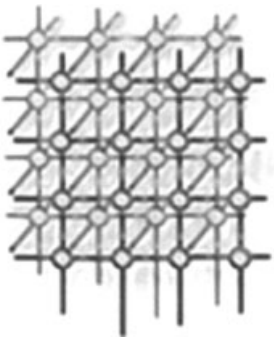
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2008; **20**:315–330
Published online 13 July 2007 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.1221



Towards workflow simulation in service-oriented architecture: an event-based approach

Yanchong Zheng, Yushun Fan and Wei Tan*, †

National Engineering Research Center for Computer Integrated Manufacturing Systems, Department of Automation, Tsinghua University, Beijing 100084, China

SUMMARY

The emergence of service-oriented architecture (SOA) has brought about a loosely coupled computing environment that enables flexible integration and reuse of heterogeneous systems. On building a SOA for application systems, more and more research has been focused on service composition, in which workflow and simulation techniques have shown great potential. Simulation of services' interaction is important since the services ecosystem is dynamic and in continuous evolution. However, there is a lack in the research of services' simulation, especially models, methods and systems to support the simulation of interaction behavior of composite services. In this paper, an enhanced workflow simulation method with the support of interactive events mechanism is proposed to fulfill this requirement. At build time, we introduce an event sub-model in the workflow meta-model, and our simulation engine supports the event-based interaction pattern at run time. With an example simulated in the prototype system developed according to our method, the advantages of our method in model verification and QoS evaluation for service compositions are also highlighted. Copyright © 2007 John Wiley & Sons, Ltd.

Received 23 March 2007; Accepted 1 April 2007

KEY WORDS: service composition; workflow simulation; event; data correlation

1. INTRODUCTION

Service-oriented architecture (SOA) is gaining increasing momentum in many domains such as enterprise information systems, software architecture and grid computing. SOA promises to

*Correspondence to: Wei Tan, National Engineering Research Center for Computer Integrated Manufacturing Systems, Department of Automation, Tsinghua University, Beijing 100084, China.
†E-mail: tanwei@mails.tsinghua.edu.cn

Contract/grant sponsor: National Science Foundation of China; contract/grant number: 60674080
Contract/grant sponsor: China National High Technology R & D project 'Business Coordination System Base on SOA'

provide a decentralized and loosely coupled environment that enables flexible, reliable and coordinated integration of dynamic applications belonging to different organizations. Furthermore, more and more companies are starting to organize their business processes by means of service aggregation; therefore, the importance of service composition has been widely recognized. Since service compositions can be described as workflow models, it is natural to apply workflow technology to automated service composition in the service-oriented paradigm, and there is much research work on that topic [1–3].

Many studies have been devoted to the design, verification and performance analysis issues related to Web service composition. Based on the XML Process Definition Language (XPDL), a model of Web services workflow is proposed [4], thus bringing workflow into the web environment. By employing Web service technology in the interaction, monitoring and control of process execution, Li and Lu [5] proposed a framework for modeling and reusing workflows as sub-workflows in service composition. Business Process Execution Language for Web Services (BPEL4WS) [6] defines an interoperable integration model which facilitates the expansion of automated process integration, and currently BPEL4WS is the *de facto* standard, supported by major companies in this area. Zhao and Liu [7] studied the modeling of organization centered workflows and their realization in the Web service environment *via* mapping to BPEL4WS. Particularly, Casati and Shan [8] developed a model and architecture that employed the element of events to achieve dynamic interaction between composite services; Wang *et al*. [9] proposed an ECA-rule-based method for end users to compose Web services conveniently; Guo *et al*. [10] introduced Pi-calculus to address the protocol level deadlock in grid workflows; Chandrasekaran *et al*. [11] explicated the power of simulation as a part of Web service composition and process design; and Chang *et al*. [12], Song and Lee [13] both utilized simulation techniques to evaluate service composition based on their QoS properties.

The research efforts have demonstrated the strength of workflow and simulation techniques in the design and performance analysis of service composition; however, to the best of our knowledge, few of them have made substantial investigations in the implementation mechanisms for simulating the actual behaviors of composite services. In other words, they neglect the interaction of internal service nodes between different services, and this interaction will probably affect the correctness and performance of composite services. In order to address this issue, we proposed an interactive-event-based workflow simulation method in this paper. Our main contribution is that, with profound analysis on the core mechanisms—the internal event interaction and data correlation, we made simulation techniques applicable to a loosely coupled environment like that of service-oriented computing.

The rest of the paper is organized as follows. In Section 2, a motivating example is presented. In Section 3, we give the workflow meta-model which serves as a foundation for performing simulation in a service-oriented environment, and we explain briefly the major components within the meta-model. Then, Section 4 elaborates on the core mechanisms of the workflow simulation method—the internal event communication and corresponding data correlation, from both the build-time and run-time perspectives. Section 5 covers the system architecture and relevant application programming interfaces (APIs), and Section 6 gives important analysis results based on the information we gathered through simulating the motivating example in the prototype system. Finally, conclusions and future research directions are given in Section 7.

DOI: 10.1002/cpe

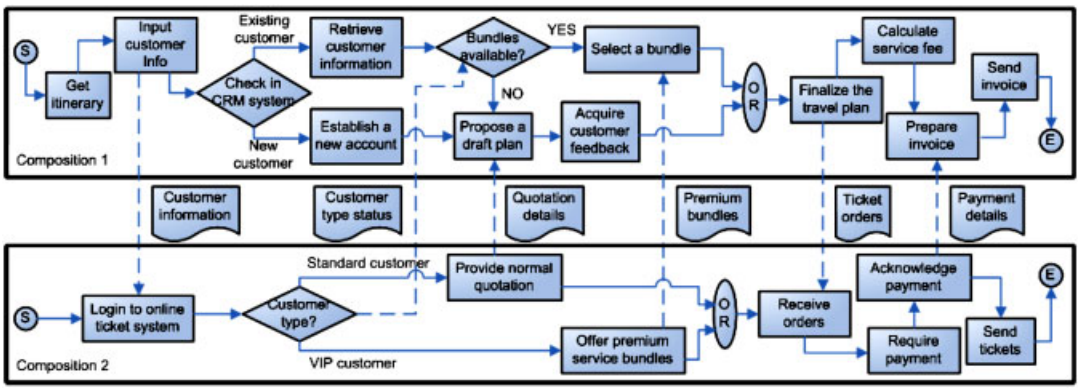


Figure 1. A travel planning example.

2. A MOTIVATING EXAMPLE

Without loss of generality, now consider two service compositions that have internal interaction as shown in Figure 1. It depicts a travel planning example including two separate composite service processes that work jointly to accomplish planning requests from customers of the travel agency. Composition 1 describes the workflow process for itinerary planning, while composition 2 is the online ticket handling process. When the agency receives initial itinerary from its customer, it will send the customer information to an independent online ticket system to acquire either quotations or bundles, contingent on the customer type specified by the online system. Simultaneously in the agency process, it will follow different procedures for existing or new customers to work out a travel plan.

Note that there is plenty of data exchange between the internal service nodes of these two compositions, which will affect the accomplishment of both workflows. For instance, the status of the decision node 'Customer type?' in composition 1 determines which route to go after the decision node 'Bundles available?' in composition 2, as bundles are only applicable to VIP customers of the ticket system. Such control logic can hardly be modeled or simulated with traditional techniques, thus arousing a need for specific supplements to the original meta-model as well as simulation mechanisms.

3. THE WORKFLOW META-MODEL

In order to ensure the interoperability between heterogeneous workflow systems and the efficient integration with other applications, we establish our workflow meta-model as an extension of the meta-data model presented by the Workflow Management Coalition (WfMC) in its Workflow Process Definition Interface—XML Process Definition Language (Interface one: XPDL) [14], enabling event communication in service computing environment by incorporating event elements. Figure 2 shows the static structure of the meta-model via the UML class diagram.

DOI: 10.1002/cpe

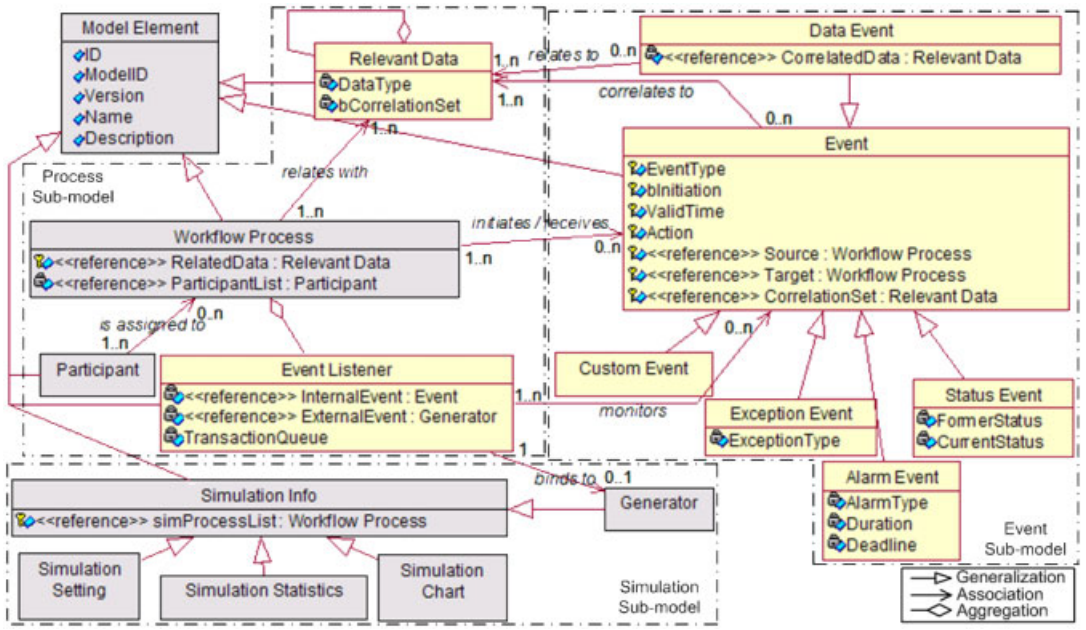


Figure 2. Static structure of the service-oriented workflow meta-model.

3.1. Sub-model definitions

The workflow meta-model is basically composed of four sub-models: the process sub-model, the event sub-model, the participant sub-model and the simulation sub-model, with three different types of relationship between individual elements—*Generalization*, *Association* and *Aggregation*. As the participant sub-model is irrelevant to our topic, we will not discuss it for simplicity. Briefly, we introduce the other sub-models as follows:

The *process sub-model* is composed of the classes *Process*, *Workflow Process*, *Activity*, *Subflow*, *Atomic Activity*, *Transition*, *Condition*, *Relevant Data*, *and Event Listener*. As the semantics of the classes except *Event Listener* complies with XPDL, we will not give detailed explanation here and readers can refer to the corresponding specification [15]. Specially, as an important extension to the XPDL model, we create the class *Event Listener* as an aggregated class to *Workflow Process*, *Atomic Activity* and *Subflow*, in order to monitor both external and internal run-time events in the lifecycle of workflow simulation, hence supporting the particular message communication patterns in a service-oriented environment. Besides, the attribute 'bCorrelationSet' in the class *Relevant Data* also plays a significant role in achieving data correlation in a loosely coupled environment. Details regarding these points will be discussed more thoroughly in the next section.

The *event sub-model* comprises the class *Event* and its five inherited classes: *Data Event*, *Status Event*, *Alarm Event*, *Exception Event* and *Custom Event*, serving as a fundamental part in the meta-model to achieve the specific interaction pattern in a service-oriented environment. An event is the encapsulation of any message which is transferred from one process/activity instance (or service instance in the service-oriented context) to another in the conversation between these instances.

DOI: 10.1002/cpe

Considering the characteristics of an event and its possible influence on the instance that initiates or receives it, we classify events into five categories, each being represented by the above five inherited classes. *Data Event* is used to specify those events having tight correlation with specific workflow relevant data; *Status Event* is for those arising from status alteration of the instance; *Alarm Event* is for those indicating duration or deadline limits for the execution of the instance; *Exception Event* is for those stemming from faults during the execution of the instance; and *Custom Event* is employed to provide extensibility for user-defined events. The common attribute 'bInitiation' defined in the superclass *Event* specifies whether this event is used to initiate an instance for the service receiving it; the attribute 'ValidTime' prescribes the time limit for a service instance to perform instance matching; and the attribute 'Action' designates the specific action incurred by an event, which will be interpreted by the ECA rule parser during workflow simulation and enactment.

The *simulation sub-model* consists of six classes—*Simulation Info*, *Simulation Setting*, *Simulation Statistics*, *Simulation Chart and Generator*. *Simulation Info* is the abstract base class for the other five classes, maintaining correlation with the process sub-model and defining commonly owned attributes. *Simulation Setting* is used to set up simulation scenarios such as the simulation schedule; *Generator*, a common class for traditional workflow simulation, is used to generate transaction queues based on specific statistical distribution models; *Simulation Statistics* and *Simulation Chart* are for presenting simulation results in user-defined formats.

3.2. Association definitions

As indicated before, there are three different kinds of relationship between individual elements in the workflow meta-model: *Generalization*, *Association* and *Aggregation. Generalization* describes the inheritance relationship between a superclass and its sub-classes; *Aggregation* shows the inclusion relationship between two elements or between an element and itself; and *Association* establishes the reference relationship between elements. We have assigned a name for each *Association* so that the relationship can be understood more easily. For example, the *Association* 'relates with' from the class *Process* to *Relevant Data* specifies that certain workflow relevant data might be correlated with an instance of *Process*, while 'is assigned to' from *Participant* to *Atomic Activity* means that necessary resources or application systems must be allocated to activity instances, or service instances in the service-oriented context, to support its execution.

Note that the multiplicity of an *Association* is indicated by the numeric signs near both ends of an *Association*. Take the Association 'correlates to' for instance, it has the following multiplicity: $Data\ Event/Relevant\ Data = 0 \ldots n/1 \ldots n$, which designates that an instance of *Data Event* must be correlated to no less than one instance of *Relevant Data*, while an instance of *Relevant Data* might not be correlated to any instance of *Data Event*, or it might be correlated to more than one instance of *Data Event*. Such multiplicity in an *Association* defines the quantitative proportion between the two associated classes.

4. SIMULATION MECHANISMS BASED ON INTERACTIVE EVENTS

In traditional simulation, individual generators are assigned to each workflow to generate random transactions independently and essentially, such kind of simulation only deals with single process, i.e. no interaction is incurred between the internal units of different workflows (see

DOI: 10.1002/cpe

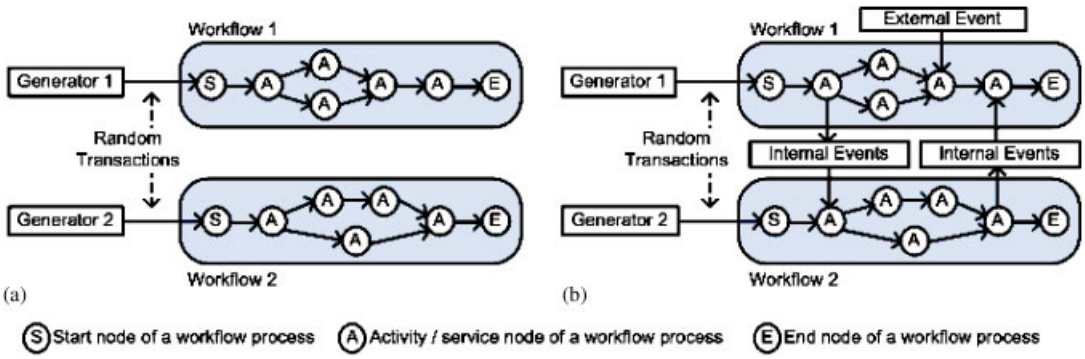


Figure 3. Comparison of simulation patterns: (a) traditional simulation and (b) simulation in service-oriented paradigm.

Figure 3(a)). However, workflow simulation in a service-oriented environment is intrinsically multi-process involved—some service in a workflow may need to interact with a service in another which is simulated simultaneously. In addition, there will also be external events acting on the internal units other than the start node and influencing the simulation process (see Figure 3(b)). Under such circumstances, the simulation engine should be modified to support specific mechanisms for external and internal event interactions, which will inevitably involve such issues as data correlation as well as asynchronous communication between different processes and the like. Based on the meta-model presented in the previous section, we will analyze the implementation of these mechanisms in the service computing environment from both the build-time and run-time perspectives in this section.

4.1. Build-time analysis

As introduced before, the class *Event Listener* acts as an event monitor in the model. Through the *Association* 'monitors', it contains a referenced attribute 'InternalEvent:Event', which actually is a queue of received events (at this point, we have counted in the asynchronous property of message communication in a service-oriented environment). Through the *Aggregation*s to *Workflow Process*, *Atomic Activity* and *Subflow*, an instance of *Event Listener* is permanently bound to a certain process instance, or activity instance, or sub-flow instance (in the service-oriented context, these are all defined as service instances), so that the events received by this *Event Listener* can influence the simulation of the corresponding process instance. Now look at the source of the events. Through the *Association* 'initiates/receives', the class *Event* contains two referenced attribute from the class *Process* ('Source' and 'Target'), indicating where the events come from and where they would go. Combining the above elements, these classes together with the associations between them provide a foundation for the implementation of internal message conversation during multi-process simulation.

Data correlation is a critical issue inherent in internal message communication. In traditional single-process simulation, data flow within a process and need not cross the boundary between different processes. Thus, the correlation of relevant data with process instances simply by instance IDs works reasonably well. However, the use of such IDs to correlate data would be rather difficult and even somewhat non-sensical when considering internal communication in multi-process

DOI: 10.1002/cpe

simulation. In order to solve this problem, we introduce the concept of Correlation Set from BPEL4WS [7] into our meta-model. On one hand, there is a Boolean attribute 'bCorrelation-Set' in the class *Relevant Data*, with which we could define whether a relevant data would be used as the correlation set. On the other hand, the class *Event* contains a referenced attribute 'CorrelationSet:Relevant Data' derived from the *Association* 'correlates to' with the class *Relevant Data*. Whenever necessary, this referenced attribute would serve as a combination of all the relevant data that is defined to be used as a correlation set, with the attribute 'bCorrelationSet' set as TRUE, so that we could easily correlate relevant data with the right process instance receiving the event.

On considering external events, we can simply regard them as random transactions. Therefore, we establish the *Association* 'binds to' between the classes *Event Listener* and *Generator*. The referenced attribute 'ExternalEvent:Generator' relates the instance of *Event Listener* to a specific instance of *Generator*, with the latter defining the statistical distribution model of the external events. In this way, we can take into account both internal message communication and external event handling in the design phase of our workflow models.

4.2. Run-time analysis

We will explore in this section the event communication behavior of part of the service nodes in the example presented in Section 2 during the running phase of simulation via the UML sequence diagram shown in Figure 4. For the sake of conciseness, we merely depict the situation of one customer in the diagram, and issues relevant to multiple concurrent customers will be rationally inferred later.

Generally, each composition has an independent generator to produce random transactions. Once the travel agency received requirement from a customer, i.e. a transaction arrived at the start node of composition 1, it created an instance of the globally defined *Relevant Data* CustomerOrder (specified as 'CustomerOrder[1]' in Figure 4) with 'CustomerID' as the *Correlation Set*. In executing 'Input customer Info', it transferred the data to subsequent nodes in composition 1 on one hand, and on the other initiated an instance of the *Data Event* CustomerInfo ('DataEvent[1]' in Figure 4), which contained part of the data in CustomerOrder and must include the *Correlation Set*. When the event listener aggregated in 'Login to online ticket system' received 'DataEvent[1]', it first performed instance matching with existing pending transactions through the operation 'InstanceMatch ()', using *Correlation Set* 'CustomerID' in the data event. Simulation involving other events is similar to this first situation.

What if the matching of instance returned a false value regarding every instance in execution? Our solution is that, the activity remains waiting for another instance to come for a predefined period. If the waiting time exceeds the deadline, a timeout exception would be initiated and the simulation engine would refer to the corresponding exception processing module. For this purpose, we include in the class *Event* an attribute named 'ValidTime' (see Figure 2).

In most cases during simulation, there may be multiple transactions and events pending at a particular service node at a certain point of time. Under such circumstances, the event listener aggregated in each node would create separate queues for each kind, by the attributes 'InternalEvent:Event' for internal events, 'ExternalEvent:Generator' for external events and 'TransactionQueue' for random transactions, respectively. As mentioned before, when the simulation engine encounters an event,

DOI: 10.1002/cpe

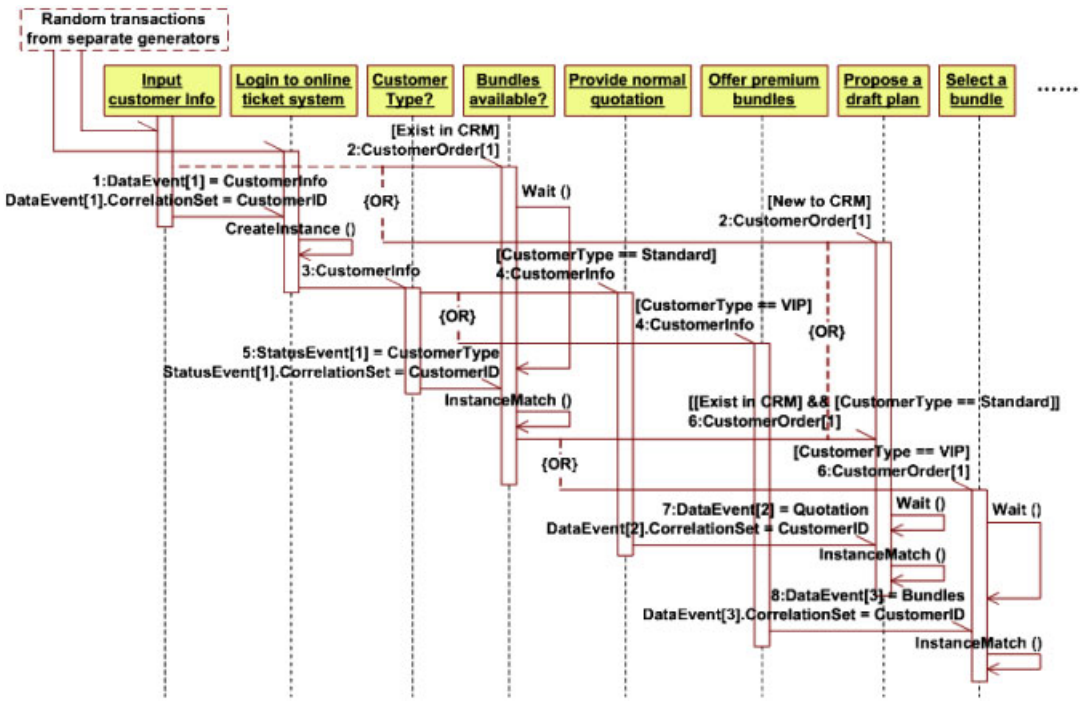


Figure 4. Simulation sequence diagram of the motivating example.

it would perform instance matching via the correlation set with each transaction in the queue until it finds the right one, or wait for new transactions until a specific deadline if no matching can be achieved with the existing instances. Simulation results regarding performance indicators of these queues should be quite useful to the evaluation of QoS.

Moreover, the potential leverage of multi-process simulation in performing model verification for service compositions, apart from the general function of simulation in performance analysis, is shown above. In most situations, two interactive compositions are independent from each other, thus possible error existed in the interaction logic between them can be revealed by examining the simulation trajectory. Such analysis is of great value for designers to compose services more effectively and correctly. Obviously, traditional simulation can never reveal such problems since both compositions are executed independently.

5. SYSTEM ARCHITECTURE

As shown in Figure 5, the workflow modeling and simulation system based on our meta-model consist of three layers: the user interface layer, the operation logic layer and the persistent storage layer, with the workflow meta-model as the supporting framework. We will elaborate on each of these layers hereinafter.

DOI: 10.1002/cpe

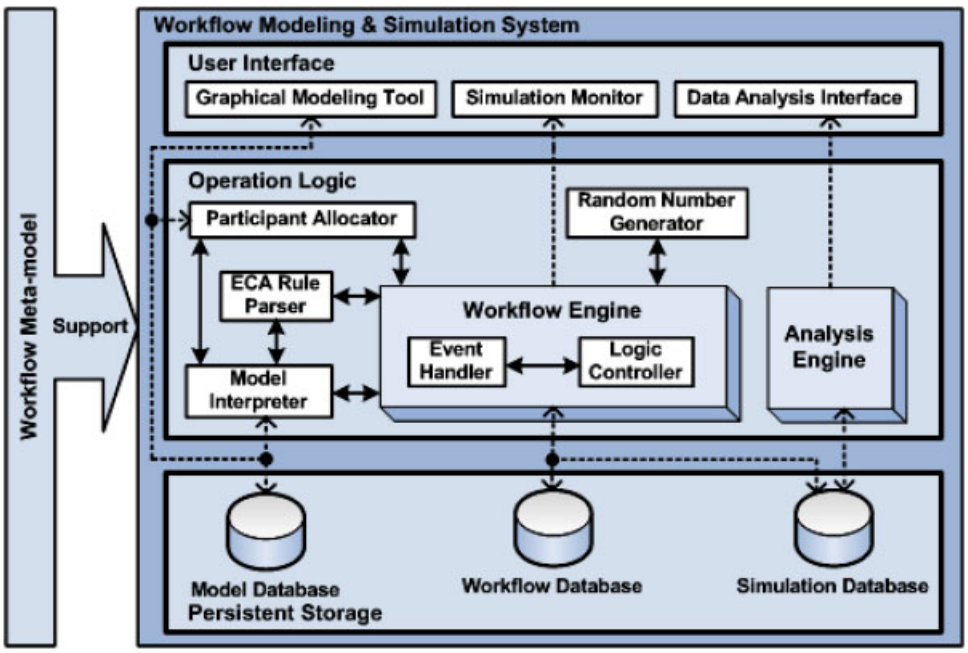


Figure 5. Architecture of the workflow modeling and simulation system.

5.1. User interface layer

The user interface layer, which is mainly composed of the graphical modeling tool, the simulation monitor and the data analysis interface, helps establish effective communication between users and the system. The modeling tool offers an easy-to-use visual interface for the modelers, who in turn can construct a multi-view enterprise model regarding the function, information, process, organization and resource elements within the company, as well as their inherent relationship. With an access to the model database, modelers can conveniently modify legacy models and create new models as well. The simulation monitor performs real-time interaction with the workflow engine during the process of simulation so as to provide real-time supervision upon some specific objects which the analyzers might be interested in. For instance, we can observe the dynamic utilization of some resource in the whole process of execution, which should be a useful basis for doing further detailed capacity analysis. The data analysis interface equips the analyzers with the ability to customize statistical reports according to multiple objectives, and its connection with the simulation database enables it to retrieve necessary statistics any time.

5.2. Operation logic layer

The operation logic layer functions as the core of the whole system, achieving control over the entire data and logic flows, as well as carrying out statistical calculation.

DOI: 10.1002/cpe

Extended WAPI Data Types:

```
typedef struct
{
        WMTAttributeList correlation_set;
} WMTCorrelationSet
typedef WMTCorrelationSet * WMTPCorrelationSet;

typedef struct
{
        WMTText event_id[UNIQUE_ID_SIZE];
        WMTText event_name[NAME_STRING_SIZE];
        WMTText event_type;
        WMTInt32 priority;
        WMTUInt32 valid_time;
        WMTUInt32 action;
        WMTPProcInst psource_proc_inst;
        WMTPProcInst ptarget_proc_inst;
        WMTPActivityInst psource_activity_inst;
        WMTPActivityInst ptarget_activity_inst;
        WMTPCorrelationSet pcorrelation_set;
} WMTEvent;
typedef WMTEvent *WMTPEvent;

typedef struct
{
        WMTPEvent p_event;
        WMTPAttributeList pcorrelated_data;
} WMTDataEvent;
typedef WMTDataEvent *WMTPDataEvent;

typedef struct
{
        WMTPEvent p_event;
        WMTPProcInstState pprev_proc_inst_state;
        WMTPProcInstState pnew_proc_inst_state;
        WMTPActivityInstState pprev_act_inst_state;
        WMTPActivityInstState pnew_act_inst_state;
} WMTStatusEvent;
typedef WMTStatusEvent *WMTPStatusEvent;

typedef struct
{
        WMTPEvent p_event;
        WMTText alarm_type;
        WMTUInt32 alarm_duration;
        WMTUInt32 alarm_deadline;
} WMTAlarmEvent;
typedef WMTAlarmEvent *WMTPAlarmEvent;

typedef struct
{
        WMTPEvent p_event;
        WMTErrRetType exception_type;
} WMTExceptionEvent;
typedef WMTExceptionEvent *WMTPExceptionEvent;
```

Extended WAPI Descriptions:

```
//Interface that creates a status event from a process instance
WMTErrRetType WMCreateStatusEventFromProcInst (
                in WMTPSessionHandle psession_handle,
                in WMTPProcInstState pprev_proc_inst_state,
                in WMTPProcInstState pnew_proc_inst_state,
                out WMTPStatusEvent pstatus_event)

//Interface that creates a status event from an activity instance
WMTErrRetType WMCreateStatusEventFromActInst (
                in WMTPSessionHandle psession_handle,
                in WMTPActivityInstState pprev_act_inst_state,
                in WMTPActivityInstState pnew_act_inst_state,
                out WMTPStatusEvent pstatus_event)

//Interface that creates a data event
WMTErrRetType WMCreateDataEvent (
                in WMTPSessionHandle psession_handle,
                in WMTPFilter pcorrelated_data_filter,
                out WMTPDataEvent pdata_event)

//Interface that creates an alarm event with a duration limit
WMTErrRetType WMCreateDurationAlarmEvent (
                in WMTPSessionHandle psession_handle,
                in WMTUInt32 duration,
                out WMTPAlarmEvent palarm_event)

//Interface that creates an alarm event with deadline
WMTErrRetType WMCreateDeadlineAlarmEvent  (
                in WMTPSessionHandle psession_handle,
                in WMTUInt32 deadline,
                out WMTPAlarmEvent palarm_event)

//Interface that creates an exception event
WMTErrRetType WMCreateExceptionEvent (
                in WMTPSessionHandle psession_handle,
                in WMTErrRetType exception_type,
                out WMTPAlarmEvent pexception_event)

//Interface that receives an event
WMTErrRetType WMReceiveEvent (
                in WMTPSessionHandle psession_handle,
                in WMTPEvent p_event)

/*Interface that performs instance matching when receiving
an event*/
WMTErrRetType WMInstanceMatching (
                in WMTPSessionHandle psession_handle,
                in WMTPFilter pcorrelation_set_filter,
                in WMTPEvent p_event)
```

Figure 6. Data types and WAPI definition.

DOI: 10.1002/cpe

The workflow engine is the kernel component of the operation logic layer, with the participant allocator, the model interpreter, the ECA rule parser and the random number generator as supportive components, and the analysis engine is for building up statistical analysis with respect to the requirements received from the data analysis interface. The workflow engine is basically composed of a logic controller and an event handler, with the former dealing with the navigation of logic flows as well as data flows in a workflow model, and the latter handling particular events in a service-oriented environment. The message communication and data correlation mechanisms described in the previous sections are realized by the event handler, and pertinent functions are encapsulated in this module.

The Workflow Management Application Programming Interface (WAPI) Specification by WfMC [15] (Interface 2 & 3) has defined standard APIs which can be supported by workflow management products. These APIs, such as *WMCreateProcessInstance*, *WMAssignActivityInstanceAttribute*, *WMChangeActivityInstanceState*, *WMTAInvokeApplication*, etc. are encapsulated in the logic controller within the workflow engine. According to the conventions of the above specification, we define several APIs (Figure 6) in the event handler module to support event handling and data correlation.

The first table in Figure 6 defines some data types specific to the concepts of *Correlation Set* as well as *Event* proposed in this paper, and the second table defines particular WAPIs, which realize the corresponding message communication and data correlation mechanisms.

5.3. Persistent storage layer

The persistent storage layer consists of three databases: the model database, the workflow database and the simulation database. The model database is mainly for storing the enterprise models constructed, as well as the underlying constraints and scheduling rules. The interaction between the model database and the model interpreter in the operation logic layer prepares necessary model information for simulation on one hand, and the interaction with the graphical modeling tool in the user interface layer realizes the creation, modification and storage of the models, and in the meanwhile offers support in the version management as well as the knowledge management during the modeling process. The workflow database is for maintaining the relevant data and the instance data which would be referred to and operated on during the course of simulation. Its communication with the workflow engine ensures that the right data would be transferred at the right time to the right instances. The simulation database is the database for those statistics calculated in simulation, including dynamic real-time statistics as well as static results. This database interacts with both the workflow engine and the data analysis engine, not only giving support in obtaining and recording data in simulation, but also helping produce customized statistical analysis reports.

6. ANALYSIS OF THE MOTIVATING EXAMPLE IN THE PROTOTYPE SYSTEM

6.1. A snapshot of the system

Figure 7 shows the simulation snapshot of the motivating example in our prototype system. Apart from the ordinary control flows between different elements within each workflow, there are also

DOI: 10.1002/cpe

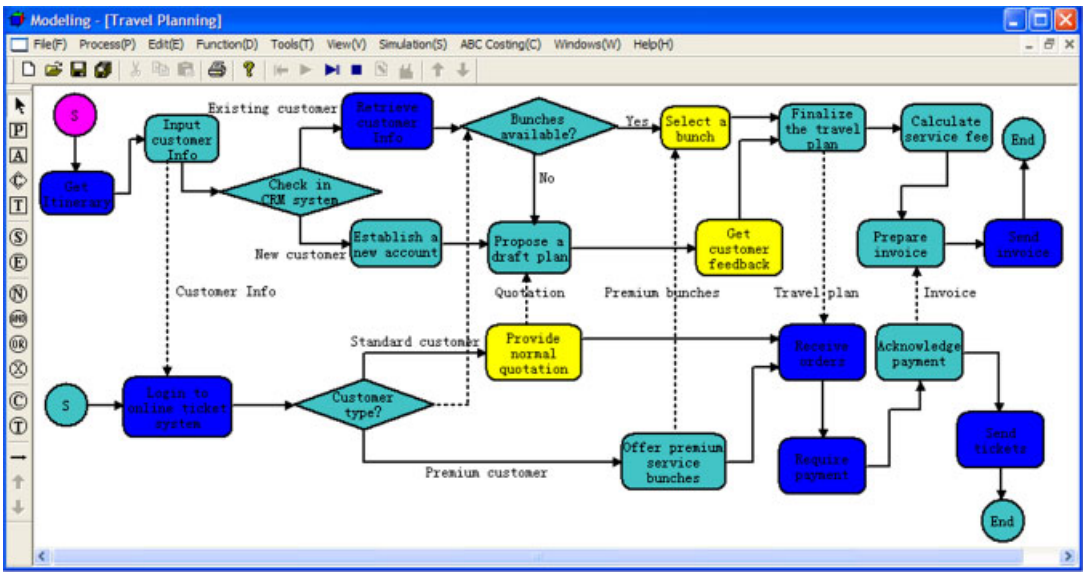


Figure 7. A snapshot of the prototype system.

explicit event flows between the two workflows, represented by dotted lines, such as the event flow carrying 'Quotation' information from 'Provide normal quotation' to 'Propose a draft plan', and the like. Event flows which have a data annotation nearby transfer data events, while those without notation represent status event flows. (For alarm events and exception events not included in our example, we assign a clock and an error symbol, respectively, to each type for identification.) Correlation sets formed by global data are passed along the event flows in order for instance correlation. Different shades of the figures indicate different states of the elements during simulation, e.g. dark-shaded rectangle means the activity is under execution, while light-shaded rectangle represents that the activity is waiting for necessary resources to perform its tasks. The system is developed on the Microsoft Visual C# platform, with SQL 2000 as the database.

6.2. Structure analysis of service compositions

The simulation trace for two certain customers in our example is given in Table I. Note that a dead lock occurred in simulating travel planning for C2, when the agency workflow waited at the node 'Propose a draft plan' for response from the ticket handling workflow, while the latter sent back bundles information to the node 'Select a bunch'. Carefully comparing the simulation tracks of both customers and examining the structure of both compositions, it is not difficult to find out the radical cause for the dead lock—that is because in our example, the two compositions have distinct criteria for classifying their customers: the agency classifies customers into new or existing ones, while the ticket system categorizes customers into standard ones or VIPs. In the agency workflow, if a customer is new to it, it will never examine whether a bundle is applicable but directly wait for normal quotation from the ticket system so as to propose a draft plan. This logic works well

DOI: 10.1002/cpe

Table I. Simulation trace of some transactions.

Trace in itinerary planning	
C1	Get itinerary → Input customer Info → Check in CRM system → Establish a new account→ Propose a draft plan → Get customer feedback → Finalize the travel plan → Calculate service fee → Prepare invoice → Send invoice
C2	Get itinerary → Input customer Info → Check in CRM system → Establish a new account
Trace in ticket handling	
C1	Login to online ticket system → Customer type? → Provide normal quotation → Receive orders → Require payment → Send tickets
C2	Login to online ticket system →Customer type? → Offer premium service bundles

if the assumption that a customer new to the agency is always a standard customer in the ticket system is satisfied. However, this is not always true considering that the two compositions are independent to each other. When a new customer of the agency is recognized as a VIP in the ticket system, simulation will run into a dead lock, which is the case for the second customer in our simulation. Compared to complex formal methods for model verification like Petri-net, revealing such underlying error in the interaction logic between distinct compositions in an apparent way can surely provide designers with helpful guidance to make necessary modification in either the structure of individual compositions or the interaction pattern between them, hence making compositions cooperate with each other more efficiently.

6.3. Workload analysis of service nodes

Aside from its benefit in model verification for service compositions, statistical results relevant to events that are calculated during simulation can be used to evaluate the performance, in particular to the work in this paper, the workload of corresponding service nodes. Such analysis can not only point out the bottleneck of the system, but help give directions to achieving load equilibrium among various services in a composition as well.

As an instance, we examined the workload of two distinct nodes, say, 'Propose a draft plan' and 'Select a bunch', in the motivating case. The performance indicator that we picked as a reflection of the workload is the maximal queue length in either node at each simulation. We changed the total transaction numbers generated in successive simulations from 10 to 50, and observed the maximal length of the event queue corresponding to respective nodes. A comparison on the resulting statistics is presented in the left part of Figure 8, from which we can easily figure out that the workload of 'Propose a draft plan' (the dark line) increased dramatically as the transaction size increased, while that of 'Select a bunch' (the gray line) stayed low. This result has given us a hint that the service node of 'Propose a draft plan' might be the bottleneck of the system.

Given the above outcome of preliminary analysis, we stepped a little further in a subsequent analysis. Considering that 'Propose a draft plan' might be a bottleneck in the system, we changed the resource allocation in the simulation settings and tried to find out how the workload of this service would alter due to increase in available capacity. Indeed, we compared the effects of two different kinds of capacity expansion (shown in the right part of Figure 8), one with increase in available network bandwidth (the dark line), and the other in labor (the gray line). The result was rather straightforward: the influence of expansion in network bandwidth on the performance of

DOI: 10.1002/cpe

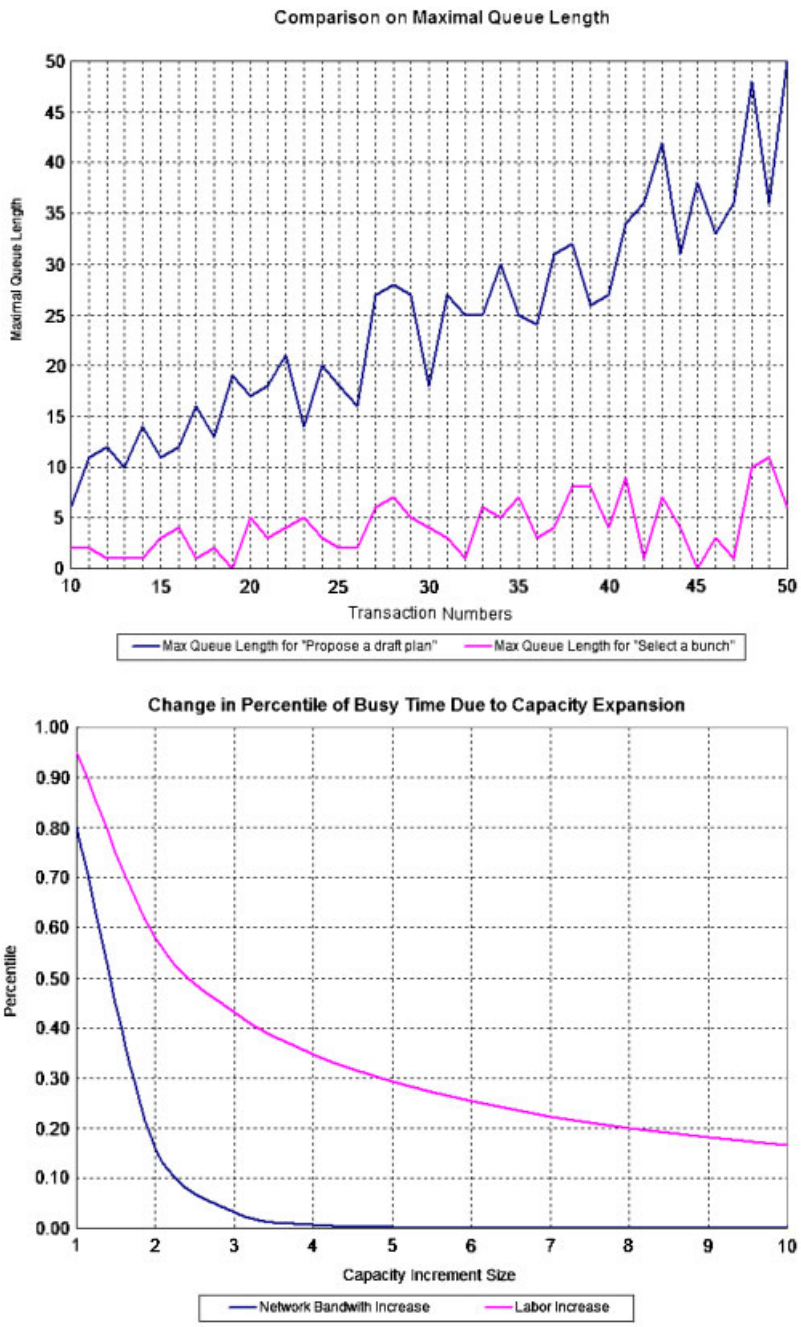


Figure 8. Workload analysis.

DOI: 10.1002/cpe

the service greatly outweighed that from labor increase. If costs are not so significant compared to lead time or cycle time in the considerations of the organization, one feasible way to improve performance should be to employ better networks.

7. CONCLUSIONS AND FUTURE WORK

Observing the lack of implementation mechanisms for simulating the actual behaviors of composite services, we propose an interactive-event-based workflow simulation method which is adaptive to the loosely coupled service computing environment. At build time, we extend the XPDL meta-model to incorporate event elements, so that interactive event flows between individual workflows can be explicitly modeled when services are designed. At run time, we developed a simulation engine to support the event-based interaction pattern as well as data correlation.

Furthermore, based on some simple yet non-trivial analysis of the motivating example, we have also revealed the advantage of our simulation method against traditional ones in performing model verification for interactive service compositions, and this should be a promising issue in the research of workflow simulation technique in service-oriented computing. On the other hand, although we have shed some light on the strength of event-based simulation in evaluating service performance, those tests were just far too simple in comparison to realistic problems. Not only do we need to enrich our analyzing templates, but we should also take into account how to realize overall performance measurement, in which case multiple aspects of performance such as costs, lead time, resource utilization, etc. would be incurred. Also, research on involving qualitative elements like customer satisfaction into the model can be another meaningful topic.

REFERENCES

1. Piccinelli G, Williams SL. Workflow: A language for composing web services. *Proceedings of the International Conference on Business Process Management 2003* (*Lecture Notes in Computer Science*, vol. 2678). Springer: Berlin, 2003; 13–24.
2. Jablonski S. Processes, workflows, web service flows: A reconstruction. *Data Management in a Connected World* (*Lecture Notes in Computer Science*, vol. 3551). Springer: Berlin, 2005; 201–213. DOI: 10.1007/11499923_11.
3. Moscato F, Mazzocca N, Vittorini V *et al*. Workflow pattern analysis in web services orchestration: The BPEL4WS example. *Proceedings of the 1st International Conference on High Performance Computing and Communications* (*Lecture Notes in Computer Science*, vol. 3726). Springer: Berlin, 2005; 395–400. DOI: 10.1007/11557654_48.
4. Xiao Y, Chen D, Chen M. Research of web services workflow and its key technology based on XPDL. *Proceedings of the 2004 IEEE International Conference on Systems*, *Man and Cybernetics*, vol. 3. IEEE Computer Society Press: Los Alamitos, CA, 2004; 2137–2142. DOI: 10.1109/ICSMC.2004.1400643.
5. Li H, Lu Z. Decentralized workflow modeling and execution in service-oriented computing environment. *Proceedings of the 2005 IEEE International Workshop on Service-Oriented System Engineering*. IEEE Computer Society Press: Los Alamitos, CA, 2005; 29–34. DOI: 10.1109/SOSE.2005.9.
6. Andrews T, Curbera F, Dholakia H *et al*. Business Process Execution Language for Web Services, Version 1.1. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/ [May 2003].
7. Zhao X, Liu C. Supporting relative workflows with web services. *Proceedings of the 8th Asia-Pacific Web Conference 2006* (*Lecture Notes in Computer Science*, vol. 3841). Springer: Berlin 2006; 680–691. DOI: 10.1007/11610133_59.
8. Casati F, Shan MC. Event-based interaction management for composite E-services in eFlow. *Information Systems Frontiers* 2002; **4**(1):19–31. DOI: 10.1023/A:1015374204227.
9. Wang Y, Li M, Cao J *et al*. An ECA-rule-based workflow management approach for web services composition. *Proceedings of the 4th International Conference on Grid and Cooperative Computing* (*Lecture Notes in Computer Science*, vol. 3795). Springer: Berlin, 2005; 143–148. DOI: 10.1007/11590354_19.

10. Guo W, Yang Y, Zhai Z. Grid services adaptation in a grid workflow. *Proceedings of the 4th International Conference on Grid and Cooperative Computing* (*Lecture Notes in Computer Science*, vol. 3795). Springer: Berlin, 2005; 172–177. DOI: 10.1007/11590354_24.
11. Chandrasekaran S, Silver G, Miller JA *et al*. Web service technologies and their synergy with simulation. *Proceedings of the 2002 Winter Simulation Conference*, vol. 1. IEEE Computer Society Press: Los Alamitos, CA, 2002; 606–615. DOI: 10.1109/WSC.2002.1172937.
12. Chang H, Song H, Kim W *et al*. Simulation-based web service composition: Framework and performance analysis. *Proceedings of the 3rd Asian Simulation Conference* (*Lecture Notes in Computer Science*, vol. 3398). Springer: Berlin, 2004; 352–360. DOI: 10.1007/b105611.
13. Song HG, Lee K. sPAC (Web Services Performance Analysis Center): Performance analysis and estimation tool of web services. *Proceedings of the 2005 International Conference on Business Process Management* (*Lecture Notes in Computer Science*, vol. 3649). Springer: Berlin, 2005; 109–119. DOI: 10.1007/11538394_8.
14. Workflow Management Coalition. *Workflow Process Definition Interface—XML Process Definition Language*, Version 1.0, October 2002.
15. Workflow Management Coalition. *Workflow Management Application Programming Interface* (*Interface 2 & 3*), Version 2.0, July 1998.