RC22466 (C0205-004) May 30, 2002
Computer Science

IBM Research Report

A Framework for e-Service Management and Invocation in Application Integration Systems

Xin Zhang, Wei Sun, Sheng Ye, Zhong Tian
IBM China Research Lab
2F, HaoHai, #7, 5th Street
Shangdi, Beijing 100085
China

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

A Framework for e-Service Management and Invocation in Application Integration Systems

Xin ZHANG, Wei SUN, Sheng YE, Zhong TIAN
({zxin, weisun, yesheng, tianz}@cn.ibm.com)
IBM China Research Lab
2F, HaoHai, #7, 5th Street, Shangdi, BEIJING, 100085, CHINA

Abstract

Application integration is the key to building new business process based on heterogeneous, distributed legacy applications or systems. As more and more applications are exposed in the form of e-Services over a network, service-oriented integration is developing on the horizon of the application integration frontier. Therefore, the management and invocation of distributed and diverse-form e-Services become a challenge for application integration system. In this paper, we introduce an Action Invocation Framework to address this problem. By leveraging private UDDI, this framework could facilitate service registration, categorization & relationship management, uniform service invocation interface exposure. It also handles the interface brokering and invocation for diverse forms of e-Services. The embedded transaction support enables the service to be securely invoked and executed, or rolled back when errors occur. In this paper, we introduce the initiatives of developing such a framework, and then the architecture of the framework is presented. At the end of this paper, a self-service banking application scenario is described, in which AIF facilitates the e-Services aggregation and transaction requirement.

1.Introduction

As is well known, enterprise business systems consist of many different technology standards conformance applications, which were built at different stages of the technology evolution and development. To adapt to the dynamic business requirements, the development of new processes, new applications are always on demand. But developing from scratch costs huge direct and indirect investment, so that making use of an application integration system, building new processes and applications by integrating the heterogeneous legacy applications has become the direction[1,2,3]. In a nutshell, application integration system allows an enterprise to build and manage complex business processes or applications by choreographing the interaction of a number of (internal and external) business functions to achieve the business goals. It is becoming a trend that more and more business functions are packaged in e-Service form and exposed over the network. So, service-oriented integration is developing on the horizon of the integration frontier[4]. These services may be defined and exposed in different access modes, and they could be local programs, intra-enterprise software components or Web services provided by the enterprise itself or external partners. So, to an application integration system, the management and invocation of all the new process and application related services are becoming a challenge, which would involve service registration, service interface exposure, interface brokering and invocation. Due to the technology complexity and shortage of skillful service adapter developers, it is essential to construct a framework for application integration systems to facilitate the management and invocation of diverse-form and distributed e-Services.

In this paper, we introduce an Action Invocation Framework(AIF) to address the problem. Here we refer action as the programming level entity of e-Service. This framework manages distributed, diverse-form actions, such as local Java program, enterprise Java bean(EJB) or Web services. These actions should be registed in the Action Registry of the framework according to corresponding registration schema, and the Registry provides uniform interfaces to external service requesters. The Action Invocation Broker(AIB) would perform the interface brokering and fulfill the actual invocation. By leveraging private UDDI, the Registry can provide flexible categorization and relationship management; especially the invocation broker has transaction support function, so that the service invocation could be safely performed or rolled back if errors occur. The system architecture and core technology components of the framework are discussed in length in this paper. A self-service banking application scenario is also presented.

2. Action Invocation Framework Architecture

e-Services could be distributed and exist in diverse forms. Obviously these different forms of services have their own specific invocation methods and interfaces. So, to describe the service invocation interfaces and invoke services case by case is a labored task. If these services could provide uniform interface, the service invocation will be easy and simple. A service registry could facilitate storing and providing the uniform interface to service requestors; the service invocation broker would transform the uniform interface to the specific actual service invocation interface and perform the invocation.

The Action Invocation Framework(AIF) is designed as a reusable service invocation middleware. Here action indicates the access interface of e-service. It is the front tier of the e-service presenting to service requester. Our approach is to handle diversified service access modes within the framework and provide a uniform action invocation interface to upper level service requester. Thus service requester can focus more on its own application logic rather than care much about the difference of service invocation mode[5,6]. Besides the service brokering, the action management is also covered in our framework for better service integration.

As illustrated in figure 1, the system includes both design time asset for action registry and management and runtime asset for action invocation and monitor. The Action Lib stores action definition and relationship between actions, which is managed by Action Definition&Relationship Management component. It leverages the private UDDI as a flexible categorization mechanism which allows user to classify action categorization strategy. The Action Registry provides user the action registration service.

At runtime, the Action Invocation Broker(AIB) acquires the invocation request, it retrieves the relative information of the action to be invoked from the Action Lib. By identifying the action type, AIB requests the corresponding adapter to perform the invocation, and captures the invocation result to return to the service requester. When transaction is needed for the action invocation, the AIB will request the Transaction Function Enabler to record transaction data and process related operations. There Action Invocation Monitor and Control module provide the service requestor interface to control and monitor current invoked service.

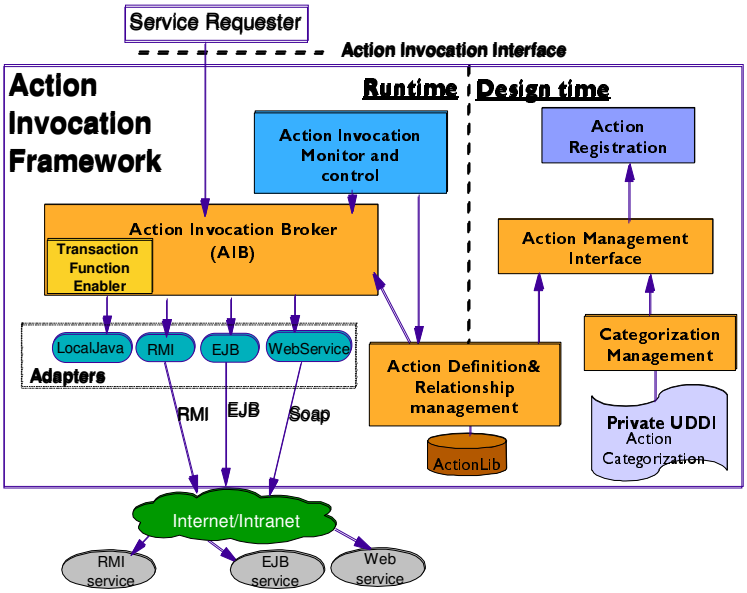

Figure 1. Action Invocation Framework Architecture

Based on the framework, we implement a prototype in Java program, which makes it easier to be reused and migrated in diversified platform.

2.1 Action Description Schema

In the framework, Action is defined as an exposed operation of a certain action target. The action target can be a Web services port, RMI object or Local Java Class and other service types. For Web services, the action is the operation described in WSDL; For RMI, the action is the exposed methods; For Local Java Class, the action is the public method. An action target may expose several actions, but no matter how many actions are defined on one action target, they have the same access method which is determined by the definition of action target.

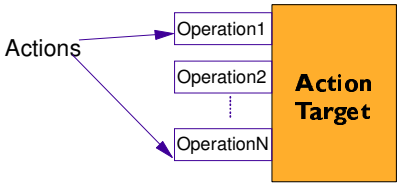

Figure 2. Action Definition

2.1.1 The action target definition

Action target defines the service object either residing on local host or at remote site. Its definition includes service type, and related information needed to find the service object. As illustrated in Figure3, four supported service types are Local Java, Web services, RMI, EJB.

Local Java Class

Field	Description
id	Action target id
Type	="LocalJava"
PackageName	The package name of Java class
ClassName	The class name of Java class

WebService

Field	Description
id	Action target id
Type	="WebService"
ServiceName	Service name
PortName	Port name

RMI

Field	Description
id	Action target id
Type	="RMI"
Host	Host to find RMI
Port	Port of the finding service on the Host
jndiName	Registered JNDI name of the RMI service

EJB

Field	Description
id	Action target id
Type	="EJB"
Target	Finding URL
initContext	Init Context class name
jndiName	Registered Jndi name of the EJB Home object
className	Class name of EJB object
pkClassName	Key class name(optional)
pkArguments...	Parameters to init pkClass instance (optional)

Figure3. Action Target Definition

2.1.2 Action definition

In the action definition, it specifies an operation on an action target. The arguments that are needed to invoke that action are also described.

Table 1. Action Definition

Field	Description
ActionID	Action id
Desc	Description
targetID	Action target id
operName	Operation name
Arguments...	Parameters of the operation
Returns...	Return values

2.2 Action Invocation Broker

Action Invocation Broker(AIB) brokers service invocation for service requester. As shown in figure 5, when an invocation request is submitted to AIB through Action Invocation Interface, the Invocation Assembly and Validation module assembles the invocation request and checks its validity by querying Action Lib, then translates the request into an internal request command. The Invocation Handler handles the request command and triggers corresponding adapter to perform the invocation. The Monitor and Control Agent provides interface to manage AIB. When transaction feature is required, the Transaction Function Enalber is involved to manage action invocation chain at the demand of Invocation Handler.

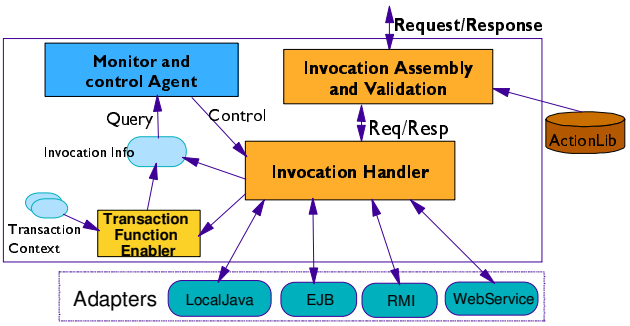

Figure 4: Action Invocation Broker

2.3 Action Invocation Interface

In our implementation, three layers of interfaces are provided: as we take Java as the implementation language, a Java API is defined as the Native Action Invocation Interface. For service requester that is also constructed in Java code or support Java call, it can leverage this interface for more efficiency and better data type supporting; A Generic Action Invocation Interface is also defined as a command line interface which combines invocation request and related parameters into an XML document to minimize the requirement for service requester; The third interface is so called Remote Action Invocation Interface which is provided by wrapping AIB itself into a Web services so that it can be accessible through network. The basic structure of an invocation request is as follows:

Table 2. Basic Structure of Invocation Request

Field	Description
ActionID	Action id
Arguments	Action parameter list, in name/value pair
Returns	Action return value list, in name/value pair
InvocationMode	Invocation mode(Sync/Asyn)
TransactionID	Parameters of the operation

2.4 Service Categorization and Relationship Management

In an application integration system, normally service requestor needs a tool to search available services and acquire information about where to find them, and how to use them. As building complex services by embedding other ones may result in mutual dependence, all services should be managed for reference integrity. To answer these challenges, Service Registry is designed in this framework to facilitate all the service management work.

In private UDDI, each service type is identified by a registered *tModelKey*. Each Web services can be published as a certain service type by given *CategoryBag*, thus allowing us to search and manage published Web services[7]. By leveraging the flexible category management mechanism provided by private UDDI, the Service Category Manager provides user interface to define the service categories. All the services would be listed under specific category, which would facilitate the service searching and categorization. Category is something related to the properties of actions. It provides an easy way to organize, search, and manage actions. User should register all services into the Registry through registry

management interface, in which the invocation method, interface, function descriptions, transaction utilities' interfaces of these services are defined and kept. Then actions can be searched by their type, owner, invocation method, register time, etc.

There could be mutual dependency and cross reference among services on an application integration system. This kind of relationship is managed by the Services Relationship Manager. The relationship is defined at design time when a service is registered in the service registry, and then recorded. Because of every service in the registry are identified by a unique ID, so the cross reference would not be influenced even if the referenced service interface is updated. A service can not be delete from the registry if it is referenced by other services. So, the service relationship management could help to maintain the services reference integrity in the registry.

2.5 Transaction Support

Transaction capability is more and more in demand for application to handle the execution of mission critical process. For example, a finished payment operation shall be rolled back if a logistics operation is proven to be failed later in a purchase process. When an application is composed by choreographing diversified services, the generic requirement emerges to achieve application level transaction by leveraging the transactional capability of underline individual services. It is really a labored work for application to handle all kinds of transactional/non-transactional actions. The challenge exists that the long-running feature of application level process demands for a relaxed transaction model.

AIB meets the gap by devising the Transaction Function Enabler(TFE). The TFE module wraps different services transaction capability and presents a uniform interface for upper service requester. Moreover, it tracks the service invocation sequence and parameters in a transaction so as to automatically perform the rollback and commit operations at application's demand. Thus it simplifies the enablement effort for application level transaction support. Currently only flat transaction model is supported.

For services that will be invoked in a transactional process, they shall register their transaction capability in action lib. For some types of action targets which can provide transaction capability for upper application in their own mechanisms, its supporting mode and related parameters shall be registered. For example, on the base of EJB transaction capability, application level transaction can be enabled by client-demarcating transaction programming[8]. AIB also defines an easy to implement interface for two-phase transaction support. By specifying transaction operations include begin, prepare, commit, rollback operations in Action Target definition, a service-based two-phase transaction support can be constructed. Compensation model is also supported by defining an undo operation on action base[9].

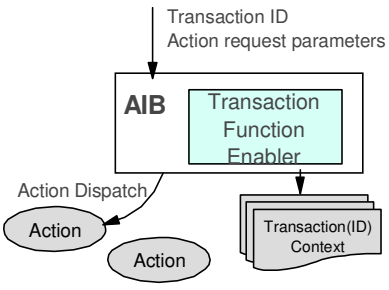

Figure 5. Transaction Context

When a service requester begins a new transaction by requesting AIB, AIB delegates the request to the TFE. TFE initializes a new instance of transaction context and assigns a

unique transaction ID to the requester. The following action invocation requests in that transaction shall be accompanied by that transaction ID. TFE will handle diversified transaction supporting method of lower level services. It performs the begin transaction operation on each action target each time the action target is first instantiated. And it invokes related operations according to the request of application level transaction. The parameters and status of invoked actions in the same transaction are tracked by TFE in the transaction context.

When a transaction meets its end, the service requester informs AIB to finish the transaction by specifying its transaction ID. TFE checks the corresponding transaction context and scans for executed actions' context. TFE commits those two-phase commit actions after voting, and skips those compensatory actions. After commits, TFE clears the transaction context. When a transaction need to roll back in case that an error is encountered, the requester sends a rollback request to AIB with the transaction ID, TFE scans the context of executed actions, rolls back requests of those two-phase commit actions and invokes compensation actions to undo the effect. The rollback sequence is in reverse order of their executing sequence.

Through wrapping service transaction capability and tracking their invocation, TFE enables an easy way as a flexible choice for service requester to implement the application level transaction.

3. Application Scenario

Let's have a close look at how AIB brokers service invocation in a self-service banking application. In this scenario, a bank wants to provide an easy-to-use stock purchase service to its customers. The bank has already owned a finance assistant service to serve customers to transfer and exchange money from their account. That enables online trade between customer and stock agent. With an external stock information service and an additional purchase report component, the stock purchase application can be built by integrating these services. The application is hosted in the bank to serve customer with timely stock information and deal the purchase request at customer's demand. A customer first navigates stock information online. After he selects stock and inputs the total amount he wants to purchase, the application begins to check his account. If local currency is insufficient for the purchase, the deficient amount shall be supplied by changing the foreign currency in customer's account into local currency. Then the application withdraws the money from the customer's account, and deposits money into the stock agent's account. At the end of the purchase process, the result will be reported to the customer. If something goes wrong during the purchase, the purchase need to be canceled and both customer's account and agent's account shall be recovered to original status.

Using an application integration system, the stock purchase process aggregates five actions from two service providers, the bank and stock purchase agent. Given that the process is hosted in the bank company, these actions are provided in different modes:

- Query online stock information, provided as Web services by stock service provider
- Withdraw money from customer's account, provided as RMI service by bank
- Exchange currency, provided as RMI service by bank

- Deposit money in the agent's account, provided as RMI service by bank
- Report dealing result, provided as Local Java class by bank

The actions are registered in the Action Lib as illustrated in figure 6.

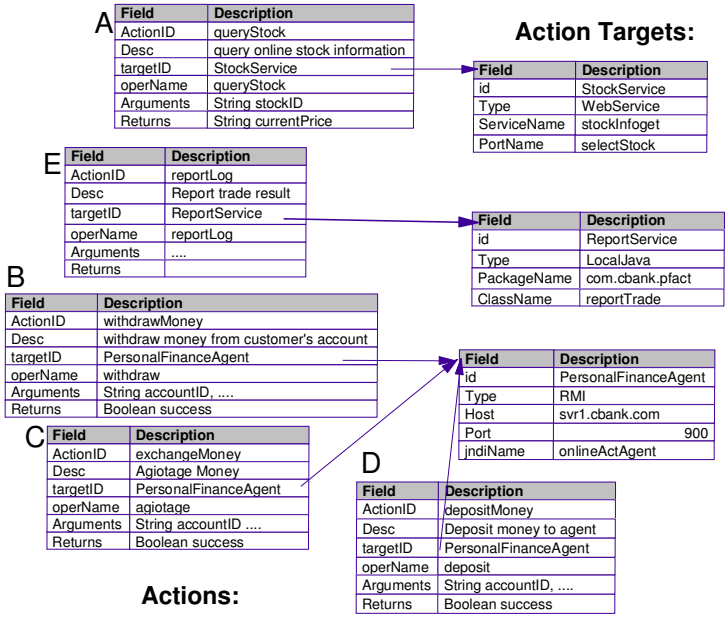

Figure 6. Registered Actions in Action Library

To guarantee the secure operations on the customer bank account, a transaction is needed during the purchase. As shown in the figure 7, we define a transaction which contains action B, C, and D in the process. They are actions in one action target which supports two-phase commit.

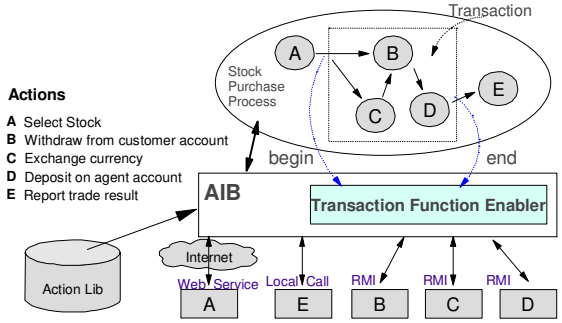

Figure 7. Stock Purchase Scenario

When a user comes online, a new instance of the purchase process is initialized to serve him. Firstly action A is invoked to help the customer to navigates stock information. When the customer submits his purchase request, the transaction begins. AIB receives a transaction start request from application and then informs the TFE to prepare transaction context and return a unique ID. Since the local currency of the customer is insufficient, the exchange

currency action(action C) is invoked. Since it's the first time to invoke the action on this action target, AIB begins the action target's transaction, and saves the action target instance in the transaction context. When the exchange currency action finished without problem, the process goes on to withdraw money from customer's account(action B). Recognizing that this invocation has the same transaction ID, TFE invokes action B in the same transaction on the action target instance as action C. The same thing happens to action D that deposits on agent account. If everything is fine, the process sends transaction finish request to AIB. TFE checks the transaction context, finds all listed action targets, and performs commit operations on all participated action targets if voting with success, otherwise rolls back each action target. If the transaction is committed successfully, the process generates a report to customer(action E). If the transaction is failed, it is rolled back and an error report will be generated.

5 Summary

Application integration system is believed to be the key to aggregating existing applications so as to composite new business service within enterprise or among trading partners. As distributed computing technologies are getting mature, more and more applications are provided in the form of e-services over the network. Obviously, the distributed and diverse-form e-services should be managed to facilitate easy access by the application integration system. AIF introduced in this paper is designed as a framework to provide the service management and invocation functions. Service registration, categorization and relationship management, uniform service invocation interface exposure, diverse interfaces brokering and transaction support are the core advantages of this framework. Although it is originated from the service management requirement of application integration system, it also suits for applications which need to invoke and manage existed e-services.

Reference

[1]Recoupling B2B Investments. p77-p79, May 2000, eAI Journal
[2]Colin,Osborne: Integration is Everything. p25, January 2001, eAI Journal
[3]Greg, Olsen: An Overview of B2B Integration. p28-p36, May 2000, eAI Journal
[4]Meeting the Enterprise Challenage: The State of Integration in Todays Business World, Hurwitz group
[5]WFMC Workflow Reference Model. http://www.wfmc.org/standards/docs/tc003v11.pdf
[6]WFMC Workflow Client Application Application Programming Interface (Interface 2 & 3) Specification http://www.wfmc.org/standards/docs/if2v20.pdf
[7] UDDI Technical White Paper. http://www.uddi.org
[8] Java Transaction Service API 1.0, sun.com. http://java.sun.com/products/jta/index.html
[9] H. Garcia-Molina, K. Salem:*Sagas*. Procs. ACM-SIGMOD, California, 1987.