The Influence of Web Services on Software: Potentials and Tasks

Frank Leymann

Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstr. 38
70569 Stuttgart
Frank.Leymann@informatik.uni-stuttgart.de

Abstract: Web services have been introduced about four years ago. Standardization is making a lot of progress as well as corresponding product support by vendors. We sketch some likely high-level impacts of this technology on software in general: On the software representing the environment for executing applications, on the software that represents the application itself, and on the way of using software.

1 Introduction

At its heart Web service technology is about "integration": Integration across platforms and integration across programming artifacts.

Integration across platforms should facilitate creating a new function within a particular application server based on functions existing in other possibly completely different application servers (for simplicity we use the term "application server" a bit generous as a runtime environment for software functions). "Different application servers" means different implementations of the same architecture (e.g. J2EE) by different vendors, or application servers of different architectures and programming models but still of similar "style" (e.g. J2EE and .Net), or completely different styles of runtime environments (e.g. the stored procedure container of a database system and a TP-monitor).

Integration across programming artifacts should facilitate in a uniform manner the usage of or interaction with functions of different kind. For example, this means that the usage model of functions implemented as methods of objects and of functions implemented as procedures should be identical. Likewise, the client of a function should not need to interact differently with that function whether it is running in the same application server as the client or whether the function is running in a completely different application server hosted by a different enterprise.

At the next level of refinement, Web service technology is about virtualization, interoperability, and loose coupling. Virtualization allows a client of a function to view this function as "just" being a Web service, independent of how the function is implemented; the underpinning for this is WSDL. Interoperability allows exchange of messages across platforms based on standardized message architecture, processing model and fault model; this is based on SOAP. Loose coupling allows a client to choose a Web service as late as when actually needed; this is based on policies [WSPF] (and to a certain degree on WSDL binding architecture).

In this manner, Web service technology immediately facilitates creating enterprise application integration (EAI) infrastructures by unifying concepts like adapters [Ke02], connectors [J2EE], wrappers ([BS95], [MMJ01]) etc. via Web services. I.e. Web service technology can be seen as a significant advancement for EAI infrastructures and solutions [HW04]. Having in mind the high percentage of a company's IT budget spend on building EAI solutions, the impact of Web services in this area is obvious.

In the rest of the paper we sketch the influence of Web services on software under three aspects: Section 2 describes the architecture of the software stack supporting the use of Web services, corresponding standards, and the role of Web Services in Grid infrastructures. The structure of software built with Web services as primary artifacts is described in section 3. A Web service oriented model for using software is sketched in section 4.

2 Service Bus

In [L03] we presented a very high-level architecture of a service bus, i.e. the middleware supporting interactions with Web services, which is more detailed in this section. In the meantime, it is not just our expectation but widespread expectation that products implementing a service bus will become mainstream (e.g. [VGH04]). Also, nuclei of a service bus implementation are being worked on in open source projects ([WSIF], [KKL03] for an introduction into WSIF architecture).

2.1 The General Architecture

Figure 1 below depicts the coarse grained architecture of a service bus; the boxes shown represent its major building blocks. The transport layer supports connectivity between Web services and its clients. Although messages in XML format are of high importance, non-XML messages are to be supported by the messaging layer of the bus too, e.g. when directly exchanging Java objects between a Java client and a Web service implemented in Java residing on the same machine. The description layer deals with metadata required to appropriately interact with a Web service, especially information about what functionality is provided (interface), how to access this functionality (binding), and other of its properties as well as properties of the interaction between client and Web service (policy). Directives especially given by policies are realized by the quality of service layer; some key aspects are shown like reliable communication, message- as well as

session level security, and transactions. The components finally providing the proper functionality are residing on top of the stack: Such a component can be atomic (i.e. it is opaque hiding any details about its potential ingredients) or composite. Atomic components may represent stateful resources. Web services that are composites typically expose their state; a composite may be explicitly represented by the predefined specification of its composition (choreography) or the composite is dynamically managed based on documented behaviour (agreement). Orthogonal to that, Web services must be discoverable, and it must be possible to negotiate a mode of interaction between a client and a Web service.

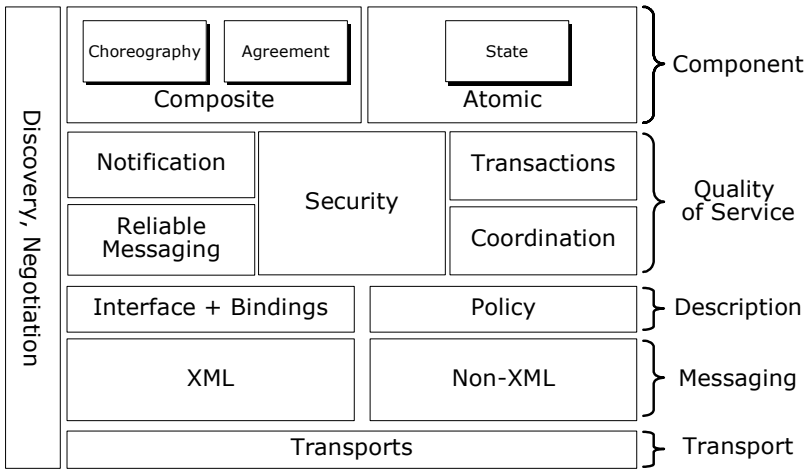

Figure 1: Overall Architecture of a Service Bus

It is important to note the role of dynamic binding: Advanced discovery features of the service bus environment allow searching for services not only in terms of interfaces they support, but also in terms of other properties. These properties are associated with a Web service based on policy mechanisms [WSPF], where a policy may specify such diverse aspects like transaction support, cost of interaction and mode of billing, or semantic descriptions [FHL03]. This way, discovery in "non-IT terms" fosters loose coupling allowing to bind a required Web service not earlier than actually needed at runtime.

Clients and Web services may be operated by different organizations within a company or even within different companies. Thus, integration across platforms has to address all the problems known when crossing organizational boundaries (e.g. security, trust, privacy etc. – see [WSSec]). In this situation, clients and Web services will likely run in different service bus implementations of different vendors, i.e. these implementations have to interoperate.

2.2 Standards

Interoperability in multi-vendor environments implies standardization. A whole series of standards, often collectively referred to as Web service stack or simply WS* (because

many of these standards begin with a WS prefix), have been proposed. To insure standards based interoperability most parties standardizing Web service technology and implementing those standards cooperate on that subject under the umbrella of a dedicated organization [WS-I]. An important deliverable of this organization is a set of so called “profiles” (e.g. [BP-1]): A profile basically is a guideline of how to jointly use a subset of Web service standards to solve problems within a particular domain. For this purpose, a profile resolves ambiguities of individual standards as well as incompatibilities of existing implementations. As a result, interoperability between different service bus implementations is achieved base on support of the same profiles. This is indicated in figure 2 where two service bus implementations comply with Web service standards (WS*) within the bus itself as well as “on the wire”, and where WS-I profile(s) compliance ensures interoperability.

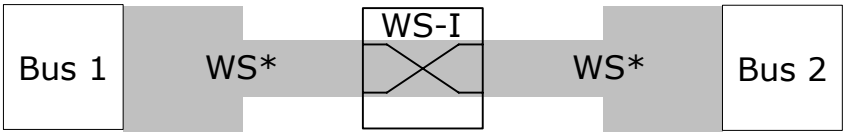

Figure 2: The Role of WS-I

In a nutshell, WS* is overlaying the architecture from figure 1 as follows: The transport layer of the Web service stack is based on ubiquitous protocols like HTTP, TCP/IP, RMI/IIOP etc. With XML Schema as default type system at the messaging layer, interoperability is on a solid base. WSDL is the standard for specifying interfaces of Web services and bindings to corresponding implementations. The series of proposed policy specifications [WSPF] completes the description layer shown. At the quality of service layer a couple of competing specifications have been proposed in the transaction and coordination domain, as well as in the area of reliable messaging. Convergence on common specifications is needed here to ensure broad interoperability; otherwise, interoperability is only realistic between products of vendors supporting the same specification. At the component layer, the resource framework [CCF04a] is likely to be generally accepted for dealing with state of non-composite Web services. Web services composed via choreographies are likely covered by BPEL [BPEL]. The discovery layer is specified via UDDI, metadata exchange, and addressing [WSA] (the latter two supporting bootstrapping).

Sometimes, the large number of specification of WS* is criticised. But this is a result of the principles of modularity and composability underlying the creation of these specifications: A feature required by more than one specification is factored out into a separate specification (modularization) and is used by other specifications for the functionality needed (composability). Note, that composability at the message level is reflected by the SOAP header architecture: This architecture defines that features from WS* specifications requiring a wire representation are reflected as message headers in SOAP messages. For example, transaction context is transported via headers, endpoint references are mapped to header fields, security information is represented by appropriate headers etc.

2.3 Grid and Web Services

Because of the strong aspect of virtualization within the Web service technology stack, tight cooperation between the WS* community and the other major community focussed on virtualization – the Grid community [GGF] – is only natural. In a nutshell, the Grid is about virtualizing IT resources [FK04]: A client requiring resources like compute power, storage, bandwidth etc. passes corresponding requests to the Grid middleware. The Grid middleware will make the requested resources available independent of where the resources reside, or what their detailed characteristics (e.g. processor speed) are etc. For this purpose, the Grid middleware has to dynamically discover and bind resources, secure access to resources, compose resources etc. – just like the service bus sketched above. Thus, work is going on to create a common base of specifications and standards; from an implementation perspective the goal is middleware for both purposes. The direction for basing Grid computing on Web service (Open Grid Services Architecture OGSA) has been sketched in [FNK02].

Web services as specified by WSDL make no statement about "state" at all. Whether a Web service is manipulating some state (like a purchase order service) or not (like a calculator service) is not visible at the WSDL level. But Web services representing IT resources in a Grid environment typically provide a view on the underlying resource's state as first class citizen: A CPU has a processor speed, available main memory etc. In order to reflect this in the WS* stack the resource framework [CFF04a] has been proposed; based on this framework, Grid technology and Web service technology can be combined [CFF04b].

In a nutshell, the resource framework specifies how to associate a type of resources with a Web service, how to distinguish different resources associated with a Web service, the life cycle of resource, etc. [WSRP]. A resource is described by a resource properties document (an XML document) that provides a view on the resource's state. Each property is described by a global XML element definition, and the sequence of these elements is associated with the port type specifying the interface for manipulating the corresponding resource. This way, Web services make the resources they are manipulating explicit (for more details see [CFF04a]). On top of this mechanism, other features originally motivated by Grid requirements are being standardized (e.g. notification capabilities [GNC04]).

3 Application Structure

Web service technology pushes flow-based construction of (application) software into mainstream. I.e. two-level programming as major paradigm for building Web service based applications will likely prevail. As a result, the focus on business processes will increase, and in the shape of choreographies they will become tradable artifacts. This will urge technologies to ease the customization of the resulting applications.

3.1 Two-Level-Programming

[WR90] introduced the concept of two-level programming in the context of workflow technology. [WWC92] used this concept for creating large application systems by scripting coarse grained components. [LR97] extended both approaches towards a workflow-based programming methodology. In a nutshell, two-level programming distinguishes programming-in-the-small from programming-in-the-large (see figure 3): Programming-in-the-small is "traditional" programming, i.e. the creation of application functions based on "traditional" programming languages like Cobol, C++, Java, C# etc. Programming-in-the-large is the "specification" of how individual application functions, especially artifacts stemming from programming-in-the-small endeavours, are used to build overall applications. Today, the artifacts resulting from programming-in-the-small are typically viewed as components. Programming-in-the-large is seen as specifying the control- and data flow between components, i.e. as specifying workflows – or modelling business processes – the activities of which are implemented by functions offered by components [LR00]. Thus, the overall application consists of components and (work-) flows (or business process models, respectively). At runtime executing such an application requires a workflow system interpreting the flows of the application as well as an application server hosting the components of the application. In our context, the components are represented as Web services in WSDL, and flows are represented as choreographies in BPEL. Thus, as first iteration, an application in a Web service world consists of a set of WSDL specifications and BPEL specifications.

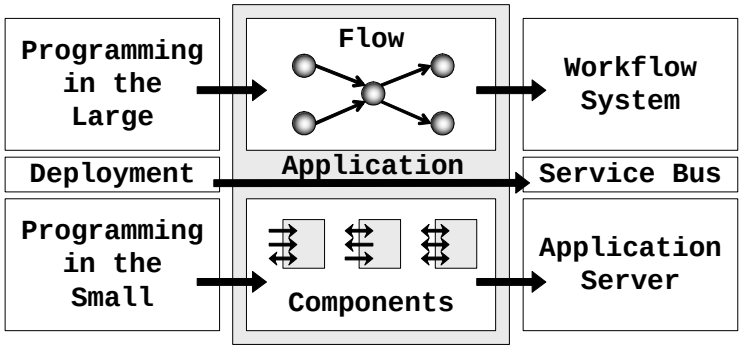

Figure 3: Two-Level-Programming

3.2 Deployment Aspects

The WSDL specifications of an application are not executable code themselves but specify what kinds of functions (i.e. port types in WSDL terminology) are needed by the overall application. Similarly, the activities within a choreography specified as a BPEL process only refer to the definition of a Web service (port type) that implements the required function. Thus, in order to become executable, each activity must be associated with a concrete Web service implementing the functions specified by the corresponding WSDL (i.e. a port in WSDL terminology). This association is specified during

deployment by filling in a deployment descriptor (a concept defined in [J2EE] but used in an extended manner here). Consequently, figure 3 explicitly adds deployment as a third aspect of two-level-programming. It further shows that the artifacts produced by deployment are targeted to the service bus (as detailed next).

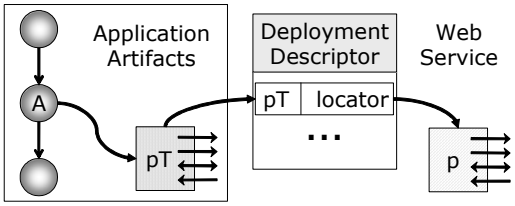

Figure 4: Deployment of Web Service Applications

In figure 4, activity A of a business process is specified to be implemented by an operation of a port type pT. When the artifacts of the corresponding application are installed in the target environment that has to run the application a concrete implementation p of port type pT has to be assigned. This assignment can be done within a deployment descriptor based on a locator. Locators can be of several kinds: For example, a static locator assigns a fixed port with the port type, or a declarative locator assigns a query to the port type. This query is evaluated at runtime to determine a corresponding port whenever the workflow engine navigates through the process model and reaches A. See [KKL04] for more details of a specific implementation within a J2EE-based environment.

The query of a declarative locator may not only be specified in terms of a port type and bindings acceptable for accessing a Web service but also in terms of policies [WSPF]. For example, the activity `setCreditLimit` of a business process is specified to be realized by the `updateMaximumCredit` operation of the `customer` port type. The deployment descriptor specifies that an acceptable implementation of that operation must support encrypted messages only, must not audit the use of the operation, and must be cheaper than 2 cents. Ports implementing the `customer` port type may advertise corresponding properties as policies. When the workflow engine reaches the `setCreditLimit` activity it simply passes the request to invoke an appropriate implementation of this activity to the service bus (together with the actual input data). The service bus then determines an appropriate port based on the associated deployment descriptor, i.e. based on the locator associated with the `customer` port type. If more than one port qualifies, other criteria are used to select one of them. This way, dynamic applications result with loosely coupled Web services.

3.3 Tradable Artifacts

Business processes are more and more seen as important assets of a company. Two-level-programming reflects this by having business processes as explicit artifacts of an application. But a complete application generally consists of more kinds of artifacts than the before mentioned specifications of required Web services (WSDL) and business

processes (BPEL). Additional artifacts like user interfaces (in the form of portlets, e.g.), data sources, business rules, message definitions etc. are needed for a complete application. To be tradable, the artifacts making up an application need to be grouped together and packaged such that they can be transported, installed and deployed.

In [LRS02] such a grouping has been called a "solution". Based on the deployment mechanisms sketched above, solutions can be customized by associating appropriate implementations with the Web services grouped into a solution. The concept of a solution has been refined in [BaS04] towards solution "templates" supporting more flexibility to allow for customization of the corresponding application. For example, customization allows changing business rules guiding the transition from one activity to another within a business process, changing the skin of a portlet to fit into a company's user interface theme, or tailoring which steps performed in a business process have to be audited and monitored by the runtime environment. But additional flexibility is still needed for customizing application: [KB04] argues to explicitly foresee flexibility of structural aspects of process models already at modelling time, and a whole research branch is focussing especially on being able to change the model even of an already running process [RD98], [RDD03]. [AW01] propose an inheritance relationship between processes where "subclassed" processes can be changed without violating the contract to its users represented by the "superclass" process. Much more research is needed in this area to consolidate and combine the various concepts proposed and to define the notion of a customizable template representing an overall application.

4 Utility Computing

Web services can be accessed over the Internet independent of their location in a secure and reliable manner. This way, Web service technology facilitates the use of functions hosted by service providers. Since choreographies are themselves Web services even complete business processes can be hosted by service providers (see [GCC], for example). Since business processes are key artifacts of applications, Web services facilitate outsourcing of (parts of) complete applications. Outsourcing is an important subject for companies today, promising not only cost savings but also significantly improved agility [RW04]. The so-called utility computing model eases outsourcing and is likely to become an integral part of information technology [Ra04]. This model allows to move application functions to (internal or external) service providers and to pay for these functions on a "per use base". In this section we sketch some usages of Web services in utility computing.

4.1 Provisioning

When hosting (parts of) an application at a utility provider the application must be customized to the outsourcing company's need and an appropriate environment for running the customized application must be set up reflecting non-functional requirements. Next, the artifacts making up the application have to be installed, deployed and configured. Considering applications as solution templates in the sense of (3.3) will

support customization. Setting up the environment can be facilitated by Web service technology because the corresponding IT resources are seen as being resource within a Grid run by the service provider. The software artifacts to be installed can be considered to be resources too managed by containers called “resource managers” that are in turn rendered as Web services. Deployment and configuration information is derived from requirements and service level agreements negotiated with the utility provider. It is interesting to note that choreographies can be automatically generated from all of this information that are then run to set up the environment, install, deploy, and configure the customized application ([ACG04], [MBC03]).

4.2 Autonomic Computing

Once the environment has been set up and the application is in use, the utility provider must operate and manage the application, and must make sure that the service level agreements are kept. In large environments typically run by utility providers, this would be a major endeavour for human beings, i.e. these tasks must be automated as far as possible by the IT environment itself in order to be cost effective. Autonomic computing (or sometimes called organic computing) initiatives have their origin here.

At the heart of an autonomic infrastructure is a control loop that monitors the environment, analyzes the captured data to detect or predict potential problems, creates plans about how to fix the problems, and executes the plans to actually resolve those problems. For that purpose, the environment and application artifacts are hosted by corresponding resource managers. A resource manager provides the interfaces that support monitoring (“sensors”) as well as taking corrective actions (“effectors”) on the hosted resources. So-called “autonomic managers” are elements of the overall environment that implement such control loops based on the sensors and effectors of resource managers (see [St03] for more details). The plans created within the control loop are often workflows that are passed to a workflow system to execute the planned corrective actions [ACG04]. Policies provide the base for autonomic managers for detecting problems and planning corrective actions [DDK04].

4.3 Role of the Service Bus

Obviously, a service bus can be seen as the underpinning for creating such an environment. The elements of the IT environment required by an application as well as artifacts of an application itself are resources as sketched in (2.3), i.e. these elements are available via the service bus. Sensors and effectors are Web services that are accessed via the bus. Monitoring is supported by Web service based notification technology [GNC04]. Choreographies represent plans created to manage the overall environment, which in turn use effectors rendered as Web services.

5 Conclusion

We have sketched the architecture of a service bus facilitating integration of both, application functions as well as IT resources in heterogeneous multi-vendor environments. The structure of applications based on Web service technology has been described, which is fundamentally derived from the two-level programming paradigm. Such applications can be run based on a utility computing model as finally described. This model supports outsourcing of information technology and agility of companies, as well as automation of IT infrastructure.

One aspect of outsourcing is moving the actual development of application functions (programming-in-the-small) into low-income countries. This kind of off-shoring might have severe socio-political impacts (see [Me04]). Luckily, business processes are often seen as corporate assets that must be treated as secrets and, thus, have to be specified (programming-in-the-large) in-house. Consequently, programming-in-the-large will likely continued to be done in high-income countries, i.e. where corporate headquarters are. Since programming-in-the-large and programming-in-the-small require different skill sets, the implicit impact of Web services on education must be discussed.

Not only this way, the two-level-programming paradigm fostered by Web service technology further increases the focus on business processes: Business processes are not only understood as key assets internal to an enterprise [S01] but also for cross-enterprise activities (see [ebXML], [RN], [RNWS], [WSCL] for example). With BPEL as a language for choreographies [LR04] widely supported by vendors [BPTc] and implemented by products (see [KKL04], for example) business processes will become a major aspect of creating applications.

In [L03] we sketched a lot of tasks that have to be worked on to make the whole vision work: Some of these tasks have been addressed in the meantime by researches but many are still unsolved. We added some additional tasks in this paper hopefully showing that Web service technology is not only an interesting area of development and standardization activities but also of thrilling research issues.

References

[AW01] W.M.P. van der Aalst, M. Weske. The P2P approach to Interorganizational Workflows, Proc. CAiSE'01, LNCS 2068, Springer 2001.

[ACG04] K. Appleby, S.B. Calo, J.R.Giles, K.-W.Lee. Policy-based automated provisioning, IBM Systems Journal 43(1) (2004).

[ACK04] G. Alonso, F. Casati, H. Kuno, V. Machiraju. Web Services, Springer 2004.

[BS95] M.L. Brodie, M. Stonebraker. Migrating legacy stystems, Morgan Kaufmann 1995.

[BaS04] M.T. Schmidt, K. Bhaskaran. WebSphere Business Integration: An architectural overview, IBM Systems Journal 43(2) (2004).

[CFF04a] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, F. Leymann, M. Nally, T. Storey, S. Tuecke, S.Weerawaranna. Modeling Stateful Resources with Web Services, Globus Alliance & IBM, 2004.

[CFF04b] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, S. Tuecke. From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution, Fujitsu, Globus Alliance & IBM, 2004.

[DDK04] A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, A. Yousse. Web services on demand: WSLA-driven automated management, IBM Systems Journal 43(1) (2004).

[FKN02] I. Foster, C. Kesselmann, J. Nick, S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, GGF 2002. http://www.globus.org/research/papers/ogsa.pdf

[FK04] I. Foster, C. Kesselmann. The Grid 2, Morgan Kaufmann 2004.

[FHL03] D. Fensel, J. Hendler, H. Lieberman, W. Wahlster. Spinning the Semantic Web, MIT Press 2003.

[GNC04] S. Graham, P. Niblett, D. Chappell, A. Lewis, N. Nagaratnam, J. Parikh, S. Patil, S. Samdarshi, I. Sedukhin, D. Snelling, S. Tuecke, W. Vambenepe, B. Weihl. Publish-Subscribe Notification for Web services, Akamai, Computer Associates International, Fujitsu Laboratories of Europe, Globus, Hewlett-Packard, IBM, SAP, Sonic, TIBCO 2004. http://www-106.ibm.com/developerworks/library/ws-pubsub/WS-PubSub.pdf

[HW04] G. Hohpe, B. Woolfe. Enterprise Integration Patterns, Addison-Wesley 2004.

[J2EE] Java™ 2 Platform Enterprise Edition Specification v1.4, Sun Microsystems Inc. 2002.

[Ke02] W. Keller. Enterprise Application Integration, dpunkt.verlag 2002.

[KB04] D. Karastoyanova, A. Buchmann. Automating the Development of Web Service Compositions Using Templates, Proc. Workshop “Geschäftsprozessorientierte Architekturen“, (Ulm, Germany, September 2004).

[KKL04] M. Kloppmann, D. König, F. Leymann, G. Pfau, D. Roller. Business process choreography in WebSphere: Combining the power of BPEL and J2EE, IBM Systems Journal 43(2) (2004).

[KKL03] D. König, M. Kloppmann, F. Leymann, G. Pfau, D. Roller. Web Services Invocation Framework: A Step towards Virtualization Components, Proc. XMIDX 2003 (Berlin, Germany, February 2003).

[L03] F. Leymann. Web Services: Distributed applications without limits, Proc. BTW'03 (Leipzig, Germany, February 2003), Springer 2003.

[LR97] F. Leymann and D. Roller, Workflow based applications, IBM Systems Journal 36(1) (1997).

[LR00] F. Leymann, D. Roller. Production Workflow: Concepts and Techniques, Prentice Hall, 2000.

[LR04] F. Leymann, D. Roller. Modeling Business Processes with BPEL4WS, Modellierung 2004 (Marburg, Germany, March 24-26, 2004), Springer 2004.

[LRS02] F. Leymann, D. Roller, M.-T. Schmidt. Flows and Web Services: B2B aspects of business process management, IBM Systems Journal 41(2) (2002).

[MBC03] E. Manoel, S. Brumfield, K. Converse, M. DuMont, L. Hand, G. Lilly, M. Moeller, A. Nemati, A. Waisanen. Provisioning On Demand. IBM Redbook SG24-8888-00, 2003. (ibm.com/redbooks)

[MMJ01] J. Melton, J.-E. Michels, V. Josifovski, K. Kulkarni, P. Schwarz. SQL/MED — A Status Report, ACM SIGMOD Record 30(1) (2001).

[Me04] P. Mertens. Information Technology in Germany – A Discontinued Model? (in German), Informatik Spektrum 27(3) (2004).

[Ra04] M.A. Rappa. The utility business model and the future of computing services, IBM Systems Journal 43(1) (2004).

[RD98] M. Reichert, P. Dadam. ADEPTflex - Supporting Dynamic Changes of Workflows Without Losing Control, Journal of Intelligent Information Systems 10(2) (1998).

[RW04] J.W. Ross, G. Westerman. Preparing for utility computing: The role of IT architecture and relationship management, IBM Systems Journal 43(1) (2004).

[RRD03] M. Reichert, S. Rinderle, P. Dadam. ADEPT Workflow Management System: Flexible Support For Enterprise-wide Business Processes (Tool Presentation), Proc. Int'l Conf. on Business Process Management (BPM '03), Eindhoven, Netherlands, June 2003, LNCS 2678, pp. 371-379).

[S01] O.A. El Sawy. Redesigning enterprise processes for e-business, McGraw-Hill 2001.

[St03] D. H. Steinberg. What you need to know now about autonomic computing, Part 2: The infrastructure, IBM Developerworks, 2003. ftp://www6.software.ibm.com/software/developer/library/i-autonom2.pdf

[VGH04] K. Vollmer, M. Gilpin, J. Hoppermann. ESB Usage Will Grow As Standards Mature, Forrester Research, 2004.

[WR90] H. Wächter, A. Reuter. Base concepts and realization strategies of the ConTract model (in German), Informatik Forschung und Entwicklung 5 (1990), 202 – 212.

[WWC92] G. Wiederhold, P. Wegner, S. Ceri. Towards Megaprogramming: A paradigm for component-based programming, Comm. ACM 35(22) 1992, 89 – 99.

Links (as of 7/18/2004)

[BPEL] Business Process Execution Language for Web Services V1.1, BEA, IBM, Microsoft, SAP & Siebel, 2003, http://www-106.ibm.com/developerworks/library/ws-bpel/

[BPTc] OASIS BPEL Technical Committee, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

[BP-1] Basic Profile Version 1.0, ws-i.org, 2004 http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html

[ebXML] ebXML, http://www.ebxml.org/

[GCC] http://www.grandcentral.com

[GGF] The Global Grid Forum, http://www.ggf.org

[RN] RosettaNet, http://www.rosettanet.org/

[RNWS] RosettaNet and Web Services, http://www.rosettanet.org/RosettaNet/Doc/0/IP0QL046K55KFBSJ60M9TQCPB3/RosettaNet+Web+ServicesFINAL+.pdf

[WSA] Web Services Addressing, BEA, IBM & Microsoft, 2004, http://www-106.ibm.com/developerworks/webservices/library/ws-add/

[WSCL] Web Services Choreography Description Language, W3C Working Draft, 2004, http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/

[WSI] Web Services Interoperability Organization, http://www.ws-i.org

[WSIF] Web Service Invocation Framework, http://ws.apache.org/wsif/

[WSPF] Web Services Policy Framework, ftp://www6.software.ibm.com/software/developer/library/ws-policy.pdf

[WSRP] Web Services Resource Properties, IBM, Globus, Computer Associates International, Fujitsu Laboratories of Europe, Hewlett-Packard, 2004, http://www-106.ibm.com/developerworks/library/ws-resource/ws-resourceproperties.pdf

[WSSec] Security in a Web Services World: A Proposed Architecture and Roadmap, IBM & Microsoft, 2002, http://msdn.microsoft.com/library/en-us/dnwssecur/html/securitywhitepaper.asp