On Developing Service-Oriented Web Applications

Sabah S. Al-Fedaghi
Computer Engineering Department
Kuwait University
sabah@alfedaghi.com

Abstract

One approach to developing service-oriented Web applications is to transform high-level business models to a composition language that implements business processes with Web services. Object-oriented analysis and design and UML-based diagrams are typically used in the software development process. In this paper, we discuss the development of business processes through introduction of a conceptual model as a foundation for designing "Web services" based on information flow. We first scrutinize current modeling used in transformation methodologies, then introduce a flow-based conceptual description of services through a case study with a high-level business description.

1 Introduction

Service-Oriented Computing uses "services as fundamental elements" [Papazoglou *et al.*, 2009 (access date); Papazoglou *et al.*, 2003].

> Service-Oriented Computing … utilizes services as the constructs to support the development of rapid, low-cost and easy composition of distributed applications. Services are autonomous, platform-independent computational entities that can be used in a platform independent way… Any piece of code and any application component deployed on a system can be reused and transformed into a network-available service… Services are most often built in a way that is independent of the context in which they are used. This means that the service provider and the consumers are loosely coupled [EclipseCon, 2009].

Web Services are a technology based on the concept of service oriented computing. Web services are standards that integrate Web-based applications through connecting and sharing business processes across the network where applications of different vendors, languages, and platforms communicate with each other and with clients. Thus applications involve assembling components of Web services. The technology provides tools for developing and implementing business processes, e.g., Web Services Description Language (WSDL) and Business Process Execution Language (BPEL).

> Services hold the promise of moving beyond the simple exchange of information … to the concept of accessing, programming and integrating *application services* … The end result is that it is then easier to create new composite applications that use pieces of application logic and/or data that reside in the existing systems… This represents a fundamental change to the socio-economic fabric of the software developer community that improves the effectiveness and productivity in *software development* … [Papazoglou *et al.*, 2009] (reference to [Leymann, 2003]) [Italics added].

Software development methodologies play an important role in this technology. According to Papazoglou and colleagues [2009]:

> Object-oriented and component based development… cannot be blindly applied to [service-oriented architecture] and Web services as they do not address the key elements of [service-oriented architecture]: services, *flows of information* and components realizing services ... One of the main challenges in the development of Services Oriented systems is the provision of methodologies that support the specification and design of compositions of services. [Italics added]

In this paper we propose that the flow model reviewed in section 4 is a promising approach to software development in service-oriented applications. Flow-based conceptual models can reflect high-level design components of Web service and e-business solutions produced early in the application development lifecycle. The models can be utilized by business managers and analysts to trace transformations to models used by software developers. They also provide a means of communication to promote collaboration and standardization.

2 A high level business model: an example

Web applications require a comprehensive approach that embraces many aspects, including technical, organizational, and legal/philosophical dimensions. Hence, information processing methods, techniques, and tools have been extended to support developing applications of this kind, e.g., Object Oriented Web Solutions. Conceptual modeling methods have been utilized to abstractly describe requirements for software development processes for the Web. For example, use cases and scenarios are applied to model functional requirements.

De Castro *et al.* [2007] defined a method for development of service-oriented Web applications that starts "from a high level business model … that simplif[ies] the mapping to a specific web service technology." The method utilizes "extended use cases"' to arrive at the Service Process Model. EclipseCon [2009] illustrates the methodology in the following example.

Example: A conference management system is required that includes three different services: submit an article, display submitted articles, and edit the data of an author, in addition to log-in or registration services. Figure 1 shows a partial "use case overview" for this conference management system given by EclipseCon [2009]. It contains <<include>>, which indicates that the behavior of the included use case is inserted into the one including it, and <<extend>>, which specifies how and when the behavior defined in the extended use case can be inserted into that behavior. The Service Process Model was then developed. "Each complex service identified in the previous model is mapped to an activity, while the basic services that it uses are represented as service activities" [Vara Mesa, 2009].

According to Vara Mesa [2009], a business service is defined "as a complex functionality offered by the system, which satisfies a specific need of the consumer." The conference management system shown in Figure 1 includes the services: "Submit article," "View submitted articles," "Edit author data," Log-in, and Register.

3 Scrutinizing UML-based methodology

We observe that Figure 1 contains conceptually arbitrary services. The author (a person) is directly connected to services "Submit article," "View submitted articles," and "Edit author data." The connection seems to mean that these three services are available to authors. Nevertheless, "availability" is not a notion that should guide the classification of services at this level. The modeling should be guided by a conceptual hierarchy of relationships of services. "Submit article" and "View submitted articles" are article-related services, while "Edit author data" is related to a different artifact (to be explained later).

The design in Figure 1 lacks criteria for constructing a conceptual picture of services that shows the hierarchy of these services. Our methodology, introduced in the following sections, involves a *flow* of artifacts such as articles and author data. In this case, the first level relationship ought to be between authors and *conference service* and contain *article service* and *author data service*.

Furthermore, the relationships between the three services and "Log-in" appear odd depicted at the same level in the conceptual picture. "Log-in" is a separate service that facilitates "entering" the services area, but in Figure 1 it is modeled like any other service.

The UML Activity diagram of the example also reflects conceptual vagueness. It mixes services (e.g., "Edit author data"), and other procedural activities such as downloading of files. Conceptually, this is analogous to mixing shop services such as "selling vegetables" and "selling medicine" with such activities as "weighing items."

We next introduce a proposed alternative conceptual description.

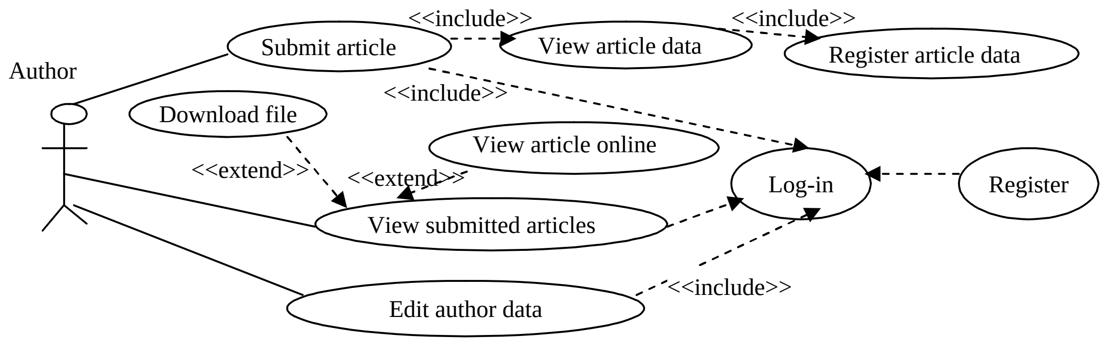
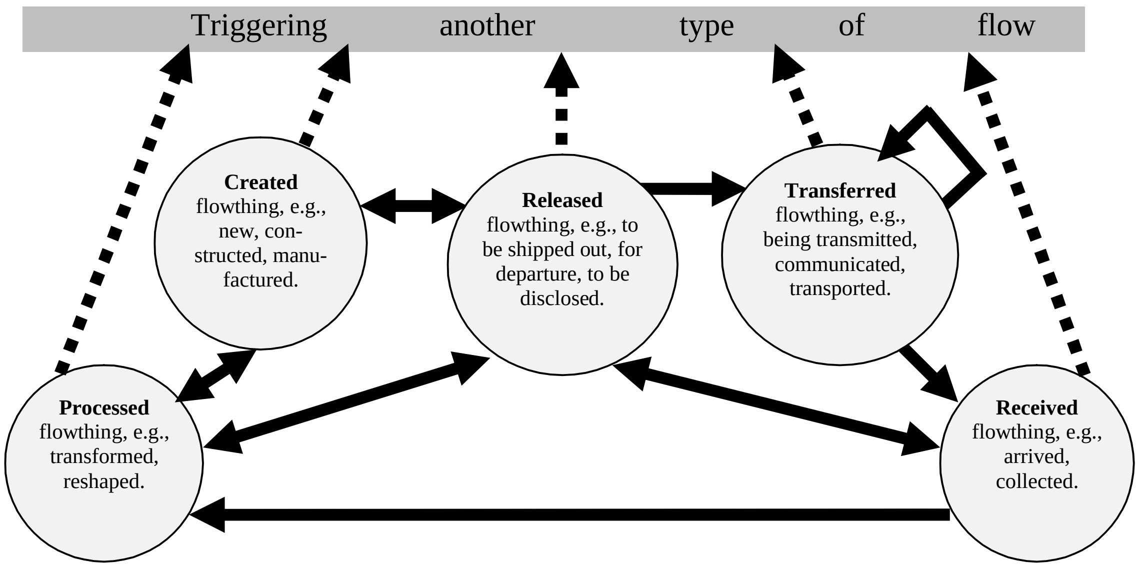


Figure 1. Use Case Overview, partial view from [EclipseCon]



Triggering another type of flow
Created flowthing, e.g., new, constructed, manufactured.
Released flowthing, e.g., to be shipped out, for departure, to be disclosed.
Transferred flowthing, e.g., being transmitted, communicated, transported.
Processed flowthing, e.g., transformed, reshaped.
Received flowthing, e.g., arrived, collected.

Figure 2. State transition diagram for the flow model with possible triggering mechanism.

4 Flow Model

The flow model (FM) has been used in several applications, including software requirements [Al-Fedaghi, 2008a,b,c; Al-Fedaghi, 2009], and privacy in RDF [Al-Fedaghi, 2008d]. This section provides a review of the basic model as it has been described in other publications.

A flow model is a uniform method to represent things that "flow," i.e., things that are exchanged, processed, created, transferred, and communicated. "Things that flow" include information, materials (e.g., manufacturing), money, etc. In economics, "goods" can be viewed as flowthings that are received, processed, manufactured (created), released, and transported. Information is another type of flowthing that is received, processed, created, released, and communicated. The state transition diagram of flowthings is shown in Figure 2. Sample flow systems are shown in Figure 3.

The flowthings in Figure 3(a) are physical persons flowing through an air travel system. In this situation, there is no "creation" stage. If it were an information system of "travelers' records," then the flow of "records of persons" is shown in Figure 3(c). Figure 3(b) shows the flow of transported materials.

Different types of flow can trigger each other. To illustrate, consider the description of interaction between customers and supplier shown in Figure 4. In the context of the flow model, the "orderGoods" arrow can be interpreted as orders flow. The "makePayments" arrow represents two types of flows: invoices and money.

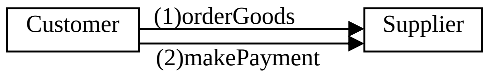
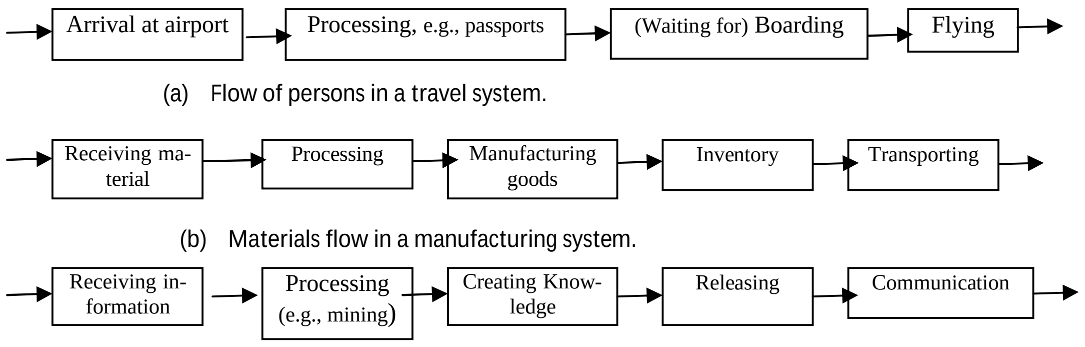


Figure 4. Typical representation.



(a) Flow of persons in a travel system.

(b) Materials flow in a manufacturing system.

(c) Information flow in an information system.

Figure 3. Sample flow systems.

Accordingly, Figure 5 shows the *flow model* description of the interaction. The customer creates an order that flows to the supplier, triggering flow of an invoice. The flow of an invoice to the customer triggers a flow of money. Upon receiving an invoice, the customer creates money (e.g., a money order) that flows to and is received by the supplier.

In this scenario, flows are coordinated, but do not "mix" with each other. It is a common sense approach, comparable to designing a house, where electricity, water, and gas flows appear on the blueprint as separate systems. Electricity may trigger the flow of gas; however, each flow is specified separately. Even in a diagram that includes electrical lines and gas pipes, they are represented differently (e.g., different types of arrows).

5 Using the Flow Model to Develop Service-Oriented Web Services

The flow model is unique in the sense that it can be used to capture core "information flows" and their relationships, in contrast to other conceptual models based on information entities/relationships such as the Entity-Relationship Model and UML-based conceptual model.

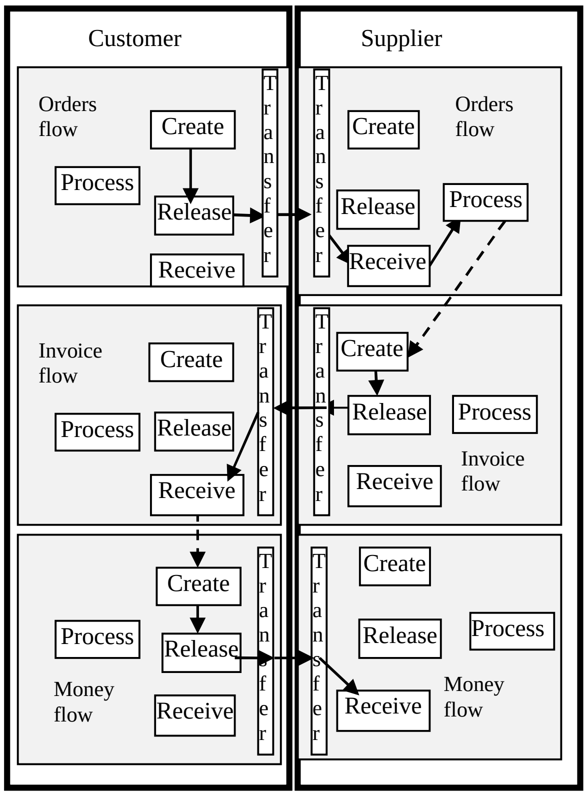


Figure 5. Flow model description of a sample customer / supplier transaction.

The fundamental concept of entities/relationships models is centered on entities and relationships: entity types, properties, entity instances participating in relationships, entity sets, instances of relationship types, relationship sets, etc. The flow model uses *flow of flowthings* as a fundamental notion. "Flows" are first identified, even before specifying the structure of flowthings and their relationship. This seems to be a common sense approach, just as in solving a problem, an agent examines the "streams" that lead to the solution. This is most clear in certain applications:

- Before deciding who is traveling, a travel agent searches for a route from city of departure to city of arrival.

- Before identifying product or customer details, a Web sales agent determines ways of delivery, e.g., only within North America.

- Before targeting an enemy post, a pilot examines the flight route.

Entities/relationships-based models obstruct early phases of conceptualizing this stream. The mere concept of an entity is usually manifested in different forms in preparation for mapping, representation, and binding: objects amenable to implementing, transformation into flows (e.g., XML streams), conversion into in-memory structures (e.g., lists), etc. Object-oriented methodologies have introduced an additional abstraction layer for entities. Still, entities precede, conceptually, the flow of flowthings. In this methodology, solving the problem of traveling from Chicago to Lhasa, in Tibet, starts with the details of travelers and their relationships before the path between the two cities is checked.

The flow model utilizes "flow" as a fundamental concept for solving problems. Many flow mechanisms exist: flowcharts, workflow, data flow, cash circular flow, flow diagram, process flow diagrams. Ordinary programming flowcharts depict fixed patterns of flow of control (sequential, irritation, etc.). Flow is a basic concept in modeling systems, including Web services applications; however, it has never been singled out, explicitly, as a foundation of modeling.

In the next section, we discuss the advantages of utilizing flow-based methodology in a case involving a high-level business model that simplifies mapping to a specific Web services technology. Identifying flows provides an abstraction layer of architecture for logical (e.g., messaging) and physical enterprise (e.g., services) bus.

For example, "orders" are logical flowthings that are created, received, processed, released, and communicated. The flow model description supplies transformation and routing of these orders. Business processes involve a flow of orders that includes purchasing, and management and inventory are modeled in one coherent picture of flows that trigger each other. The purchasing process may involve several flowthings, including information. The inventory process may involve such flowthings as information and actual materials.

In such a scenario, orders are realized "physically" (in the sense of computer jargon) as messages (exchange of information between processes) that can be built upon the same description. Figure 6 shows this correspondence between the logical (description of processes in reality) and physical descriptions (implementation in information system) of this example. To save space, the figure is limited to the inventory process, where it is assumed that some type of assembly is performed before the product is delivered (e.g., toys). The upper view in Figure 6 includes two flows: orders and products. The lower view represents the corresponding conceptual model used early in the application development lifecycle. In this view, instead of products, the flowthing is information that flows in the information system (IS) and is actualized as a software shadow of the (real) flow of materials. An order triggers:

Assembling: actualized in IS by signals/information to the assembler to start assembling products.

Releasing: actualized in IS by receiving a "finished" signal from assembler.

Transporting: actualized in IS by printing address, informing management, etc.

Thus, the lower view in Figure 6 is expanded to implement control, monitoring, and tracking in the logical process of inventory upon the arrival of new orders.

6 Reconsidering the conference management system

Consider again the development of a conference management system discussed in section 2. Viewing it from the perspective of flowthings, four types of services can be identified:

Manuscript service: This service provides uploading, viewing, and editing of manuscripts.

Manuscript record service: This service allows creation of a manuscript record (information about a manuscript such as title, number of words, etc.) and viewing and editing of manuscript records.

Author record service: This service allows creation of an author record (information about author, such as name, affiliation, etc.) and viewing and editing of authors' records.

Service request service: This service allows selection of an available service.

Services are identified by their flowthings: manuscripts, manuscript records, author records, and service requests. This contrasts with the approach described in section 2, where flowthings are fragmented and services are disorganized hierarchically and mixed with other activities.

In the flow model, the "object" (flowthing) flow forms a nucleus of classification and hierarchy of that classification. This is analogous to common scene conceptualization such as, for example, a pharmacy providing services related to *medicine* flow, and a factory providing services related to its *products* flow. Similarly, a "*manuscript* service" in our scheme is a category that appears before descriptions of different types of processes included in this service.

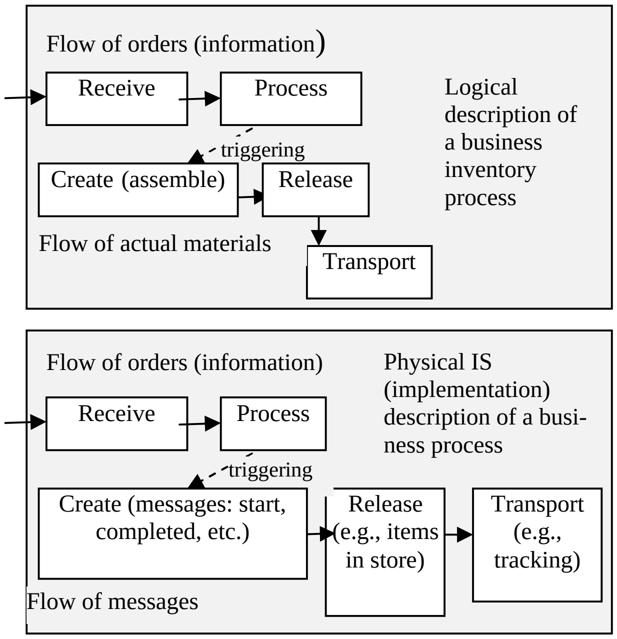

Flow of orders (information)
Receive
Process
Logical description of a business inventory process
triggering
Create (assemble)
Release
Flow of actual materials
Transport
Flow of orders (information)
Physical IS (implementation) description of a business process
Receive
Process
triggering
Create (messages: start, completed, etc.)
Release (e.g., items in store)
Transport (e.g., tracking)
Flow of messages

Figure 6. Illustration of the correspondence between logical description and its shadow initial description used in application development.

Identifying flowthings leads to identifying services, and it also leads to specifications of flows. Each type of service is a realization of flows starting from the flow of requests for that service. A "new manuscript service" request can be mapped as shown in Figure 7, leading to the following actions:

Receiving, and **processing** a *request*,

Triggering a *manuscript* service that includes

Receiving a *manuscript*,

Triggering a *manuscript* record service that includes

Creating a *manuscript* record.

A service is a flow that receives, processes, creates, releases, and communicates flowthings. Flowthings change their states according to the stages of the flow model. This is in line with Hill's [1977] definition of services as "a change in condition or state of an economic entity (or thing) caused by another."

A conceptual description of this example is given in Figure 8. The figure includes four flows. It can be used to specify procedures such as the one described above for "new manuscript service." The numbers in circles identify triggering points. For example, upon receiving a service request:

If the service request is for new manuscript then (1) is activated, else (2) is activated.

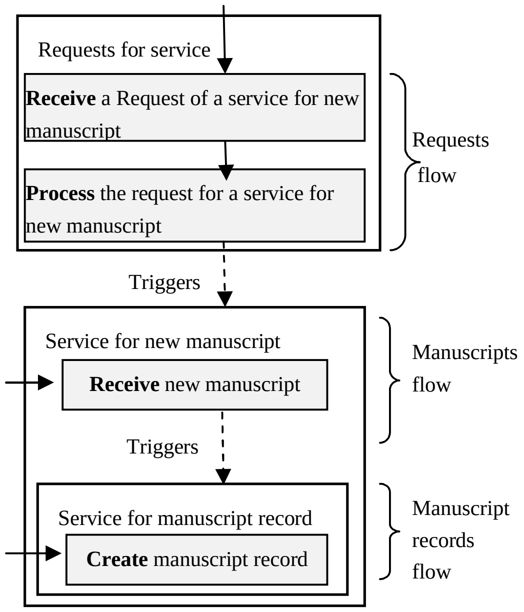

Requests for service
Receive a Request of a service for new manuscript
Process the request for a service for new manuscript
Requests flow
Triggers
Service for new manuscript
Receive new manuscript
Triggers
Manuscripts flow
Service for manuscript record
Create manuscript record
Manuscript records flow

Figure 7. A service request is a realization of flows.

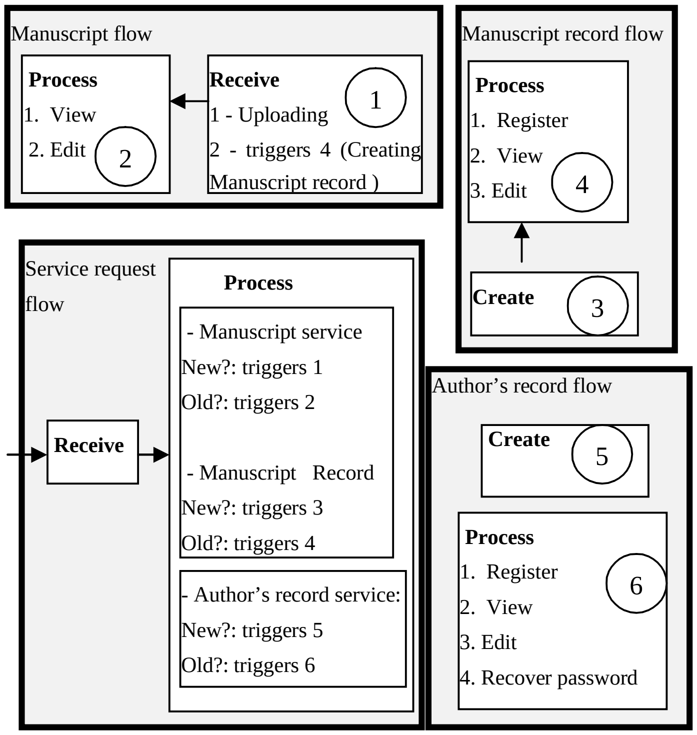


Figure 8. Flow-based conceptual description of the of a conference management system

References

[Al-Fedaghi, 2009] S. Al-Fedaghi. Flow-based description of conceptual and design levels. *IEEE International Conference on Computer Engineering and Technology 2009*, January 22–24, 2009. Singapore.

[Al-Fedaghi, 2008a] S. Al-Fedaghi. Scrutinizing UML activity diagrams. *17th International Conference on Information Systems Development (ISD2008)*, Paphos, Cyprus, August 25-27, 2008.

[Al-Fedaghi, 2008b] S. Al-Fedaghi. Software engineering interpretation of information processing regulations. *IEEE 32nd Annual International Computer Software and Applications Conference* (IEEE COMPSAC 2008), Turku, Finland, July 28–August 1, 2008.

[Al-Fedaghi, 2008c] S. Al-Fedaghi. Systems of things that flow. *52nd Annual Meeting of the International Society for Systems Sciences* (ISSS 2008), University of Wisconsin, Madison, USA, July 13–18, 2008.

[Al-Fedaghi, 2008d] S. Al-Fedaghi and A. Ashkanani. Privacy–based RDF. *International Journal of Metadata, Semantics and Ontologies*, 3(2), 2008.

[De Castro *et al.*, 2007] V. De Castro, J. M. Vara, and E. Marcos. Model transformation for service-oriented Web applications development. *Workshop Proceedings of 7th International Conference on Web Engineering*, pages 184-198, July 2007.

[EclipseCon, 2009 (access date)] EclipseCon. Modeling Web applications: Detailed description http://www.eclipse.org/m2m/atl/usecases/webapp.modeling/description.php#using.model.annotation

[Hill, 1977] T. P. Hill. On goods and services. *Review of Income and Wealth*, 23(4): 315–338, 1977.

[IBM, 2009 (access date)] IBM. Managing information technology services, http://www935.ibm.com/services/us/its/pdf/managing_it_services_white_paper.pdf

[Knowledgerush, 2009 (access date)] Knowledgerush. http://knowledgerush.com/kr/encyclopedia/Service/

[Leymann, 2003] F. Leymann. Web services: Distributed applications without limits. In *Proc. BTW'03* (Leipzig, Germany, February 26–28, 2003), *Lecture Notes in Informatics*, vol. P-26, *Gesellschaft fuer Informatik* (GI), Bonn, Germany, 2003.

[Papazoglou *et al.*, 2003] M. P. Papazoglou and D. Georgakopoulos. serviced-oriented computing. *Communications of ACM*, 46(10): 25–28, 2003.

[Papazoglou *et al.*, 2009 (access date)] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing. Research roadmap. ftp://ftp.cordis.europa.eu/pub/ist/docs/directorate_d/st-ds/services-research-roadmap_en.pdf

[Vara Mesa, 2009 (access date)] J. M. Vara Mesa. ATL/AMW use case modeling eb applications: Detailed Description and User Guide http://www.eclipse.org/m2m/atl/usecases/webapp.modeling/resources/User.Guide.pdf

[Verner, 2004] L. Verner. BPM: The promise and the challenge. *Queue of ACM*, 2(4): 82-91, 2004.