Towards Patterns of Web Services Composition

Presented by: Kevin Tsang

Based on a paper with the same name authored by B. Benatallah, M. Dumas, M-C. Fauvet, F.A. Rabhi, available at: http://citeseer.ist.psu.edu/472153.html
Introduction

- The term *(Web) service* denotes an *abstraction* of a set of computational and/or physical activities intended to fulfill a class of *customer needs* or *business requirements*.
- It provides an *interface* to access functionalities offered by *information systems, application programs*, and *business process*.
- Enterprises are continuously discovering new opportunities to form *alliances* with other enterprises, in order to *share* their *costs, skills* and *resources* by offering integrated services (composite services).
- The lack of *high level abstraction* for *Web service integration* has triggered a considerable amount of research and development efforts.
- The report summarized a number of *design patterns* for the definition and implementation of service integration.
Review of enabling technologies

- Service composition is an active area of research and development in different fields:
  - Component-based frameworks
  - Cross-enterprise workflows
  - Electronic Data Interchange
  - XML-based B2B frameworks
Component-based Frameworks

- E-commerce applications rely on distributed object frameworks such as CORBA, DCOM, EJB and other state-of-the-art technologies such as Enterprise Application Integration (EAI) and Enterprise Resource Planning (ERP)

- EAI suites provide standard data and application integration facilities (e.g. pre-built application adaptors, data transformations, and messaging services among heterogeneous system)

- ERP systems provide a single, homogenous solution for a number of back-office applications
Cross-enterprise Workflows

- Automate business processes that interconnect and manage communication among disparate systems

- New emerging service composition projects consider loosely coupled services (CMI, EFlow, CrossFlow, Mentor, CPM, SELF-SERV, ADEPT)

- These projects consider critical requirements of B2B e-commerce such as dynamic selection, adaptability, and external manageability of services
Electronic Data Interchange - EDI

- EDI is the interorganizational application-to-application transfer of *business documents*

- EDI documents are structured according to a *standard* and *machine-processable format* (e.g. ANSI X12 and UN/EDIFACT)

- Mostly used for the automatic transfer and processing of documents in industries which *trade on high volumes* (e.g. goods transportation, food manufacturing, and automobile production)
XML-based B2B Frameworks

- Provide a **common format** to publish and exchange **business information** over the Internet
- To support B2B interoperability, describe the **semantics** and **structure** of data and operations of services using XML & domain ontologies
- **Ontology** defined terms to describe **entities** (e.g. service properties, operations) of a specific domain (e.g. healthcare, finance, travel) and **relationships** among terms
- Some organizations (e.g. RosettaNet) developed common ontologies for different industries
- E-commerce platforms that rely on XML-based standards and protocols including IBM WebSphere, WebMethods, Sun ONE, and BEA Collaborate
Patterns of Service Composition

- Elementary Service-based *Interactions*
  - The External Interactions *Gateway* Pattern
  - The Contract-Based Outsourcing Pattern

- Service *Composition*
  - Service *Composition* Pattern
  - Service *Discovery* Pattern

- Composite Service *Execution*
  - *Central* Authority Pattern
  - *Peer-to-Peer* Execution Pattern
Elementary Service-based Interactions

- In the setting of B2B e-Service, it is the interaction of Information System (IS) between service provider and service consumer.

- Their IS are heterogeneous in both the managerial and technological viewpoints.

- Service provider needs to make sure their IS has a clearly defined interface to their e-service.

- Service consumer needs to make sure their IS interact properly with the e-service interface.
The External Interaction Gateway Patterns

- Each of the services provided by the organization has its own interaction requirements (e.g. document formats, data model, domain ontologies, message sequencing)

- Issue arise in this situation:
  - For different data model and format of business document, how the conversion between formats operated?
  - For different interaction protocols, how to ensure proper interaction between applications?
  - For the exchange of critical business information, how to ensure the confidentiality, integrity and non-repudiation?
The External Interaction Gateway Patterns

Solution: using a software entities called as *External Interaction Gateway* (EIG)

- Internal architecture of EIG:
The External Interaction Gateway Patterns

- Handling *document format heterogeneity* based on separation between syntax and data model of a standard
- The *syntax* of a document standard is specified as an XML DTD or an XML Schema
- The *data model* is specified in the RDF Schema Language
- Transformation of document XD (with XML standard S) into document XD’ (with XML standard S’)
  - Abstraction: XD $\rightarrow$ RD (data model of S)
  - Conversion: RD $\rightarrow$ RD’ (data model of S’)
  - Refinement: RD’ $\rightarrow$ XD’ (syntax of S’)

XD – XML Document
RD – RDF Document
The Contract-Based Outsourcing Pattern

- Contract is a *planned set of actions and interactions* that need to undertaken during the delivery of a service
- Contracts for a given service are abstracted into *contract templates* with a set of parameters
- Contract templates are included in the *advertisement* of a service offer
- Typical steps:
  - Queries *service catalog(s)*
  - Retrieves *service offers* with their *contract templates*
  - Instantiates the contract by providing a set of *parameter values*
  - For special requirement, *negotiation* for contracts with providers may be needed
  - Execute the contract through *contract enactment module*
The Contract-Based Outsourcing Pattern

- **Known Implementations**
  - CrossFlow – *contracts are statically specified* (no dynamic negotiation) by service providers and advertised in a service marketplace
  - MEMO – exchange standardized messages based on *speech-act theory*, structure, sequencing and semantics of the message exchanged during the negotiation is fixed
  - ADEPT – using one-to-many negotiation framework based on *multi-attribute utility theory*, each agent try to *maximizes its own utility function* which encodes the preferences and business constraints of the organization
Service Composition

- Fast and dynamic integration of business process is an essential requirement for organization

- Business partners with *permanent* (long term) relationships
  - Components are known in advance and alliances are statically defined
  - Static composition of service is sufficient

- Business partners with *temporary* (short term) relationships
  - Not assume an a priori trading relationship among partners
  - Dynamic composition of service is needed
Service Composition Pattern

- Important characteristics for static composition:
  - Describe interaction of services without referring to any implementation or execution model
  - Support nesting of composite services
  - Maintain a high level specification of a composite service while ensuring its executability

- Solution: aggregation specification with control flow and data flow specification
Service Composition Pattern

- Use of statechart as a *formal notations* for *workflow specification*
- Statechart made up of *states* and *transitions* with *Event-Condition-Action (ECA) rules*
- Example of control flow specification of “Travel Solutions” using statechart:
Service Composition Pattern

- **Known Implementations**
  - CMI – service is modeled by *state machine* that specifies the possible states of a service and their transitions
  - EFlow – composite service is modeled as a *graph*, it defines the order of execution of service component among the nodes (service, decision, event)
  - WebBIS – adopts an *ECA-rule* approach for defining composite service
  - SELF-SERV – use a subset of *statecharts* to express the control-flow of composite services
Service Discovery Pattern

- Problem relates to Web-based service integration in large, autonomous, heterogeneous, and dynamic environments

- Important characteristics for dynamic composition:
  - Information to identify service components at run-time
  - Integrate component services with a high level specification of composite service

- Solution:
  - automated service discovery facility
  - composite service specification allows automatically discovery of service components
Service Discovery Pattern

- **Known Implementations**
  - CMI – use *placeholder activity* as an abstract activity that will be *replaced* at runtime with a concrete activity, *selection policy* is specified to choose the best implementation
  - EFlow – service node contains a search recipe, it is a query represented in a query language
  - WebBIS – use a concept of *push-community* which describes the capabilities of a desired service, actual service can *register* with one or several push-communities, need a *mapping* of the operations in the community and the actual services
Composition Service Execution

- Execution of a composite service assuming that its control and data semantics are already defined

- Execution involve the activation of all its component services hosted on a number of remote providers

- Two possible execution patterns
  - Components are coordinated by a central scheduler
  - Coordinate the execution through peer-to-peer communication
Central Authority Execution Pattern

- Provider of composite service S should hold a **Composite Service Scheduler**

- The scheduler responsible for:
  - *Invoke* each of S’s *components* according to the order and conditions in control flow specification
  - Receive and processes *service requests*
  - Handling and *processing data* according to the data semantics of composite service
Central Authority Execution Pattern

- Example of centralized execution of “Travel Solutions” service
Central Authority Execution Pattern

Known Implementations

- ADEPT – a workflow can be recursively decomposed into sub-workflows, leading to a tree structure

- EFlow – execution model is based on centralized process engine, not support recursive definition of composite services
Peer-to-Peer Execution Pattern

◆ Responsibility of coordinating the execution of a composite service is distributed across the providers
◆ A software components called coordinators are hosted by each of the providers
◆ Coordinator responsible for:
  ◆ **Initiate** the execution of service components
  ◆ **Notify** the completion of this execution to the next coordinators
  ◆ **Interrupt** the service execution during the occurrence of a certain external events

```
    Gateway
     0..N
    +--------+---------+             +------------------+
    |        |          |       +------------------+
    |        |          |       |                |
    +--------+          +------------------+
          |          
          |          
          v          
    Coordinator
     1..N      delegates control to
      +--------+          1
      |          |          |
      |          |          |
      |          |          |
      v          v          v
    Composite Service Scheduler
       0..N             0..N
```

invokes

delegates control to

derived from

invokes
Peer-to-Peer Execution Pattern

- Example of distributed execution of “Travel Solutions” service
Peer-to-Peer Execution Pattern

- **Known Implementations**
  - SELF-SERV – responsibility of coordinating the composite service execution is distributed across several *lightweight software components* hosted by the service providers.
  - CPM – support the execution of inter-organizational business processes through *peer-to-peer collaboration*.
  - Mentor – *partition* the overall workflow specification *into several sub-workflows* and distributing the execution of the sub-workflows.
Conclusion

- Discussed a number of patterns for the definition and implementation of service integration
- These patterns suggest a methodology for building a new composite service
  - Identify elementary services and expose them through a gateway interface (External Interactions Gateway Pattern)
  - Specify control and data flow semantics of the new service based on these elementary services or other composite services (Service Composition Pattern)
  - Component services can be identified at run-time (Service Discovery Pattern)
  - Coordination for the execution of composite service can be centralized or distributed across the service providers (Service Execution Patterns)