services (ICWS'2004)*, San Diego, CA, USA, July, 2004, pp. 734-737.

[6] H. Foster, S. Uchitel, J. Magee and J. Kramer, "Model-based verification of web service compositions," *18th International Conference on Automated Software Engineering*, 2003, pp. 152-161.

[7] Casey K. Fung, Patrick C. K. Hung, Guijun Wang, Richard C. Linger and Gwendolyn H. Walton, "A Study of Service Composition with QoS Management," in *2005 IEEE International Conference on Web Services (ICWS'05)*, Orlando, Florida, USA, July 12-15, 2005

[8] IBM WebSphere Business Modeler, Available at: http://www-306.ibm.com/software/integration/wbimodeler

[9] IBM WebSphere Integration Developer, Available at: http://www-306.ibm.com/software/integration/wid

[10] IBM WebSphere Integration Developer Sample: Loan application, Available at : http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/topic/com.ibm.wbit.sample.appl.3.doc/loanapplication/pdf/loanapplication.pdf

[11] R. Kazhamiakin, P. Pandya, M. Pistore, "Representation, Verification, and Computation of Timed Properties in Web Service Compositions," In *proceeding of International Conference on Web Services (ICWS) 2006*, Chicago, USA, Sept. 18-22, 2006.

[12] S. Koizumi, K. Koyama, "Workload-aware Business Process Simulation with Statistical Service Analysis and Timed Petri Net," In *proceeding of International Conference on Web Services (ICWS) 2007*, Salt Lake City, Utah, USA, July 9-13, 2007.

[13] D. Lübke, "Unit Testing BPEL Compositions," in Book *Test and Analysis of Web Services*, Springer, 2007, pp. 149-171.

[14] P. Mayer, D. Lubke, "Towards a BPEL Unit Testing Framework", in *Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications*, Portland, Maine, 2006.

[15] Y. Qian, Y. Xu, Z. Wang, G. Pu, H. Zhu and C. Cai, "Tool support for BPEL verification in ActiveBPEL engine," in *Proceeding of 18th Australian Software Engineering Conference(ASWEC),2007*. Australian 2007, pp. 90-100.

[16] S. Thatte, "XLANG Web Service for Business Process Design," 2001. Available at: http://xml.coverpages.org/XLANG-C-200106.html

[17] Web Services Flow Language (WSFL), Available at: http://xml.coverpages.org/wsfl.html

[18] Workflow Management Coalition, "Workflow Management Coalition Terminology & Glossary," available at: http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf

[19] Workflow Management Coalition, "Workflow Process Definition Interface—XML Process Definition Language," available at: http://xml.coverpages.org/XPDL20010522.pdf

[20] J. Yu, T. P. Manh, J. Han, Y. Jin, et al., "Pattern Based Property Specification and Verification for Service Composition," *Web Information Systems-WISE 2006,* vol. 4255/2006, 2006.