Interactive-Event-Based Workflow Simulation in Service Oriented Computing

Yanchong Zheng, Yushun Fan, Wei Tan
Department of Automation, Tsinghua University, Beijing 100084, China
mccarrot99@mails.tsinghua.edu.cn, fanyus@tsinghua.edu.cn, tanwei@mails.tsinghua.edu.cn

Abstract

The evolution of Grid technologies has brought to the world a loosely coupled environment that enables flexible integration among heterogeneous systems. On building a service-oriented architecture for Grid systems, more and more research is focused on service composition, in which workflow and simulation techniques have shown great potential. Simulation of services’ interaction is important since the services eco-system is unstable and in continuous evolution. However, there is a lack in the research of services’ simulation, especially models and methods for simulating interactions between composite services. In this paper a workflow simulation method based on interactive events is proposed to fulfill this requirement. By introducing an event submodel in the workflow meta-model, the event interaction pattern is achieved from both the build-time and run-time perspectives. 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 composition are also highlighted.

1. Introduction

Grid technologies have developed quickly to a degree of maturity in recent years. By embracing Web services on Grid [1], there are emerging needs for service composition. In the meanwhile, workflow technology has been applied to automated service composition in the distributed computing environments of Grid services [2-4].

Many studies have demonstrated the strength of workflow and simulation techniques in the design and performance analysis of service composition. [5] proposed a conceptual model of Web services workflow, and [6] proposed a framework for modeling and reusing workflows as sub-workflows in service composition. By mapping to the Business Process Execution Language for Web Services (BPEL4WS) [7], [8] studied the modeling and implementation of organization centered workflows in the Web service environment. Particularly, [9] developed an architecture that employed event elements to achieve dynamic interaction between composite services; [10] proposed an ECA-rule-based method for web service composition; [11] introduced Pi-Calculus to address the protocol level deadlock in grid workflows; [12] explicated the power of simulation as a part of Web service composition and process design; and [13, 14] both utilized simulation techniques to evaluate service composition based on their QoS properties.

However, to the best of our knowledge, few of the above research has made substantial efforts in studying the specific 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. With a motivating scenario in section 2, we expatiate on the core mechanisms from the build-time and run-time perspectives in section 3 and 4 respectively, based on the workflow meta-model presented. The system architecture is presented in section 5, and section 6 gives important analysis results based on the information we gathered through simulation. Finally, conclusions and promising future work are given in section 7.
2. Motivating scenario

In traditional simulation, individual generators are assigned to each workflow to generate random transactions independently and no interaction is incurred between the internal units of different workflows (see Fig 1.1). However, simulation of interactive service compositions is intrinsically multi-process involved – some service in a composition may need to interact with a service in another which is simulated simultaneously. In addition, there may also be external events acting on the internal units other than the start node and influencing the simulation process (see Fig 1.2). 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 workflows and the like.

Now consider two service compositions that have internal interaction as shown in Fig. 2. 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 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 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 2 determines which route to go after the decision node ‘Bundles available?’ in composition 1, 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. Build-time analysis of the interactive-event-based simulation mechanism

As shown in Fig. 3, our workflow meta-model is 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) [15], enabling specific mechanisms for event interaction by incorporating an event sub-model and establishing its relations with the other two sub-models: the process sub-model and the simulation sub-model. For simplicity, the class Workflow Process is designated here as the miniature of process elements such as subflow, atomic activity, etc. defined in XPDL [15], and relations connected to it are considered relations with these elements.

An event is the encapsulation of any message which is transferred from one process/activity instance (in the service computing context, we regard it as a service instance in the rest of our paper) to another in the conversation between these instances. The common attribute ‘bInitiation’ defined in the super class 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 class Event Listener is created for monitoring both external and internal run-time events in the lifecycle of workflow simulation. 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 service computing). Through the Aggregation to Workflow Process, an instance of Event Listener is permanently bound to a certain service instance, so that the events received by this Event Listener can influence the simulation of the corresponding service instance. On the other hand, through the Association ‘initiates/receives’, the class Event contains two referenced attribute from the class Workflow 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.

![Figure 3. Static structure of the workflow meta-model](image-url)
Data correlation is a critical issue inherent in internal message communication. In traditional single-process simulation, data flows 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 nonsensical when considering internal communication in multi-process simulation. In order to solve this problem, we introduce the concept of Correlation Set from BPEL4WS [7] into our metamodel. On one hand, there is a Boolean attribute `bCorrelationSet` 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.

For external events, we 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 event communication and external event handling in the design phase of our workflow models.

Take for instance, the ‘Customer Information’ and the ‘Customer type status’ in our motivating example can be defined as a data event and a status event respectively as in Fig. 4 (these event definitions will be referred to in the run-time analysis of the simulation mechanisms in the next section).

![Figure 4. Event definition examples](image)

### 4. Run-time analysis of the interactive-event-based simulation mechanism

We will explore in this section the event communication behavior of part of the service nodes in the motivating example during the running phase of simulation via the UML sequence diagram shown in Fig. 5. For the sake of conciseness, we merely depict the situation of one customer in the diagram, and issues relevant to multiple customers will be rationally inferred later.

Generally, each composition has an independent generator to produce random transactions, emulating the arrival of customer requirements. 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 Fig. 5) 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 Fig. 5), which contained part of the data in CustomerOrder and must include ‘CustomerID’, the Correlation Set. When the event listener aggregated in ‘Login to online ticket system’ received ‘DataEvent[1]’, it initiated an instance bound to the current customer through the operation ‘CreateInstance()’, as the attribute ‘bInitiation’ was set as TRUE in the definition of this data event (please refer to the former section).

Suppose that the workflow instance arrived at ‘Bundles available?’ earlier than response from ‘Customer type?’, its operation would be suspended. Once the status event CustomerType (‘StatusEvent[1]’ in Fig. 5) arrived, the simulation engine would first perform instance correlation with existing pending transactions through the operation ‘InstanceMatch()’, using the Correlation Set ‘CustomerID’ in the event. If matching succeeds, simulation will continue on the subsequent nodes.

What if the matching of instance failed regarding every instance in execution? Our solution is that, the activity remains waiting for another instance to come for a predefined period specified by the attribute ‘ValidTime’. 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.

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, 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 are quite useful to the evaluation of QoS.

Figure 5. Simulation sequence diagram of the motivating example
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 Fig. 6, the workflow modeling and simulation system based on our method consists of three layers: the user interface layer, the operation logic layer, and the persistent storage layer. The user interface layer includes interfaces for workflow model presentation and user interaction, while the persistent storage layer comprises databases for storing the workflow models and relevant data.

The core of the operation logic layer is the workflow engine, with the participant allocator, the model interpreter and the ECA rule parser as supportive components. The engine is basically composed of a logic controller and an event handler, with the former dealing with the navigation of control flows as well as data flows in a workflow model, and the latter handling particular events in service computing. The event communication and data correlation mechanisms described in the previous chapters are realized by the event handler, and pertinent functions are encapsulated in this module.

6. Analysis of the motivating example in the prototype system

The simulation snapshot of the motivating example in our prototype system is shown in Fig. 7. Apart from the ordinary control flows between different elements within each workflow, there are also 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 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 colors of the figures indicate different states of the elements during simulation, e.g., dark blue means the service is under execution, while yellow represents that the service is waiting for necessary resources to perform its tasks.

The simulation trace for two certain customers in our example is given in table 1. 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’s not difficult to find...
out the radical cause for the dead lock – that’s 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 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.

Figure 7. A snapshot of the simulation example in the prototype system

Table 1. Simulation trace of some transactions

<table>
<thead>
<tr>
<th>Trace in Itinerary Planning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>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</td>
<td></td>
</tr>
<tr>
<td>C2 Get Itinerary → Input customer Info → Check in CRM system → Establish a new account</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trace in Ticket Handling</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C1 Login to online ticket system → Customer type? → Provide normal quotation → Receive orders → Require payment → Send tickets</td>
<td></td>
</tr>
<tr>
<td>C2 Login to online ticket system → Customer type? → Offer premium service bundles</td>
<td></td>
</tr>
</tbody>
</table>

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 of corresponding service nodes. For example, we monitored the length of each event queue in our simulation, and the result turned out that the node ‘Prepare invoice’ often had the longest queue, which meant that this node might be a
bottleneck for reducing the cycle time of the agency workflow. In another case, we compared the number of events processed by ‘Propose a draft plan’ and ‘Select a bunch’, and found out that the former handled a lot more than the latter. Similar analysis can help evaluate the workload of different services, therefore giving directions to achieve load equilibrium among various nodes in a composition.

7. Conclusions and future work

In this paper, we propose an interactive-event-based workflow simulation method which is adaptive to the loosely coupled service computing environment. Based on some extension of the XPDL meta-model to incorporate event elements, interactive event flows between individual workflows are explicitly modeled at design time, and event interactions with data correlation are implemented at run time.

Furthermore, based on some simple yet non-trivial analysis of the motivating example, we have 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, future efforts can also be dedicated to the performance evaluation of service compositions based on simulation.

Acknowledgements

This work was granted by National High-Tech R&D (863) Plan of China (2005AA411912), and IBM-Tsinghua Joint project ‘Service-Oriented Business Process Modeling and Optimization’ co-sponsored by IBM China Research Lab and the National CIMS Engineering Research Center, Tsinghua University, China.

References


