基于注解的服务编排

王　斌　黄鹤远　徐景民　朱　俊　(IBM 中国研究院 北京 100193)

Annotation-Based Service Orchestration

Bin Wang,　Heyuan Huang,　Jingmin Xu,　Jun Zhu (IBM China Research Lab, Beijing 100193)

Abstract: Service orchestration plays a vital role in assembling services into business processes in a Service Oriented Architecture. In current practices, the orchestrating logic is usually described by a process language, which is therefore separated from the services in the system implemented by certain programming language. It introduces two issues: 1) It cost additional efforts for developers to be proficient with a new process language/script, and its running environment. 2) It causes development performance degradation due to transformation efforts for process language and programming language, such as transforms Java services into web services. To overcome these issues, this paper proposes a novel alternative system which takes advantage of the annotation construct of Java programming language to represent business processes. Through the experiments, we found that developers can efficiently develop business processes based on their current proficient programming language skill using the proposed system to achieve the service orchestration.

Key words: service orchestration; annotation service; oriented architecture

1　Introduction

Web services orchestration refers to an executable business process that can interact with both internal and external Web services. The interactions occur at the message level. They include business logic and task execution order, and they can span applications and organizations to define a long-lived, transactional, multis-tep process model[1].

Service orchestration plays a vital role in assembling services into business processes in a Service Oriented Architecture(SOA) In current practices, the orchestration logic is usually described by programming language neutral scripts(e.g. BPEL[2], Business Process Execution Lan- guage, WSCI[3], Web Service Choreography Interface, and etc), and can be executed by their supporting orchestration engines. The process logic is separated from the services in the system implemented by certain programming language. In spite of the greater flexibility this way introduces, it brings a couple of issues at the meantime:

(1) The orchestration script is usually a different language from the one developers use to develop the services in the system. This requires the developers be proficient on both languages and able to deal with the introduced mediation issues such as variable mapping. For example, BPELJ[4] is introduced to enable the cooperation between Java and BPEL by enclosing sections of Java code in XML. Obviously, a BPELJ developer must spend efforts to learn BPEL and the usage of Java within BPEL.

(2) The orchestration script can not be seamless integrated into existing services environment naturally. For example, BPEL can orchestrate web services only, if one system contains many existing services with different type, the BPEL process developer must transform these non-web services into web services, this may degrade the development performance.

(3) The orchestration script such as BPEL is usually running in a different context from the original system,

which is shown in Fig.1.

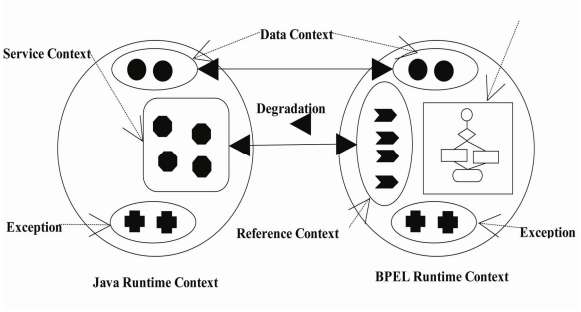

Fig.1 Elements and degradation analysis in an application system with Java and BPEL

The resulted runtime performance degradation is caused by 2 points: (1) the newly introduced runtime context(BPEL Runtime Context in Fig.1) needs to re-define concepts like data types, service references, and exception handling artifices that are duplicated from the original system. This consumes additional system computing resources and time. (2) Inter-process communication between two systems is also a big cost not acceptable for some real-time process systems.

In order to cover these issues to some extent, this paper proposes a novel alternative system which employs annotations in Java programming language to represent business processes. Existing Java language constructs (such as method, field, and etc.) are annotated to specify business activities, variables, service references, and etc and therefore as the basis to provide the process orchestration logic like sequencing, joining, splitting, and etc. In contrast to introduce a new and complex script language, this approach uses the programming language plus simple annotations for representing the orchestration logic. It not only relieves the developers from learning a new language, but also boosts the build-time and runtime performance of the orchestration engine.

In the following part, some related works in service orchestration area are introduced with identified limitations. Then, the syntax of the proposed annotation-based approach is given with samples to model typical process patterns. After that, the architecture of the system to enable the approach is illustrated. Some experiments were conducted to assess the proposed approach versus existing approaches. The last section concludes this paper and points out some future research directions.

2 Related Works

Currently, many WS-BPEL vendors have provided web services-based orchestration system products, such as the IBM Websphere Process Server, the Oracle BPEL Process Manager, and the ActiveBPEL Engine. Besides web service orchestration, JOpera and ESB (Enterprise Service Bus) is proposed for legacy service composition. They can integrate existing enterprise applications such as RMI, EJB and etc.

The business process execution language (BPEL) is the leading standard language for orchestration of Web services[5], which defines a model and a XML based grammar for describing the behavior of a business process based on the interactions between the process and its partners.

WS-BPEL utilizes several XML specifications: WSDL 1.1, XML Schema 1.0, XPath 1.0 and XSLT 1.0. Obviously, XML language is different from the programming language, which could be Java, C# and so on. Considering the communication gap between these different languages, performance degradation might appear. Besides, WS-BPEL also has several other limitations. First, based on the XML foundation, it is not easy for a freshman to start from scratch. Secondly, manipulating XML-based data is difficult and inefficient.

As the combination of BPEL and Java programming language, BPELJ allows the two languages to be used together to build business process applications. BPELJ enables Java and BPEL to cooperate by enclosing sections of Java code, called Java snippets. In order to allow processes to use Java resources rather than Web services, BPELJ makes it possible to create partner link types whose interfaces are defined using Java interfaces rather than WSDL port types.

Although BPELJ is allowed to communicate with Java directly, it is still a kind of mixed language and the gap between XML and Java may be a potential trouble. The variables defined in XML Schema or WSDL message needs map to Java variables and vice versa. Furthermore, not every kind of Java type could be mapped to an XML format. In addition, considering that the embedded Java snippet is hard to reuse, it's not a good design to mix Java snippets into HTML for early JSP.

3 Annotation-Based Process Modeling

Based on our analysis of some programming languages, such as Java, we realized that these languages provide additional constructs, such as annotation, which could be leveraged to extend new syntax in a non-intrusive way. In the meanwhile, existing constructs of the language could be leveraged to deal with data manipulation, service invocation like logics required by service orchestration. Thus, we don't need to introduce a dedicated service orchestration language.

In this section, we first give the syntax of our annotation based approach. To prove the feasibility and applicability of our approach for service orchestration, we give some samples to model typical process patterns. Furthermore, we introduce human-task activity modeling.

3.1 Annotation definition

We employ the annotations in Java language which is published by Sun Microsystems Inc. in 2005.

There are three categories of annotations in our system (1) Variables in Process (2) Activities in Process (3) Service Binding.

(1) Variables in process, we need to indicate which one is used for a process. It therefore could be saved for a long-running process. Moreover, we need to set up the linkage between the outside message and the process instance for correlation. @Variable and @CorrelationSet are introduced accordingly.

(2) Activities in process, such as invoke, receive and so on, are specified by the second kind of annotation. For example, there are @InvokeActivity and @Receive Activity. Moreover, the control logics of a process (such as parallel, merge, and choice) are also involved in this category.

(3) The outside services referenced in the process also need to be indicated explicitly; otherwise the programmer needs to take care of them in case like making a web service call. @Reference is used in our system for this purpose.

Below are the detailed definitions and descriptions of the annotations used in the system:

[F] @Variable (name=String)

The @Variable annotation is applied on Java fields, indicating that the fields are used in the process. The "name" attribute is the same as the field's name by default.

[F] @CorrelationSet (name=String, fields=String[])

The @CorrelationSet is used to declare a linkage between a set of fields and a process instance.

[M] @Start (name=String, post=String)

The @Start annotation is used to indicate the first activity of a process.

[M] @End (name=String)

The @End annotation is used to indicate the end of a process. The process instance is ended when the annotated method is invoked.

[M] @InvokeActivity (name=String, pre=String[], post=String)

The @InvokeActivity annotation is used to indicate that the following method will call a referenced service or outside application.

[M] @ReceiveActivity (name=String, pre=String[], post=String)

The @ReceiveActivity annotation is used to indicate that the process instance waits for a message, and the message name is in its "pre" attribute.

[M] @AndSplit (name=String, post=String[])

The @AndSplit annotation is used to indicate that the process splits to multi-threads to execute the following activities in parallel after the annotated method. The post attribute is a String collection, contains all the following activities' names.

[M] @AndJoin (name=String, pre=String[], post=String)

The @AndJoin annotation is used to synchronize the previously split activities. The annotated method will be treated as a join activity to wait for the previously split activities.

[M] @OrSplit (name=String, post=String[])

The @OrSplit annotation is used to indicate that there is an exclusive choice; the run-time engine selects a branch to continue the process. All candidates are contained in the post attribute.

[M] @OrJoin (name=String, pre=String[], post=String)

The @OrJoin annotation is used to merge previous process branches. The "pre" attribute obviously

contains all previous activities' names.

[M] @Message (name=String)

The @Message annotation is used to designate a listener. When the annotated method is invoked from outside partner, the related receive activity (The name appears in ReceiveActvity's "pre" attribute) is executed automatically.

[F,M] @Reference (name=String, target=String)

The @Reference annotation can be employed both on a Java field and a Java method. The purpose is to ask the run-time engine to find and inject a service instance which implements the service interface.

3.2 Typical process patterns modeling

In Service Oriented Architectures (SOA) workflow modeling languages have found a good application to define an executable model of the flow of information between a set of services [6].

The control-flow perspective (or process) perspective describes activities and their execution ordering through different constructors, which permit flow of execution control, e.g., sequence, splits, parallelism and join synchronization[7]. The service orchestration logic describes the interchange ordering of the message, so we apply workflow patterns to model process.

Currently, most workflow languages support the basic constructs of sequence, iteration, splits (AND and OR) and joins (And and OR) [8]. In this section, we introduce the usages of above annotations for each pattern.

(1) Sequence

There are 4 activities in Fig.2. "selectProducts" follows "getProductList"; it is followed by "payMoney", and the last activity is "rsvProducts". The code snippet shows the usage of annotations without the details of each activity itself.

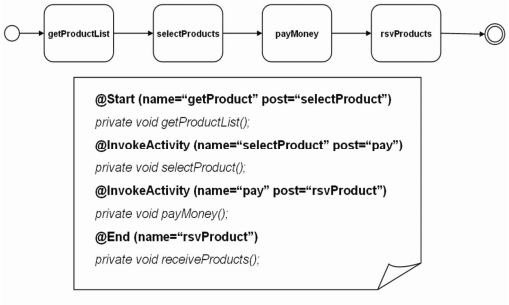

Fig.2 Sequence pattern

(2) Parallel & synchronization

In Fig.3, activities "RsvHotel", "RsvCar", and "RsvFlight" are executed simultaneously after "Reservation" activity. The "prepare" method is annotated with @AndSplit and with the attribute "post", which indicates that it is followed by three activities: "hotel", "car", and "flight".

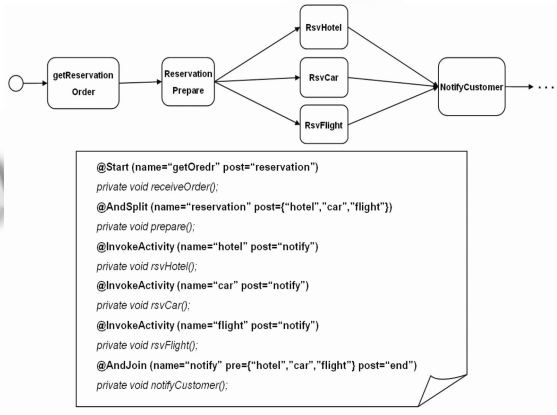

Fig.3 Parallel & synchronization patterns

(3) Exclusive choice & simple merge

Fig.4 shows how to use annotations to model selective branch. "Accept" and "Reject" is an opposite pair and one of them is selected at run-time. The condition is described in their previous activity "check". The "check" method returns a String value, which is the selected result. The returned String value should be either "accept" or "reject". The following snippet is a sample of this method.

(4) Arbitrary cycles

In Fig.5, if the guesser does not answer right, the flow would start another round, making up a loop to keep guessing until the answer is right. The "checkAnswer" is also an "OrSplit" activity.

3.3 Human-task activity modeling

In order to enable human-task in a service orchestration, we define three new annotations to model the human-task activity.

[M]@HumanActivity(name=String,begin=String, finish=String, post=String, roles=String[])

[M]@BeginWork(name=String)

[M]@FinishWork(name=String)

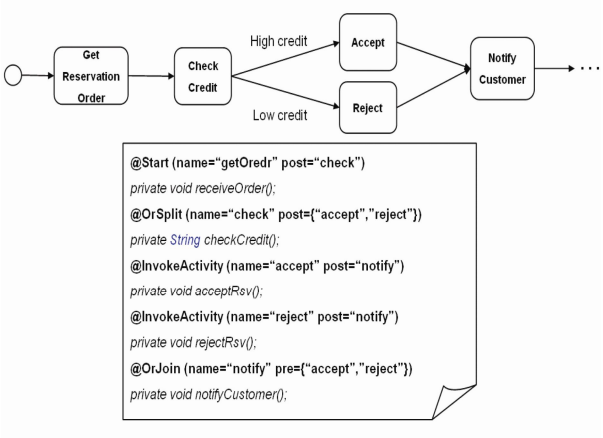

Fig.4　Exclusive choice & simple merge patterns

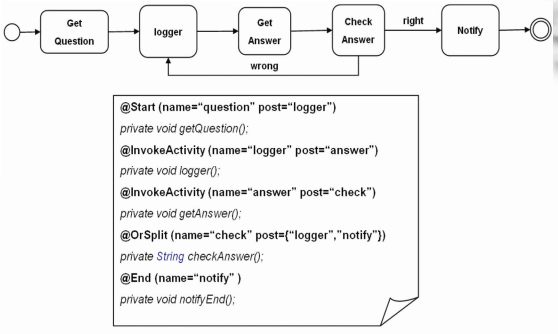

Fig.5　Arbitrary cycles pattern

These three annotations are all used on methods. They work together to describe one human-task activity. The @HumanActivity annotation represents an activity in the process like invoke or receive activity. It has three important attributes: "begin", "finish" and "roles". The "begin" attribute indicates the activity has begun when the annotated method with @BeginWork and name of "begin" is invoked. The "finish" attribute indicates the activity has finished when the annotated method with @FinishWork and name of "finish" is invoked. The "roles" attribute indicates who can perform this activity.

4　Annotation-Based Service Orchestration System

To enable aforementioned annotation based service orchestration, we implemented a supporting system accordingly. In the following, we will illustrate the architecture of the system and process registration/ execution steps respectively.

Fig.6 is the system architecture of the service orchestration system.

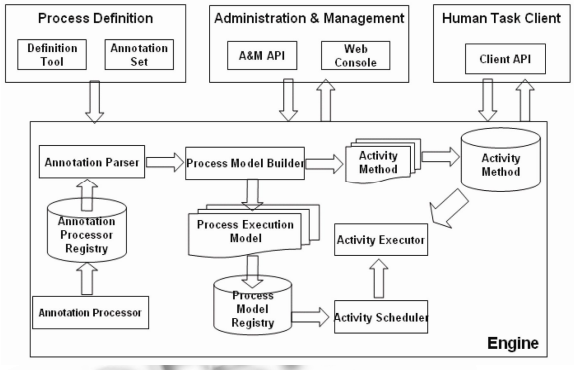

Fig.6　The system architecture

The orchestration system is composed of following main modules:(1) Process Definition (2)Administration & Management (AM)(3)Human Task Client and (4) Engine.

(1) The Flow Definition module is composed of two components. The first one is the visual process editor; it's a convenient tool for process designer to use. The second is an annotation set which provides some built-in annotations introduced in Section 3.

(2) Administration & Management module enables monitoring and management ability for the orchestration system.

(3) We provide a human task client API. The API implements two functions: queries a user's to do list and sends a message to the engine when the users have finished their work.

(4) The engine is the most important module. It provides process registration and execution functions. Fig.7 shows the steps of the process registration: firstly, a programmer registers process class to the engine. Secondly, Annotation Parser extracts the annotation of each method, and then finds the corresponding annotation processor by the annotation type from the annotation processor registry. Thirdly, Annotation Processor reads the attributes of the annotation, such as "name", "post" and etc. Then it adds this separate information to the process definition. Fourthly, when all the annotations are processed successfully, the engine makes a whole process definition. Fifthly, Model Builder builds an execution model as the runtime process controller, such as Petri-Net based or State-Machine based model. The builder also extracts each method from the artifact such as a Java

class. Sixthly, the newly created execution model will be stored in a registry and indexed by the process definition information. Simultaneously, each activity method is stored in a registry and indexed by names.

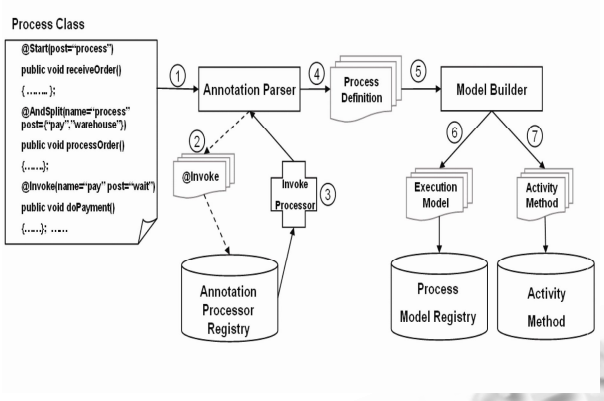

Fig.7 Process registration steps

Compared with XML-based process registration/parsing steps, there are some benefits:

(1) Do not need to parse the verbose XML document, which consumes lots of time and memory.

(2) Do not need to generate new Classes and compile for each translated XML described activity and do not need to deploy these classes and artifact into a run-time container, such as EJB container.

(3) No heavy resource and time burden during debugging a process due to frequent registration/parse steps.

After a process is registered in the orchestration system, an invocation from outside may fire the run-time engine to orchestrate the registered process. Fig.8 describes the execution steps.

Firstly, when there is an invocation from outside, such as a soap-based web service call, the engine creates a flow instance for this conversation. Secondly, Activity Scheduler finds the corresponding execution model for each process instance. Thirdly, the corresponding execution model in charge of the whole execution process orchestrates the ordering of activity invocation. Fourthly, Activity Scheduler dispatches each activity method to Activity Executor with their name. Fifthly, Querying for method command is issued by Activity Executor. Engine finds it in the activity method registry. Sixthly, Activity Executor executes the method as the action of the activity when the corresponding method is found out. Finally, the Activity Executor will return the result to the Activity Scheduler, and the scheduler could schedule the rest work until the last activity is finished.

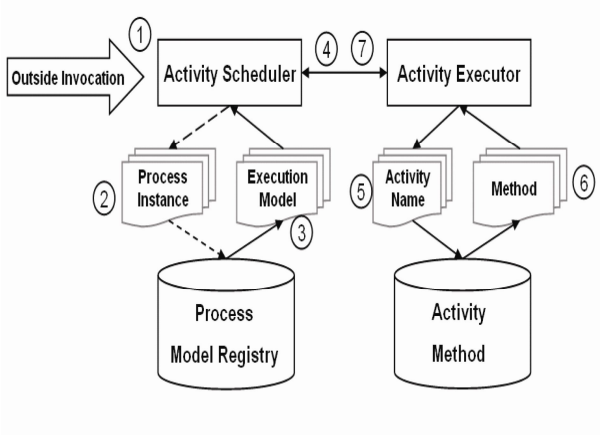

Fig.8 Process execution steps

5 Experiments and Results

To get better understanding of the pros and cons of our approach, we conducted some experiments using both our approach and other existing approach.

The service orchestration scenario in our experiments is a travel booking process, which is showed in Fig.9. The process starts when a customer enters the data for his travel arrangements. Then the credit card information is checked for validity. If the credit card data is incomplete or not valid, the process ends. Otherwise, three different reservations are made for the fight, hotel and car. And then if all the reservations are completed successfully, a confirmation number is generated. Finally, the confirmation number or an error message is returned to the customer who triggers the travel booking process.

To show the benefits of the orchestration system, we recorded the development time using BPELJ with WID 6.0 (WepSphere Integration Developer) and ASO (Annota- tion-based Service Orchestration) with Eclipse.

In Table 1, the total process development time cost, it takes only a half of the time using BPELJ for using ASO, which is 50 minutes. In the process definition phase, we find that using ASO costs nearly 1/3 of using BPELJ. To better understanding the time cost, we divide the process definition phase into 3 steps: define partners, define variables and define & link activities, as showed in Table 2. We find that every step for using ASO is faster than the corresponding one for BPELJ, especially for

activities definition and linkage, which takes 36% of the time of BPELJ.

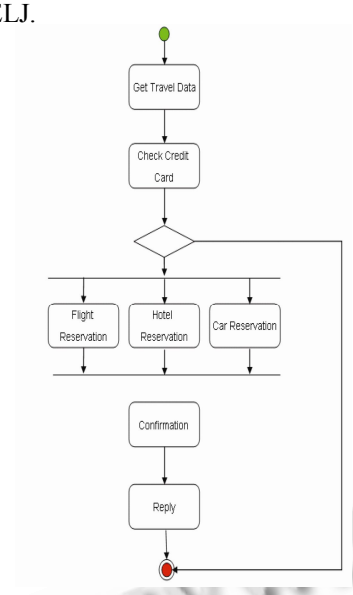

Fig.9　Travel booking scenario

Table 1　Time cost of overall scenario development

	Tool	Develop Process				Total
		Define Data Type	Define Interface	Create Services	Define Process	
BPELJ	WID	9 min.	6 min.	20 min.	58 min.	95 min.
ASO	Eclipse	5 min.	5 min.	10 min.	19 min.	50 min.

Table 2　Time cost of process definition step

	Define Process			Total
	Define Partners	Define Variables	Define & link activities	
BPELJ	9 min.	6 min.	20 min.	58 min.
ASO	5 min.	5 min.	10 min.	19 min.

6　Conclusions

In this paper, we propose an annotation-based service orchestration system. The proposed system can overcome traditional orchestration issues resulted from the separation of orchestration logic from the original system. We explain the main elements of the annotations and the system architecture. To verify the efficiency of the proposed system, we design a service orchestration scenario, and record the development time cost both with BPELJ and ASO. In the experiment, it shows that ASO is an effective orchestration approach that nearly saves one half of the time compared with BPELJ. In summary, with the new annotation-based service orchestration system, developers are expected to develop business process more efficiently based on their current programming language skills.

In the future, we manage to develop an integrated tooling to facilitate the service orchestration logic specification. Moreover, we will try to apply the idea in state machine related orchestration to further explore the feasibility of the approach.

References

1 Peltz C. Web services orchestration and choreography. Computer, 2003,36(10):46－52.

2 OASIS. Web Services Business Process Execution Language Version 2.0.OASIS Standard, 2007:1－264.

3 W3C. Web Service Choreography Interface(WSCI) 1.0. W3C Note 8. 2002.

4 Blow M, Goland Y, Kloppmann M. BPELJ: BPEL for Java. BEA Systems, Inc. (“BEA”) and International Business Machines Corporation (“IBM”). 2004:1-24.

5 Monsieur G, Snoeck M, Lemahieu W. Coordinated Web Services Orchestration. 2007 IEEE International Conference on Web Services. 2007.

6 Leymann F, Roller D, Schmidt MT. Web services and business process management. IBM Systems Journal, 2002,41(2):198–211.

7 Van der Aalst WMP, Barros AP, ter Hofstede AHM, Kiepuszewski B. Advanced Workflow Patterns. Proceedings of the 7th International Confer- ence on Cooperative Information Systems. CoopIS 2000.

8 Lawrence P. Workflow Handbook 1997, Workflow Management Coalition. John Wiley and Sons,New York, 1997.