Discovery-Based Service Composition *

Jonathan Lee, Shang-Pin Ma, Shin-Jie Lee, Yao-Chiang Wang and Yin-Yan Lin
Software Engineering Lab, Department of Computer Science and Information Engineering
National Central University, Taiwan
{yjlee, albert, jie, ycwang, slose}@selab.csie.ncu.edu.tw

ABSTRACT

The focus of this paper is to propose a discovery-based service composition framework to better integrate individual services in both static and dynamic manners. In the making of this framework, we emphasize two key features: (1) Shape the lifecycle of service composition for capturing the essence of service-oriented computing. (2)Extend Contract Net Protocol to coordinate activities of service discovery, service composition, and service invocation based on the service composition lifecycle.

Keywords: Business Process Execution Language for Web Services (BPEL), Contract Net Protocol, Service Composition, Service Composition Lifecycle, Service Discovery, Service-Oriented Architecture (SOA), and Web Service.

1: INTRODUCTIONS

Service-orientation is a trend in software engineering that promotes the construction of applications based on entities called services [4]. Nowadays, many industrial and academic research work have been developed to solve some issues for service-oriented technologies, such as service discovery, description and invocation (message passing), and service composition. However, service composition that enables one to aggregate or compose existing services into a new composite service is still highly complex but critical task in service-oriented technologies. Several key challenges in service composition need to be addressed:

- How to facilitate the discovery of services? In real world, there are usually multiple services which offer seemingly similar features but with some variation (e.g., different service interfaces, different attributes, different quality, etc). If we can't locate all possible services which can take over each other, we can't design and execute composite services appropriately.
- How to enhance reliability of composite services? The services together with their communication links and information sources may appear and disappear dynamically in the Internet. A composite service needs to query for new appropriate services dynamically and proactively, and completes its tasks while some constituent services fail to work.

In order to address these challenges, we propose a discovery-based service composition framework, including a proposed service composition lifecycle for revealing the process and state transitions of service composition, and Extended Contract Net protocol (ECNP), to monitor and control service composition based on the lifecycle.

The organization of this paper is as follows. Background work that is related closely to this research will be outlined in the next section. In section 3, the proposed discovery-based service composition framework will be introduced. In section 4, some important and extant service composition approaches will be introduced. Finally, we conclude this proposed approach.

2: BACKGROUND WORK

2.1: Issues of Service Composition

Many researchers have pinpointed many issues of service composition, such as coordination, transaction, context, conversation modeling, execution monitoring, etc..[7] We summarize several important observations as follows:

Cervantes and Hall [3, 4] believe that service compositions must handle issues relating to service discovery and service dynamics. They mention specially that recovery from a service departure can also be achieved through self-adaptation techniques. Our research will focus on both service discovery and service dynamics that they stressed.

J. Yang and M.P. Papazoglou et al. [17] believe that e-service composition has several characteristics which makes it very different from workflow integration and software component integration. These can be summarized: (1) The service space is normally large and dynamic and new services become available on a daily basis. (2) There are usually multiple services which offer seemingly similar features but with some variation. (3) Composition of services needs to be generated on the fly based on the requests of the customers. Our

*This research was sponsored by Ministry of Economic Affairs in Taiwan under the grant 95-EC-17-A-02-S1-029

research will tackle these issues and provide some solutions.

Charfi and Mezini [5] mentioned that evolution of the service composition must be paid attention to, such as changes in the environment, technical advances like updating of web services, addition or removal of partner web services, and variation of non-functional requirements. Solving the problems caused by the changes of service composition is also a key factor in this research.

2.2: Contract Net Protocol

The Contract Net Protocol (CNP) [14, 12] is a specification of problem-solving communication and control for nodes in a distributed problem solver. Its significance lies in that it was the first work to use a negotiation process involving a mutual selection by both managers and contractors. In contract net, the agents were totally cooperative, and selection of a contractor was based on suitability, adjacency, processing capability, current agent load, etc.. Main steps in contract net protocol are (1) Request for bids: A manager requests a bidder to submit a bid. (2) Submission of bids: The bidder prepares a bid and submits it to the manager for evaluation. (3) Awarding of contracts: The manager evaluates the bid, which could (or not) be awarded as a contract to the bidder. (4) Acceptance of contracts: If the contract is awarded, the bidder is requested to accept (or decline) the execution of the contract. (5) Submission of results: The bidder submits the results of executing the contract.

Our research will extend CNP to coordinate activities of service composition, service discovery, and service invocation.

3: DISCOVERY-BASED SERVICE COMPOSITION FRAMEWORK

In the beginning of this work, we identify some principles of service-orientation, especially for service composition. The service composition issues mentioned previously have consolidated in these principles. These principles must be satisfied and as the high-level requirements of our approach, described as follows:

- Recursively Compositional: The composite service can be as a constituent service of another composite service.
- Service Reusable: Constituent services can be incorporated within different compositions or reused independently by other service requestor.
- Service Discoverable: The constituent services can be discoverable via the mediation service.
- Service Substitutable: Constituent services can be substitutable instead of fixed (by means of compliant interfaces). Furthermore, constituent services can be decided based on the requests of end user at runtime, and can be replaced appropriately when it is missing or malfunctions.
- Contract-Driven: There must be contract relationships between the composite service and the constituent services to ensure the availability and reliability of constituent services.

To realize the identified requirements, we define the service composition lifecycle, and develop our solution based on the lifecycle.

3.1: Service Composition Lifecycle

We use two views to shape the lifecycle of service composition for capturing the essence of service composition: from the workflow perspective and from the state transition perspective. We will depict the two perspectives in the following sections.

3.1.1: Service Composition Process

From the workflow perspective, we believe that a complete composition process should include the following activities at least: (1) Composite Service Definition: The activity for building the composite service schema or an executable composite service. (2) Service Deployment: The activity for deploying executable composite services in the service execution engine. (3) Service Execution: The activity for performing tasks of the composite service in the execution engine.

However, only composition-related activities are not enough. Service's dynamics nature must be stressed by integrating service discovery activities. We will detail how to combine service composition and service discovery in the next two sections.

Static Service Composition

Static composition is the style that services to be composed are decided at design time [7]. If we just consider static composition, service discovery will needed when service developer want to obtain constituent services to produce a (concrete) composite service. Interface matching (matching by inputs and outputs) and conditional matching (matching by non-functional conditions, such as cost, performance, etc.) will both be utilized according to the developer's needs.

Dynamic Service Composition

Dynamic composition is the style that services to be composed are decided at runtime [7]. If we want to dynamically discover the best available services that fit the needs of a specific customer, dynamic composition technology must be ready. All three phases in dynamic composition must rely on service discovery:

- Composite Service Definition: Before invoking the composite service, we need to produce the abstract composite service flow

(i.e. the composition schema). To construct the flow, interface matching is needed to assure the interfaces of the constituent services in this composition.

- Service Deployment: When we want to deploy the composition, all required service bindings must be ready. Semantic matching is required for selecting most suitable concrete constituent services. The concrete service can be replaced according to the user's requirements.
- Service Execution: If some constituent service leaves or malfunctions, interface matching or semantic matching can be re-performed.

3.1.2: State Transitions of the Composite Service

The second perspective of service composition lifecycle is expressed by the state transitions of the composite service. To understand the state transitions of a composite service, we should analyze what are the building blocks of a composite service first. Three main components can be concluded to form a composite service (see as Figure 1).

- Composition Schema: Composition schema is to describe the control flow and data flow between the constituent services in the composition.
- Constituent Service: Constituent service is the component service that will be synthesized in the composition. A composite service can be as a constituent service of an other composite service.
- Service Proxy: Since the composite service is just a client of the constituent services, not really incorporates with the constituent services, the composition will contain several service proxies which are responsible for invoking constituent services. Additionally, because of the loose coupling characteristic of service-orientation, each service proxy can bind different concrete services as long as they have the same service interface.

Based on the static structure of a composite service, we can depict the state transitions of the composite service by depicting state transitions of the constituent service and the service proxy separatively.

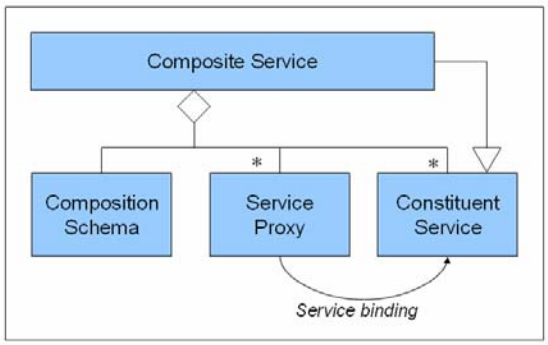

Figure 1: Conceptual Model: Relationship between constituent services and service proxies

Lifecycle of Constituent Service

The state transitions of a constituent service construct the standard lifecycle of a service component. The detailed descriptions of states and state transitions (see as Figure 2) are briefed as follows:

- Waiting for Execution: The constituent service is capable of accepting and processing requests (i.e. the service is available) in this state. When service requests are coming, the service will transit to "Service Execution" state.
- Service Execution: In this state, the constituent service will process requests, perform tasks, and send back the service results. Generally, if the execution is performed successfully, the service will transit back to "Waiting for Execution" state. Otherwise, the service will transit to "Unable" state if the event of service termination occurs.
- Unable: The service component is not capable of accepting any requests (i.e. the service is not available) in this state.
- Registered: In this state, advertisement of this service is published to the service registry.
- Unregistered: Service advertisement of this service will be removed from the service registry in this state. The service can transit between "Registered" state and "Unregistered" state mutually.
- If the service reaches "Unable" and "Unregistered" states simultaneously, the service will transit to the final state (dead state) automatically.

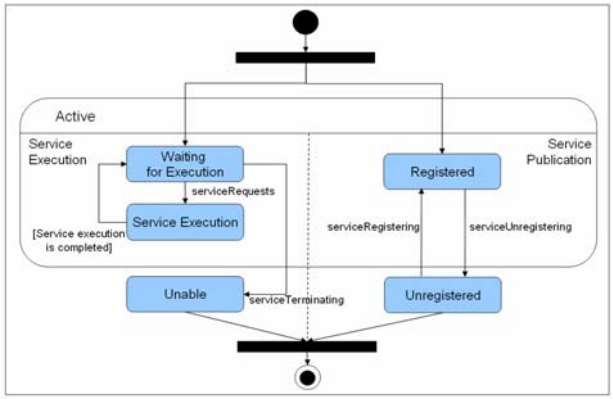

Figure 2: State Transition Model of Constituent Service

Lifecycle of Service Proxy

For the purpose of binding constituent services, a composite service should include service proxies to invoke constituent services. We believe that a service proxy should not simply be enabled to invoke services, more responsibilities should be added to the service proxy for flexible binding. Based on the notion, the states and state transitions of the service proxy in a composite service are described as follows(see Figure 3):

- Locating Services: We assume that the composite service schema will be ready before service deployment and service execution. All constituent services required in the composition should be discovered/located first to assure the service availability. If no qualified service can be found, service proxy will transit to "Inactive" state, and waiting for "relocating" event.
- Ready: When completing service discovery, candidate services should be attached to the service proxy, the concrete service with highest score should be bound as the default service, and service proxy will transit to the "Ready" state. If all service proxy is in "Ready" state, the composite service can be deployed. If all candidate services are missing or fail to work in this state, the service proxy will transit back to the "Locating Service" state.
- Service Invocation: when all chosen constituent services are bound (the constituent services must be in "Active" state") in the composition, the composite service can be invoked, and delegate its tasks to constituent services when accepting requests. Generally, if the invocation is performed successfully, the service proxy will transit to the "Ready" state. But some alternative scenarios will be triggered in some occasions:
  - If the composite service is expired, the service proxy will transit to the final state.
  - If the constituent service is missing or malfunctions, the service proxy will transit to "Locating Services" state for re-discovering constituent services.
- Inactive: If the service proxy can not bind a concrete service, the service proxy will be transit to "Inactive" state, and waits for "re-locating" event or the composite service is expired.

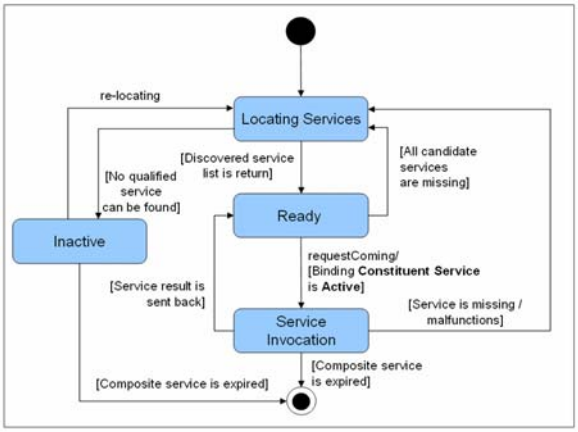

Figure 3: State Transition Model of Service Proxy

3.2: Extended Contract Net Protocol

From the two aspects of service composition lifecycle, we have known there are several complex activities and interactions when completing the whole service composition process, and complex state transitions must be maintained during the lifetime of composite services. How to coordinate complex interactions between service composition activities and service discovery activities to perform both static and dynamic composition? How to monitor and manage the state transitions of composite services? We extend contract-net protocol, called ECNP, to achieve the above goals. On the basis of the service composition lifecycle, we break ECNP into four scenarios: making contracts, awarding tasks, normal decommiting, abnormal decommiting, described by Sequence Diagram in UML 2.0[8] as follows:

- Making Contracts: In this scenario, SCE (Service Composition Engine) will make contracts with constituent services (by mediation of SME (Service Matchmaking Engine)) to ensure the support for using of those constituent services in the future. Steps in this scenario (see Figure 4) are as follows:
  1. Service Provider advertises web services to SME.
  2. SCE requests for services that match the proposed I/O interfaces.
  3. SME replies the matched services to SCE.
  4. SCE requests SME for proposals of the matched web services.
  5. SME replies bids to SCE if the matched web services agree to make contracts.
  6. SCE makes the contracts and sends the contracts to SME.
  7. SCE advertise the composite service to SME.

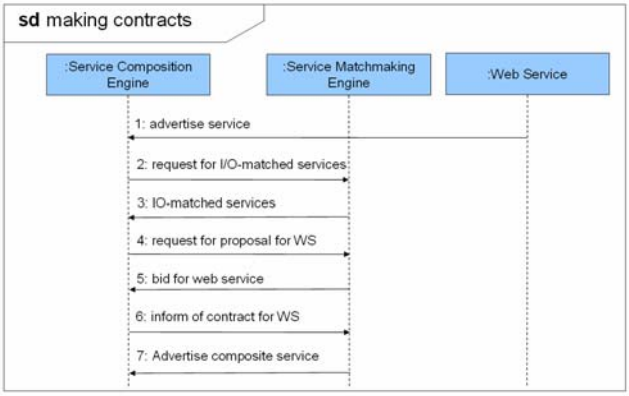

Figure 4: Making Contract

- Awarding Tasks: While SCE receives a request (by the service requestor), it starts to award tasks to constituent services. The key step in this scenario is that SCE will request SME for appropriate services again in a semantic manner before awarding tasks. Steps in this scenario (see Figure 5) are as follows:
    1. SME forwards requests for the composite service to SCE.
    2. SCE searches the appropriate services by querying SME after receiving a service request.
    3. SME returns the appropriate services to SCE.
    4. SCE chooses the service with highest score towards to each activity in the composition.
    5. SCE awards the tasks to the chosen web services.
    6. The matched web services start to fulfill the task and return the task results to SCE.

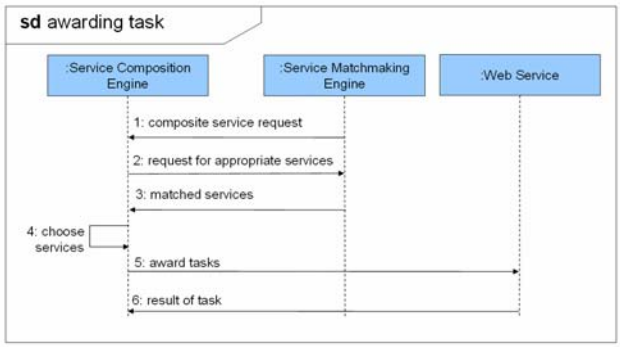

Figure 5: Awarding Contract

- Normal Decommiting Scenario: If the provider of some constituent service wants to repudiate the contracts, SCE can take the actions described in the scenario (see Figure 6). Steps in the scenario are as follows:
    1. The web services requests SME for unadvertising.
    2. SME decommits the web-service contracts to SCE.
    3. SCE decommits the contracts and replies the agreements to SME if it agrees the decommitment.
    4. If there is still available contractor services, perform Awarding Task scenario again (binding different constituent service).
    5. If there is no available contractor service, perform Making Contract scenario again to ensure the availability of constituent services, and perform Awarding Task scenario afterwards.

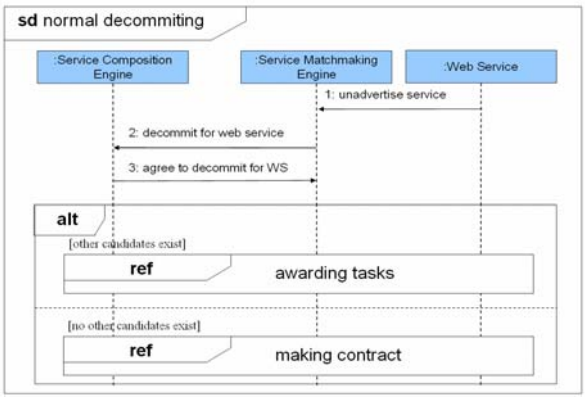

Figure 6: Normal Decommiting

- Abnormal Decommiting Scenario: Constituent services may decommit the contracts without unadvertising itself. While SCE obtain error messages when awarding tasks, SCE can recognize that contract decomminting occurs. SCE has to be able to perform its task by performing the scenario (see Figure 7). Steps in the scenario are as follows:
    1. SCE awards tasks to web services.
    2. A web service returns an error to the SCE if it fails to provide service.
    3. SCE requests SME for decommiting.
    4. SME returns the agreement to SCE if it agrees to decommit the contract.
    5. If there are still available contractor services, perform Awarding Task scenario again (the same as the "Normal Decommiting" Scenario).
    6. If there is no available contractor service, perform Making Contract scenario again (the same as the "Normal Decommiting" Scenario).

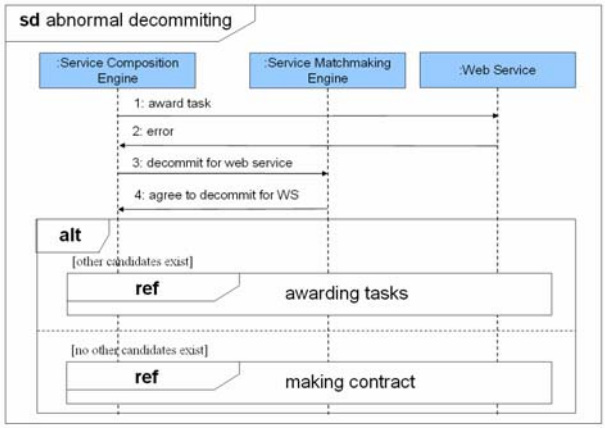

Figure 7: Abnormal Decommiting

3.3: Implementation

On the basis of ECNP, we develop a system that integrates the proposed service composition processes with service matchmaking, and realize the requirements for solving identified issues. The system architecture (see Figure 8) will be described as follows.

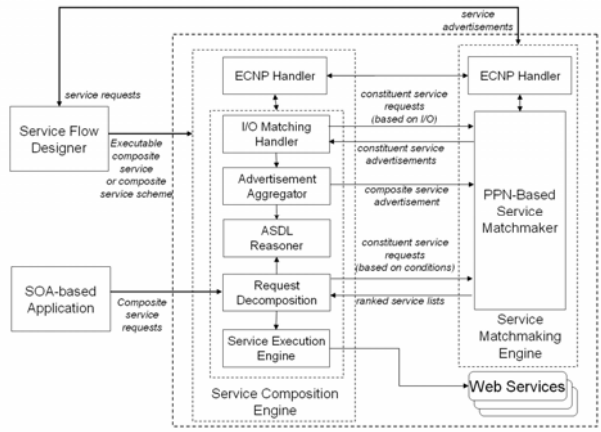

Figure 8: System Architecture of Discovery-Based Service Composition

There are two main components in this architecture: the service matchmaking engine (SME) and the service composition engine (SCE), that deal with how to locate suitable services based on service requests, and how to invoke constituent services based on the composition plan (service composition schema with service bindings) individually. We leverage Possibilistic Petri-Nets-based service matchmaker[9] to realize tasks of service discovery and service selection in SME, and apply standard: BPEL[6] as the service composition language in SCE. Especially, we strengthen these two engines with ECNP-handling capability of signing contracts and performing tasks according to the contracts, and develop some facilities to solve some implementation-level issues, including I/O matching (for matching interface compatible services), conditional service matching (for matching services in a semantic manner), advertise aggregation, request decomposition, etc.. In this architecture, there are two kinds of potential clients: Service Flow Designer and SOA-based Application.

The users of Service Flow Designer are composite service developers. Developers can search services or service descriptions that they want to integrate in the target composition through SME. After the developer completes the design of the composite service, the produced service composition schema can be deployed in SCE. Particularly, SCE will sign contracts with candidate constituent services and bind default concrete service before deployment of abstract composite services.

SOA-based Application plays the roles as the service requestor of simple or composite services. If the application want to invoke an abstract composite service, the service bindings of constituent services will be decided at runtime based on the end user's requirements. We utilize the request form mechanism proposed by [9] to capture users' requirements. Besides, if some constituent service is missing or malfunctions during the delivery of the composite service, SCE will assign other candidate constituent service to take over according to the contract relationship. We believe that both static and dynamic service composition can be realized in this architecture.

4: RELATED WORK

eFlow[2] is a pioneering system that supports the specification, enactment and management of composite services. eFlow uses a static workflow generation method. A composite service in eFlow is modeled by a graph that defines the order of execution among the nodes in the process. eFlow allows service selection at runtime. However, eFlow does not address the service discovery in different phases in the whole lifetime of the composite service.

Self-Serv [1] , is a middleware infrastructure for the composition of Web services. In Self-Serv, web services are declaratively composed and then executed in a dynamic peer-to-peer environment. Self-Serv postpones the decision of which specific service handles a given invocation until the moment of invocation by conducting multiattribute dynamic service selection. However, Self-Serv also does not address the service discovery in different phases of service composition.

The Web Component[15, 16] approach treats services as components in order to support basic software development principles such as reuse, specialization, and extension. The main idea is to encapsulate composite-logic information inside a class definition, which represents a Web component. A Web component's public interface can then be published and used for discovery and reuse. Although service discovery, composability and compatibility checking are involved in the planning stage of this approach, it dose not stress service discovery.

METEOR-S[11] aims to extend the web service standards with Semantic Web technologies to achieve greater dynamism and scalability. MWSCF (METEOR-S Web Service Composition Framework) [13] reduces dynamic composition of Web services to a constraint satisfaction problem, and uses a multi-phase approach for constraint analysis. Besides, MWSCF utilizes Integer Programming to optimize QoS of composite services. METEOR-S is a contemporary multi-phased service composition approach, but it also dose not handling missing or malfunctioning constituent services.

SWORD[10] is a developer toolkit for building composite web services using rule-based plan generation. SWORD does not deploy the emerging service description standards such as WSDL and DAML-S, instead, it uses Entity-Relation (ER) model to specify the Web services. It is a typical AI Planning-Based service composition approach, and not emphasizing the significance of service discovery like other approaches.

5: CONCLUSION

In this paper, we propose a Discovery-Based Service Composition methodology and framework by extending Contract Net Protocol. Our approach supports both static service composition and dynamic service composition, with some key features as follows:

- The composite service can bind the best available services that fit the end users needs through interface and semantic service matching under the umbrella of the ECNP.
- During execution of the composite service, if some constituent service is missing or malfunctions, the composite service will not fail to work because other candidate constituent services will take over according to the contract relationship.

In addition, all activities of service composition, discovery, and invocation can be coordinated by ECNP. ECNP provides a reference model to composing services based on the service composition lifecycle, and resolving some frequently encountering problems, such as service discovery and service dynamics.

REFERENCES

[1] B. Benatallah, Q. Sheng, and M. Dumas. The self-serv environment for web services composition. IEEE Internet Computing, 7(1):40-48, Jan.-Feb. 2003.

[2] F. Casati, S. Ilnicki, L. jie Jin, V. Krishnamoorthy, and M.-C. Shan. Adaptive and dynamic service composition in eflow. In CAiSE '00: Proceedings of the 12th International Conference on Advanced Information Systems Engineering, pages 13-31, London, UK, 2000. Springer-Verlag.

[3] H. Cervantes and R. S. Hall. Automating service dependency management in a service-oriented component model. In Proceedings of the 6th ICSE Workshop on Component-Based Software Engineering: Automated Reasoning and Prediction, 2003.

[4] H. Cervantes and R. S. Hall. Chapter I: Service Oriented Concepts and Technologies in the book Service-Oriented Software System Engineering: Challenges and Practices. Idea Group Publishing, 2005.

[5] A. Charfi and M. Mezini. Aspect-oriented web service composition with ao4bpel. In Proceedings of ECOWS 2004, 2004.

[6] F. Curbera, Y. Goland, and et al. Business process execution language for web service (bpel4ws) 1.0. Technical report, IBM Corp. and Microsoft Corp., 2002.

[7] S. Dustdar and W. Schreiner. A survey on web services composition. Technical report, Distributed Systems Group, Technical University of Vienna, 2004.

[8] O. M. Group. Unified modeling language: Superstructure, version 2.0. Technical report, August, 2005.

[9] J. Lee, Y. C. Wang, C. L. Wu, S. J. Lee, S. P. Ma, and W. Y. Deng. A possibilistic petri-nets-based service matchmaker for multi-agent system. International Journal Fuzzy Systems, 7(4), 2005.

[10] S. R. Ponnekanti and A. Fox. Sword: A developer toolkit for web service composition. In Proceedings of the 11th World Wide Web Conference, 2002.

[11] P. Rajasekaran, J. Miller, K. Verma, and A. Sheth. Enhancing web services description and discovery to facilitate composition. In Proceedings of SWSWPC 2004: International Workshop on Semantic Web Services and Web Process Composition, 2004.

[12] T. Sandholm and V. Lesser. Issues in automated negotiation and electronic commerce: Extending the contract net framework. In Proceedings of the First International Conference on Multiagent Systems (ICMA-95), 1995.

[13] K. Sivashanmugam, J. A. Miller, A. P. Sheth, and K. Verma. Framework for semantic web process composition. International Journal of Electronic Commerce, 9(2):71-106, 2005.

[14] R. G. Smith. The contract net protocol: high level communication and control in a distributed problem solver. IEEE Transaction on Computers, 29(12):1104-1113, December 1980.

[15] J. Yang and M. Papazoglou. Web component: A substrate for web service reuse and composition. In Proc. 14th Conf. Advanced Information Systems Eng. (CAiSE 02), page 21V36, 2002.

[16] J. Yang and M. Papazoglou. Service components for managing the life-cycle of service compositions. Information Systems, 29(2):97-125, 2004.

[17] J. Yang, M. Papazoglou, and W.-J. V. Heuvel. Tackling the challanges of service composition. In Proceedings of ICDE-RIDE Workshop on Engineering E-Commerce/E-Business, 2002.