On Web Services Aggregation

Rania Khalaf[1] and Frank Leymann[2]

[1] IBM TJ Watson Research Center, Hawthorne, NY 10532, USA
rkhalaf@us.ibm.com
[2] IBM Software Group, Boeblingen, Germany
ley1@de.ibm.com

Abstract. The Web services framework is enabling applications from different providers to be offered as services that can be used and composed in a loosely-coupled manner. Subsequently, the aggregation of services to form composite applications and maximize reuse is key. While choreography has received the most attention, services often need to be aggregated in a much less constrained manner. As a number of different mechanisms emerge to create these aggregations, their relation to each other and to prior work is useful when deciding how to create an aggregation, as well as in extending the models themselves and proposing new ones. In this paper, we discuss Web services aggregation by presenting a first-step classification based on the approaches taken by the different proposed aggregation techniques. Finally, a number of models are presented that are created from combinations of the above.

Keywords. business process modeling, composition, aggregation, web services.

1 Introduction

Web services [10,20] offer an XML-based framework that embodies the concepts of the Service–Oriented Computing (SoC) paradigm, created as a result of the movement away from the tight integration previously required for distributed IT offerings that cross enterprise boundaries. In the SoC model, applications from different providers are offered as services that can be used, composed, and coordinated in a loosely-coupled manner.

The Web services framework consists of an extensible, modular stack of open XML-standards that enable an application to expose its functionality in a machine–readable, implementation–neutral description such that it may be discovered, bound to, and interacted with possibly over a number of different protocols regardless of its location in the network. This environment is therefore intrinsically heterogeneous, distributed, and dynamic.

With a viable service model in place, the aggregation of services to provide composite applications and maximize reuse becomes key. For example, service aggregators may reuse services that have already been created, or offer new services formed by choreographing interactions with available services offered by other providers. Although Web services overlay on top of existing IT technologies, the Web services environment is more dynamic and loosely coupled.

B. Benatallah and M.-C. Shan (Eds.): TES 2003, LNCS 2819, pp. 1–13, 2003.

In comparing agents and components, [18] notes that agents concentrate on dynamicity and components concentrate on composability. Web services need both, making their aggregation models of particular interest.

Choreography-based composition has taken the forefront when considering aggregation in Web services. However, other less-structured aggregation models exist. As multiple languages and mechanisms are proposed, understanding their relation to each other and to prior composition and aggregation models is useful when deciding how to create an aggregation, as well as in extending the models themselves and proposing new ones.

In this paper, we study prevalent aggregation mechanisms by creating a classification that groups them based on their approach and applicability. The aim of this classification is two-fold. First, it distills the space by grouping aggregation mechanisms based on their design goal, allowing one to step back from the plethora of acronyms, specifications, and systems. Second, it begins identifying primitive aggregation techniques, allowing one to reason about useful combinations.

We present each of the approaches in the classification and discuss future work. Finally, a number of models created from combinations of the above are presented.

2 Background: Defining Web Services

Before discussing aggregation, we provide an overview of the functional description of a Web service, usually provided by the Web Services Description Language (WSDL)[8]. WSDL embodies the Web services principle separating the abstract functionality of a service from its mappings to deployed implementations, thereby enabling an abstract component to be implemented by multiple code artifacts and deployed using different communication protocols and programming models.

In WSDL, the abstract part consists of one or more "portTypes", constituting the service's interface. PortTypes specify supported operations and their input and/or output message structures. These may then be used by third parties aggregating compo-nents or by client code invoking operations.

The concrete part of a WSDL definition consists of bindings, ports, and services. A binding is a mapping of a portType to an available transport protocol and data encoding format. A port provides the location of a physical endpoint that implements a portType using one of the available bindings. A service is a collection of ports.

The aggregation mechanisms covered in the rest of this paper, in particular those performing type–based aggregation, will make extensive use of the WSDL definitions of their constituent services.

3 A Classification of Web Services Aggregation

For the focus of this paper, we define Web services aggregation as the combination of a set of Web services to achieve a common goal. An aggregation may be

created both at the interface–level (agnostic to service implementations), or directly at the instance level. The former provides more flexibility in allowing late binding to actual deployed instances. We consider mechanisms whose purpose is the first–class creation of such aggregates.

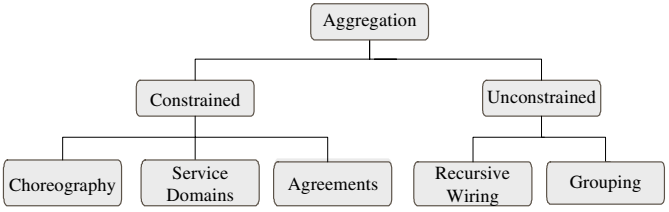

Fig. 1. Classification of Web Services Aggregation Models

The classification driving the discussion is illustrated in figure 1, and discussed in detail throughout the paper. The initial split depends on whether or not the aggregation is constrained. Unconstrained aggregation does not impose any control on the interactions between the services used or on the services that may be part of the aggregation. Approaches of this kind include simply grouping a set of services in one "bag" or defining an architectural setup through the use of join points and connectors.

On the other hand, constrained aggregation imposes constraints on the aggregated services, specifically regarding the functionality they must support and/or the interactions they may have with each other and with the aggregate itself. Three sub-categories are presented: choreography, service domains, and agreements.

This classification may be extended at each of the current leaves, and/or by adding higher level branches if significantly different models are proposed. For example, the choreography category can be further branched down based on the process meta–model used, including "calculus-based" [28], "state-chart based" [6],"graph-based" [16], and so on.

3.1 Unconstrained Aggregation

Grouping. Grouping simply provides a collection of services. Two cases are presented here, grouping of interfaces and grouping of instances.

– Interface Inheritance
  Multiple interface inheritance has been proposed for WSDL 1.2 in the form of portType inheritance. It introduces substitutability semantics at the instance level: An instance of a subtype in the inheritance hierarchy can be used wherever an instance of one of its supertypes can be used. Copy semantics apply at the specifications level as shown in figure 2, such that those of the supertypes are copied to the subtype. The implementation of the subtype provides a single endpoint with a single binding at which implementations

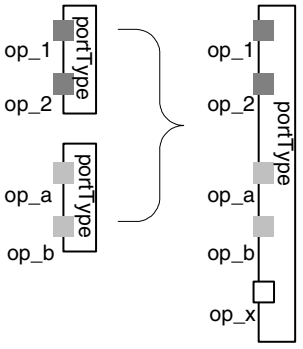

Fig. 2. Aggregating interfaces using portType inheritance

of all inherited operations are made available. This is different from service domains below.

While the area of multiple interface inheritance is relatively well understood due to prior experience in OOP, two issues have to be addressed that complicate the scenario for Web services: the lack of control over the portTypes being extended (they may be owned by a third-party) and the introduction of policies. The first leads to the problem of method signature clashes from methods coming from two different, inherited portTypes. Normally, one may be able to redefine or rename one of them, but those definitions may be out of the control of the user extending them. The second complication is the flexibility of the Web services quality of service framework, WS-Policy [3, 2]. Policies defining quality of service capabilities or requirements may be attached not only to the concrete part of the WSDL (part of the binding, or the port), but also to portTypes or operations within them. It is not yet clear what this would mean to multiple portType inheritance. The problems that could occur become clearer if one considers the following two cases: inheriting two operations with the same signature and different policies; inheriting a set of operations that may have conflicting policy requirements.

– Instance Grouping

An unstructured grouping of instances may be created to offer a number of related service instances together to a user. One example of its use is the creation of an aggregate as a collection of WSRP [15] portlets. Basically, a number of portlets of arbitrary functionality may be added to a portal that provides a unified access point to the user.

The WSRP specification enables portlets to be grouped into a portal, allowing a user to interact with them from a single location such as a Web page. While this approach is user-interface centric, it is one of the few examples of a pure grouping of instances. The container mediates the interaction between the user and the embedded services, but the portlets themselves are not connected to each other, and at the time of this writing, there was no way in WSRP to specify piping the output from one into the input of another as is done in “recursive wiring” below.

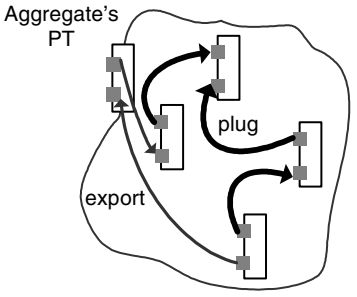

Fig. 3. Aggregation using Recursive Wiring

Recursive Wiring. One form of aggregation is to wire services together such that they may use each other's services. The aggregation created in this way may then itself be exposed as a service which can be wired and invoked. A simple example is a service composed of a chain of filters, where each filter is a Web service. Wiring occurs by connecting the output of the first filter to the input of the next and so on. The input of the first filter and the output of the last one may then be exposed on the aggregate.

This form of aggregation has been commonly used in component-based software engineering where component definitions include pluggable "ports", each port referring to an interface. Components are wired by binding input ports to output ports. The creation of compound components is done by binding an input port on an internal component to an input port on the created compound one, and similarly for output ports [23,25]. The wiring in these approaches is implicit, defined as part of the component definitions themselves. In Java, a similar approach is the use of event sources (output) and event listeners(input), with wiring occurring by adding one Bean as a listener to events from another. [13] presents a scripting language that makes the wiring first class. In [14], the focus is on making the connectors themselves first class, with an architecture created by combining components with connectors. Connectors are of different types, which specify the behavior of the interaction between the services they connect.

In Web services, WSFL's Global Models [16] define a variation on the recursive combination of components using explicitly defined connectors. In Global Models, directed "plug links" connect an output operation, op_{out} , from a portType of one service to the input operation, op_{in} , from a portType of another service. Message manipulations to match message types are enabled within the plug link definitions. "Dangling" operations may be exported to the boundary of the aggregation where they can be arranged in portTypes representing the interface of the aggregate, as illustrated in Figure 3.

WSFL's global model concept results in a recursive aggregation model at the interface (portType) level. External usage of operations of these portTypes will be delegated to the exporting operations. It can be perceived that the portTypes at the boundary of the aggregate are implemented by the corresponding exporting operations, i.e. the aggregation mechanism specifies an implementation of the

operations via delegation. This exporting of operations enables recursive composition, similar to the creation of the compound components described earlier and to aggregation in COM [26].

The key differentiators between this model and those in the cited literature are that the wiring in Global Models is at the operation level, it does not require an exact mirror of the wired operations due to the ability to define message transformations, and most importantly it does not refer to instances of services but builds off of their abstract descriptions. The plug links may define a "locator", however, that defines how an instance may be located at runtime (for example, through a registry lookup).

3.2 Constrained Aggregation

Choreography. The wiring of services described above is based on the services themselves driving an instance of the aggregate. This is not sufficient for more complex cases such as those required in executing business functionality. In such situations, one needs to model a business process by choreographing the interactions with the services in the aggregate. The workflow and business process modeling mechanisms used in Web services fall under this category. Choreographies are constrained, proactive compositions that drive the interactions with the services they aggregate by defining execution semantics on a set of activities that perform these interactions. These semantics may be defined in a number of ways, such as connecting them with directed edges to form a DAG [21,16, 7], nesting them in compound activities with execution semantics [28], or using Petri Nets [17].

Web services choreography languages are based on the interfaces of the services, decoupled from specific service instances which may therefore be found and bound to during either the design, deployment [9], or execution phases of an aggregation. In [31,20], selection during execution enables the aggregate to meet (global) quality of service constraints. There are two main reasons for this: the dynamic nature of Service-Oriented Computing in which services and their requirements may change frequently, and the need to reuse business process logic across different implementations.

To illustrate, we describe one of the choreography mechanisms in more detail: The Business Process Execution Language for Web Services (BPEL4WS). The flow of control in BPEL4WS is defined using a combination of the aforementioned graph and calculus based approaches. A predefined set of simple activities is provided to: 1)perform Web services interactions: invoke operations on Web services, receive and reply to invocations on the process itself from the services it composes, and 2)provide additional functionality such as waiting, throwing faults, and manipulating data associated with the process. Control is defined using both (or either) directed control links and compound activities that impose, on the activities that are contained within them, semantics such as parallelism, sequencing, or non-deterministic choice. BPEL also defines compensation and fault handling mechanisms, [11].

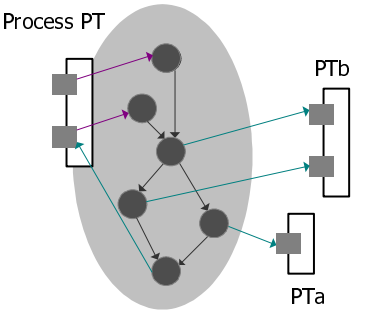

Fig. 4. Aggregating interfaces using choreography

BPEL4WS compositions are based purely on the interfaces of the composed services, that is, their abstract definitions in WSDL (portTypes, operations, and message types). From an architectural perspective, the resulting process is itself a Web service that can be exposed using WSDL definitions that define the portTypes it provides to clients. Additionally, the BPEL process also declares which portTypes it requires from each of the services it composes.

Mechanisms for binding service instances to a process are intentionally left up to the runtime and therefore out of the process definition. Dynamic binding is enabled through the definition of an activity that, when activated, copies a service reference from a received message (possibly from an invocation to a UDDI registry lookup or from a known service provider) into the runtime.

The "executable variant" of this semantics results in proactively driving the services as shown in figure 4. The "abstract interface variant" defines constraints in using the corresponding operations. The latter includes the degenerated case of a single portType, allowing one to specify ordering constraints in using the operations of that port type. Similarly, message exchange patterns (MEP) [1] between sets of portTypes can be defined via choreography.

In related work, semantic information is being used to enable the automatic creation of choreographies. For example, [24] presents a backwards chaining planner to automatically compose a set of Semantic Web services by working from a given goal.

Service Domains. A Service Domain is an aggregation formed by a set of implementations that complement (or compete) with each other to collectively implement a collection of portTypes. The service instances that may become part of this aggregate are constrained by the set of portTypes the aggregate includes, as illustrated in figure 5. The sum of the instances provides the implementation of the interfaces in the domain, with the domain dispatching each incoming call to the appropriate instance that can execute it. At the type level, a service domain SD is a set of portTypes, i.e. $SD = \{pT^1, ..., pT^n\}$. At the instance level a service domain sd is a collection of ports, i.e. $sd = \{p_1^1, ..., p_{k(1)}^1, p_1^n, ..., p_{k(n)}^n\}$. Here, each port $p_i^j \in sd$ is of portType $pT^j \in SD$.

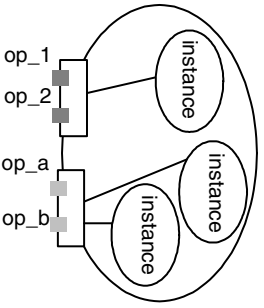

Fig. 5. Service Domains

The ports of a service domain *sd* may be provided by different service providers. An instance of a service domain is created by attaching ports for the aggregated portTypes to the instance. Different service providers may contribute ports for the same portType, and a given service provider may contribute ports of only a subset of the port types of the service domain.

A requestor is blind to the services constituting the aggregate, and picks a portType that he needs to interact with an instance of. A "hub" managing the instance of the service domain then selects a port of this portType from those made available to it by the different providers, and dispatches the call made by the requestor to it.

The next level of refinement of a service domain is the introduction of service level agreements (SLA) [22] to be used in the dispatching: Each portType of a service domain is associated with a set of SLAs. A provider that wants to contribute a port of a portType of the service domain registers with one of these SLAs, committing to the corresponding service level.

A requestor specifies a SLA with the service level he expects in his interactions with the ports of the instance of the service domain. The hub managing the instance of the service domain performs the choice underlying its dispatch decision based on matching SLAs and optimizing overall resource utilization. The owner of the hub specifies the rules to influence these choices.

This is similar to the aggregation model used in [27] and the *service communities* in [6], in which the service communities themselves are exposed as Web services.

Agreements. Agreement based aggregation is much more loosely constrained than choreographed aggregation. It is the creation of a distributed activity by the temporary grouping of a set of service instances, at the end of which a joint outcome is reached and possibly disseminated. This joint outcome is reached based on the members of the aggregation following a pre-defined set of protocols. Note that the portTypes of the services involved are not defined as part of this aggregation mechanism; instead, it relies purely on the ports (service endpoints) themselves.

One example is that of the sealed bid auction presented in [20]. Buyers and sellers are Web services following a certain coordination protocol and being co-

ordinated by an auctioneer. At the end of the bidding period, the auctioneer decides who the winner of the bid is and informs both buyer and seller and the temporary collection of services ceases to exist.

The Web Services Coordination specification (WS–C for short)[4] is a realization of this approach. A distributed activity is a unit of computation that involves a set of services and at the end of which these services jointly agree on an outcome based on a coordination protocol. A coordination protocol consists of a set of messages and a specification of how these messages are to be exchanged. Coordination types then group a set of coordination protocols needed between the different services of a distributed activity. Additionally, methods are defined for registering participating services with coordinators. WS-C provides a pluggable framework for this form of aggregation, such that different and new coordination protocols may be used. The coordination protocols are defined separately, for example the WS-Transactions specification (WS–Tx for short)[5] describes two protocols for long-running transactions and for atomic activities.

WS–C has been considered in conjunction with WS–Tx. The use of this model will depend on the ease of introducing new protocols for different aggregations and additional real–world scenarios.

4 Instance Versus Interface Aggregation

The aggregation models discussed thus far aggregate either on the instances directly, such as is the case with service domains, or on the interfaces (WSDL portTypes) of the aggregated services. In the latter case, the binding to actual instances is done at some later point and is not the main concern of the aggregate's definition.

Table 6 summarizes the aggregation approaches defined above and shows corresponding Web services realization of these approaches. The level of granularity added to this table is the separation of aggregation concerned with instances from those concerned with interfaces.

5 Combined Aggregation Models

The categories defined above constitute different approaches to aggregation; however, useful patterns may be created by combining some of them together. The youth of the Web services technology means that such combinations are still in their very early stages. In this section, we describe a few possible combinations.

- *Choreography with Agreements:* The ability to coordinate multiple Web services extends naturally to coordinating both within and among aggregates. WS-Coordination (WS-C) was released in conjunction with BPEL4WS, along with WS-Transaction (WS-Tx), to be used either separately, or in tandem for more robust composition as described in [12]. The BPEL4WS specification contains an appendix on using WS-C/WS-Tx to coordinate nested scopes within a choreography.

how \ what		Types	Instances
Constrained	Choreography	BPEL4WS, MEPs	
	Service Domains	N/A	Service Domains, Service Communities
	Agreements	N/A	WS-Coordination
Unconstrained	Recursive Wiring	Global Models (WSFL)	
	Grouping	WSDL PortType Inheritance*	WSRP

*proposed, not part of current specification

Fig. 6. Summary of aggregation mechanisms

- *Choreographed Recursive Wiring*: The model of choreography presented in this paper has so far been from the perspective of the choreographing service. However, it is necessary to define global interaction choreographies that multiple partners may plug into. While recursive wiring allows plugging different services together, it does not restrict or define how they may interact - execution semantics are part of the implementations of the services. Outside of Web services, this requirement has been addressed in work on interorganizational workflow. The approaches used in the workflow literature map onto a combination of choreography and wiring: First, one must define the structure identifying the components and the channels of communication. This is similar to Global Models in section 3.1, but recursiveness is not addressed. Second, the literature defines interaction protocols either in terms of Petri Nets [30] or as sets of Message Sequence Charts [29,19].
  The realization of this combination will require further work in Web services, as BPEL4WS defines a protocol that drives an interaction instead of what is described here which is a protocol which defines how the different parties should drive the parts of the interaction that they are participating in.
- *Any Type-Based Aggregation with Service Domains:* Type-based aggregation mechanisms may refer to a service domain, restricting the choice of instances at runtime to the discretion of the domain. One example of this approach is used in [6] through its combination of choreography and service communities.
- *Recursive Wiring with Instance Grouping:* This combination results in instance based recursive composition, wiring together the instances that make up a grouping. For example, a set of portlets may be wired to pipe output from one portlet and into another.

6 Conclusion.

The area of Web services aggregations is seeing a large amount of activity as aggregation mechanisms are still evolving. Some are being extended and new

ones created to enhance their capabilities. As multiple proposals emerge for aggregating Web services, it is important to understand where the mechanisms needed fit in and how they relate to existing approaches.

In this paper, we have discussed Web services aggregation by presenting a first–step classification based on the approaches taken by the different proposed aggregation techniques. The main commonality is that an aggregate is created to group a set of services in order to achieve a common goal. From there, we have shown an aggregation may be either constrained or unconstrained, depending on how much structure is needed to achieve the goal in question and on how independent the behavior of the constituent services may be from the composition itself. A choreography presents a composition-driven aggregation which defines control sequences of interactions with the aggregated services, whereas an aggregate created using recursive wiring does not care about sequencing. Instead it simply connects service interfaces together, so that once bound to instances the service instances drive their interactions along these connections. The ways in which aggregations may be formed is then further refined as shown in figure 1.

As the Web services model itself (WSDL, policy, etc) stabilizes, aggregation will become more central as it is a key capablity made more challenging by the distributed, dynamic, and heterogenuous nature of Web services. With the aim of clarification of aggregation models, and the youth of this technology, we have started with a small set of categories that identify the main areas as they stand today.

Ongoing work will reflect the effects of the evolution of core specifications, including WSDL, as well as the growth and adoption of Web services aggregation techniques. Refining and expanding the classification will consider both adding categories, and additional dimensions for existing categories, such as level and focus of constraints. We are also interested in identifying primitive aggregation mechanisms, and understanding the conditions under which they may or may not be combined

Acknowledgment. The authors acknowledge the contribution of comments on this paper with our colleagues at IBM, in particular, Ravi Konuru, Stefan Tai, and Francisco Curbera.

References

1. Web Services Message Exchange Patterns. Published online by W3C at http://www.w3.org/2002/ws/cg/2/07/meps.html, July 2002.
2. Web Services Policy Attachment (WS-PolicyAttachment). Published online by IBM, BEA, Microsoft, and SAP at http://www-106.ibm.com/developerworks/webservices/library/ws-polatt, 2002.
3. Web Services Policy Framework (WS-Policy Framework). Published online by IBM, BEA, and Microsoft at http://www-106.ibm.com/developerworks/webservices/library/ws-polfram, 2002.
4. WS-Coordination. Published online by IBM, BEA, and Microsoft at http://www-106.ibm.com/developerworks/library/ws-coor, 2002.

5. WS-Transaction. Published online by IBM, BEA, and Microsoft at http://www-106.ibm.com/developerworks/library/ws-transpec, 2002.
6. B. Benatallah, M. Dumas, and Z. Maamar. Definition and execution of composite web services: The self-serv project. *Data Engineering Bulletin*, 25(4), 2002.
7. Fabio Casati, Mehmet Sayal, and Ming-Chien Shan. Developing e-services for composing e-services. In *Proc. of CAiSE2001*, volume 2068 of *LNCS*, pages 171–186. Springer-Verlag, 2001.
8. Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1. Published online by W3C at http://www.w3.org/TR/wsdl, Mar 2001.
9. F. Curbera, M. Duftler, R. Khalaf, N. Mukhi, W. Nagy, and S. Weerawarana. BPWS4J. Published online by IBM at http://www.alphaworks.ibm.com/tech/bpws4j, Aug 2002.
10. Fancisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana. Unraveling the web services web: An introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, 6(2):86–93, 1 2002.
11. Francisco Curbera, Rania Khalaf, Frank Leymann, and Sanjiva Weerawarana. Exception handling in the bpel4ws language. In *International Conference on Business Process Management(BPM2003)*, LNCS, Eindhoven, the Netherlands, June 2003. Springer.
12. Francisco Curbera, Rania Khalaf, Nirmal Mukhi, Stefan Tai, and S. Weerawarana. Web services, the next step: Robust service composition. *Communications of the ACM: Service Oriented Computing*, 10 2003. to appear.
13. Francisco Curbera, Sanjiva Weerawarana, and Matthew J. Duftler. On component composition languages. In *Proc. International Workshop on Component-Oriented Programming*, May 2000.
14. Eric M. Dashofy, Nenad Medvidovic, and Richard N. Taylor. Using off-the-shelf middleware to implement connectors in distributed software architectures. In *Proc. of International Conference on Software Engineering*, pages 3–12, Los Angeles, California, USA, May 1999.
15. A.L. Diaz, P. Fischer, C. Leue, and T. Schaeck. Web Services for Remote Portals (wsrp). Published online by IBM at http://www-106.ibm.com/developerworks/webservices/library/ws-wsrp/, January 2002.
16. Frank Leymann et. al. Web Services Flow Language (WSFL) 1.0. Published online by IBM at http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf, May 2001.
17. Rachid Hamadi and Boualem Benatallah. A petri net-based model for web service composition. In *Proc. of the Australian Database Conference(ADC2003)*, Adelaide, Austaralia, 2003.
18. Denis Jouvin and Salima Hassas. Role delegation as multi-agent oriented dynamic composition. In *NetObjectDays*, 2002.
19. E. Kindler, A. Martens, and W. Reisig. Inter-operability of workflow applications. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *LNCS*, pages 235–253. Springer-Verlag, 2000.
20. Frank Leymann. Web services: Distributed applications without limits - an outline. In *Proceedings, Database Systems for Business, Technology, and Web (BTW)*, LNCS. Springer-Verlag, Feb 2003.

21. Frank Leymann and Dieter Roller. *Production Workflow: Concepts and Techniques*. Prentice Hall, 2000.
22. H. Ludwig, A. Keller, A. Dan, R.P. King, and R. Franck. A service level agreement language for dynamic electronic services. *Journal of Electronic Commerce Research*, 3, mar 2003.
23. Jeff Magee, Andrew Tseng, and Jeff Kramer. Composing distributed objects in CORBA. In *3rd International Symposium on Autonomous Decentralized Systems (ISADS97)*, 1997.
24. Mithun Sheshagiri, Marie desJardins, and Tim Finin. A planner for composing services described in daml-s. In *Conf. on Autonomous Agents and Multi-Agent Systems(AAMAS03), Workshop on Web Services and Agent-based Engineering*, Melbourne, Australia, July 2003.
25. Vugranam C. Sreedhar. Mixin'up components. In *Proc. of the international conference on Software engineering (ICSE2002)*, Orlando, Florida, 2002.
26. Kevin Sullivan, Mark Marchukov, and John Socha. Analysis of a conflict between aggregation and interface negotiation in microsoft's component object model. *IEEE Transaction on Software Engineering*, 25(4), 1999.
27. Y.S. Tan, B. Topol, V. Vellanki, and J. Xing. Manage web services and grid services with service domain technology. Published online by IBM at http://www-106.ibm.com/developerworks/ibm/library/i-servicegrid/, 2002.
28. Satish Thatte. XLANG. Published online by Microsoft at http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm, 2001.
29. W. van der Aalst. Interorganizational workflows: An approach based on message sequence charts and petri nets. *Systems Analysis - Modelling - Simulation*, 34(3):335–367, 1999.
30. W. van der Aalst and M. Weske. The p2p approach to interoganizational workflows. In *Proc. of CAiSE2001*, volume 2608 of *LNCS*, pages 140–156. Springer-Verlag, 2001.
31. Liangzhao Zeng, Boualem Bentallah, Marlon Dumas, Jayant Kalagnanam, and Quang Sheng. Quality driven web services composition. In *Proc. of WWW2003*, Budapest, Hungary, May 2003.