The (Service) Bus: Services Penetrate Everyday Life

Frank Leymann

Institute of Architecture of Application Systems,
University of Stuttgart, Universitätsstr. 38,
70569 Stuttgart, Germany

Frank.Leymann@informatik.uni-stuttgart.de

Abstract: We sketch the vision of a ubiquitous service bus that will be the base for hosting and accessing services everywhere. The utility model for using IT artifacts is implied. Applications on top of the service bus will be centered on business processes and will be adaptive in multiple dimensions. The ubiquitous service bus will change the way we think about information technology.

1 Introduction

Service oriented computing and service oriented architectures are accepted as the next step in building distribute applications. Especially, Web services ([1], [16]) as particular incarnation of service oriented technology has broad acceptance in the industry and is supported by products of many vendors.

In this paper we sketch the vision of a globally available infrastructure for hosting and accessing services everywhere. Services in our context are not only software functions usually thought of when talking about services but also hardware artifacts. The latter is brought to the area of service orientation by recent movements of Grid computing towards Web service technology [4].

Section 2 describes the basic component of this infrastructure, namely the service bus, and its key capabilities supporting our vision. The new model of using IT in a manner we are familiar with from traditional utilities is sketched in Section 3. Application structures fostered by the envisioned infrastructure and envisioned exploitation model are portrayed in Section 4.

2 The Bus

The architecture of a middleware platform for realizing service oriented computing based on Web service standards is outlined in [16]. We refer to this middleware simply as service bus. Complying to Web service standards a particular implementation of a service bus interoperates by definition with all other implementations of a service bus – at least when ignoring all the interoperability issues addressed by initiatives like WS-I, which we take the liberty to do in sketching our vision. In this sense, the collection of

B. Benatallah, F. Casati, and P. Traverso (Eds.): ICSOC 2005, LNCS 3826, pp. 12 – 20, 2005.

interacting service bus implementations can be viewed as one single piece of middleware referred to as *the service bus* (or even just *the bus*) – similar to the Web being realized by a collection of interacting components like HTTP origin servers, proxies, browsers, etc.

2.1 Virtualization

The main functionality of the service bus is *virtualization* (see Fig. 1): Since all services accessible via the service bus are described by WSDL the service bus hides from a user of a service the implementation details of a service like the programming language used for its implementation, the hosting application server, the underlying operating system platform, and so on. When making a request, a user of a service simply refers to the (WSDL) interface an implementation of which is needed and passes the data to be processed by an implementation, and the service bus will select one of the available corresponding implementations of this interface to perform the user's request [11].

To further support proper selection done by the service bus, both, requests as well as services may be annotated by policies. Policies describe non-functional properties like transactional capabilities, security features, costs etc. Basically, services publish the non-functional properties they support, and requests specify the non-functional properties expected. The service bus uses the policies associated with a request to further reduce the number of matching services. In doing so, the service bus determines based on both policies an "effective policy" that will govern the actual interaction between the requestor and the service chosen.

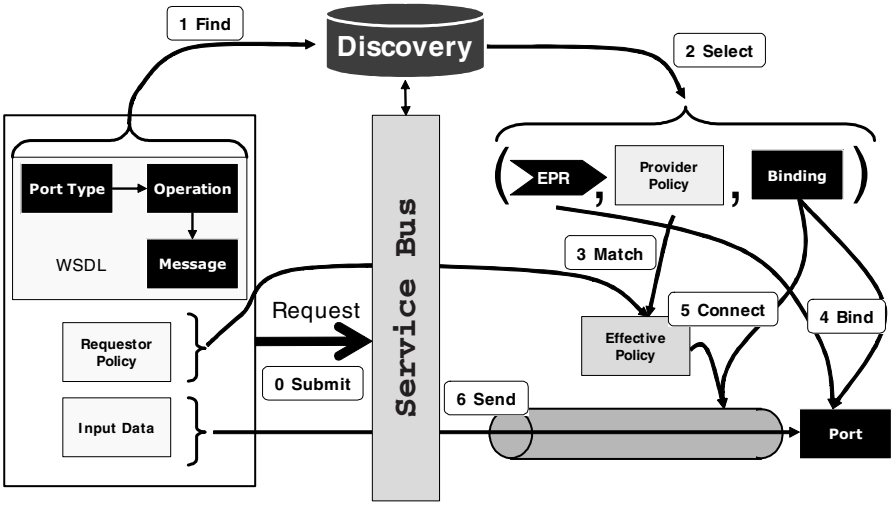

Fig. 1. Executing Requests within the Bus

The service bus may even support requests without requestors having to specify the interface a service has to implement. Services may be annotated with semantics describing the business meaning of the functions provided. In turn, requestors have to provide semantic descriptions of the function requested instead of explicitly naming a corresponding interface. The service bus locates and selects a matching service based on these semantic descriptions. Web services describing the functions they offer semantically are referred to as "semantic Web services" [6].

2.2 Optimization

Thus, the service bus virtualizes services based on interface descriptions or semantic descriptions as well as based on non-functional properties: All service implementations supporting the interface or semantic description as well as the non-functional properties of a request are interchangeable from the requestor's point of view. The corresponding services can be jointly viewed as a pool of services being able to satisfy the request. The members of such a pool can be distributed over the network, they can be implemented in very different environments, they can be accessible over very different protocols etc. In case more than one service qualifies the service bus will use additional criteria to select a particular member from that pool.

When selecting such an implementation on behalf of a user the service bus has all liberties as long as the service chosen matches the functional and non-functional properties of the request of the user. Consequently, the spectrum of possible mechanisms for choosing a member from the pool of all qualifying services reaches from very simply mechanisms like random selections up to sophisticated optimization mechanisms.

Optimization may be done according to various sets of criteria. For example, the service bus may consider the workload of the overall environment (evenly distributing work), the cost of mediating the request (preferring local implementations of a service over remotely available implementations) etc. Optimization may favor implementations of the provider of the local service bus used by the requestor, may strive towards reducing costs for the requestor, or may strive towards maximizing profit across all requests served by a certain provider considering a set of service level agreements, and so on (see Section 3.3 below).

2.3 Management

To enable optimization the service bus must able to retrieve information required for the various kinds of optimizations like state data and so on. Similarly, the service bus must be able to influence the state of services like restarting a certain service. For this purpose, services have to support corresponding interfaces in addition to the interface providing the proper (application) functions.

Data that is providing the context for performing a request offered by a service is referred to as a "WS-Resource". An element of this data context is called a "resource property". The resource framework ([24], [25]) specifies certain requests to manipulate resource properties and to manage the lifecycle of WS-Resources (see [18] for more details). Changes of resource properties may influence optimization decisions or may require actions on the corresponding services to change their state.

For this purpose, a topic-based notification or publish/subscribe infrastructure has been defined that is part of the service bus itself ([26], [27]) and allows to register for changes of resource properties.

The corresponding infrastructure can be used in a much broader sense: Any data required to decide about proper management of a resource may be specified as a collection of corresponding resource properties. Here, a *resource* is any software or hardware artifact made available as a (manageable) service. Resources of different types support specific operations to enable management of its instances. The publish/subscribe features of the service bus may then be used by systems management components to monitor resources and properly react by using the resource specific interfaces. In this sense, the service bus itself becomes the basis for managing an overall environment in a service-oriented manner [28].

The publish/subscribe features can also be used as the basis for realizing feedback loops to control resources in an autonomic manner [13]: The monitoring component of such a feedback loop filters and aggregates notification events from the resources, the analysis component correlates the remaining events and predicts potential critical situations, the planning component decides on actions needed to prevent such situations by generating a corresponding plan, and the execution component performs this plan (see [20] for more details). After executing the plan, the predicted situation is unlikely to occur. Furnishing the service bus with such feedback loops results in an infrastructure that can protect, optimize, and heal itself [5].

For example, a critical situation may indicate that a certain application needs more resources to meet its goals in terms of the number of users to be supported with a certain response time. The plan for preventing not meeting this goal is a flow with activities that use the interfaces of the resource types required (like CPUs, storage, installation services). After executing such a "provisioning flow" [8] additional resource are available to the application such that it will not miss its goal [2].

3 Utility Computing

The service bus is the basis for sharing resources. For example, resources owned by one company can be made available to other parties – and this can be done on a fee base enabling a business for outsourcing IT artifacts.

3.1 Traditional ASP Model

Abstracting from technology, this is the "traditional" service provider model. Within the application service provider (ASP) model, the provider hosts, runs, maintains an application on behalf of another company for a fee. When the ASP model came up fees had been determined upfront based on an estimation of the resources needed to satisfy the non-functional requirements (response time, availability, number of users etc.) of a customer. Typically, these estimations were based on expected peak loads and as a consequence, customers had to pay for resources that they seldom need. This is often seen as an obstruction to the broad acceptance of the ASP model – despite the fact that companies ask for the ability to outsource parts of their IT infrastructure to be able to focus on their core business competencies.

3.2 Dynamic Provisioning

Dynamic provisioning technology and autonomic technology will remove this hurdle: Customers specify service levels objectives for outsourced resources with their provider and will only pay for the actual resources used. At the provider side this is based on the kind of feedback loops sketched above that ensure to meet the service levels agreed, with provisioning flows being performed when service level objectives are jeopardized. When dynamically provisioned resources are no longer needed they will be automatically de-provisioned. Thus, over-provisioning will no longer take place as in the original ASP model resulting in an economy of scale that reduces fees for outsourced resources.

The corresponding model is referred to as "utility model" [14]: Paying only for what has been actually used is the model of classical utilities (power, gas, water…). Using compute resources (both, software and hardware) in such a manner will create a new kind of utility called "computing utility". Making compute resources available when needed and for the time needed is also called "computing on demand".

3.3 Software as a Service

Using software in the utility model implies that the provider does also provide the hardware and middleware required to actually run the software. Thus, using software in the utility model typically means for a customer to outsource the corresponding complete infrastructure to the provider. The customer uses "software as a service" (aka SaaS).

If critical functions are used based on this model customers negotiate service level objectives such as average response time, availability etc. with the service provider. The agreed too set of objectives together with fees to be paid by the customer if the objectives are met and penalties to be paid by the provider if objectives are not met result in a service level agreement (SLA) [3]. The service provider strives to optimize profits based on the set of SLAs negotiated with his customers: This is one kind of optimization mentioned above (see Section 2.2) that the service bus must support (either directly or by some component extending its functionality).

4 Applications

Often, services are composed of other services. Business processes are the most widespread example for such a composition. The term *orchestration* got established in the meantime for supporting the composition of services into a business process. Since the services offered by an orchestration may be used in other orchestrations a recursive composition model for services results. BPEL ([21], [22]) is the established language for specifying orchestrations in the Web service area.

When specifying an orchestration it is opaque whether or not a service used is also an orchestration. If a service used in an orchestration is again structured as an orchestration and both of the structures are considered for composition more precise interaction details can be specified. Such a specification is referred to as *choreography* in the meantime. WS-CDL [23] has been proposed as language for specifying choreographies.

4.1 Structure

Applications based on services thus consist of orchestrations and services they use, i.e. applications in a service environment are based on a two-level programming model ([9], [17]). To be precise, an orchestration specifies the types of services used, and during deployment of an orchestration additional information must be provided that allows the service bus to select appropriate services at runtime of the orchestration (see [12], [13]).

The overall infrastructure, thus, includes an orchestration engine as an integral part which is typically based on workflow systems [10] that support BPEL. Even business processes that include interactions with human beings may be supported [19]. The orchestration engine navigates through the underlying process model and determines the kind of service needed. The underlying service bus selects a matching service on behalf of the orchestration engine and returns the response of the service chosen to it.

In doing so, quality of services are folded in based on policies that describe the requirements of the business process and polices that are associated with the candidates considered by the service bus (see Section 2.1). For example, a business process may specify that messages exchanged between the orchestration and the service chosen must be encrypted and transported reliably, or that an invocation of a service must be done transactional. Thus, non-functional requirements of an application can be specified that will be enforced at runtime by the service bus.

4.2 Adaptability

The service bus may choose for different instances of the same business process model different services for one and the same activity of the business process model. Thus, the overall orchestration is adaptive in terms of services used, i.e. the underlying services available to an orchestration may change in terms of different providers, different implementation etc. This is similar to adaptability in terms of people performing a certain activity which is supported since long in workflow systems [10]. Adaptability in terms of services chosen may even go further by supporting the selection of services that deviate from the type of service prescribed by the business process model [7]. Adaptability in terms of the logic (i.e. control and data flow) of an orchestration may be supported too [15]. Finally, based on dynamic provisioning technology the environment hosting an application is adaptive too as described in Section 2.3.

Additional flexibility can be supported based on providing “skeletons” of process models (Fig. 2). Being characterized as a skeleton has various aspects, represented to the outside via “points of variability” (see v_1,…,v_4 in the figure below). For example: A business process model may only specify that certain kind of activities have to be performed and the type of messages exchanged with each of such an activity, but the type of services to be used is not specified – the type has to be detailed by the organization deploying the process model (point of variability v_1 below). Or a business process model may define some of its structure as fixed while other parts of the model may be changed by the deploying organization; point of variability v_2 below allows to change transition condition q, for instance, and point of variability v_4 allows to omit

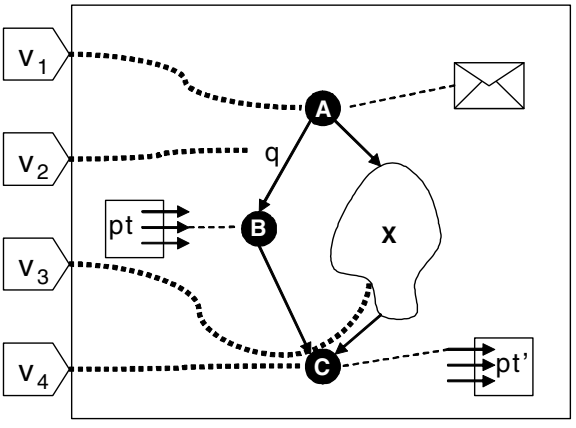

Fig. 2. Variable Applications

activity C at all in the business process model. Or a business process model vaguely specifies that some sort of actions must happen in course of the business process but the whole corresponding fragment of the model must be provided by the deploying organization (point of variability v_3 below). Or a business process model may only specify its externally observable behavior while its internal implementation is "arbitrary" (and possibly hidden) as long as the specified behavior results.

This spectrum of adaptability is important for reasons like customization of applications, representing best practices, or specifying constraints for using collections of services. Applications will externalize their points of variability, and tools will present them allowing to modify the applications accordingly. Not only will application logic be represented as points of variability but also environment aspects of an application; these aspects correspond to service level objectives, for example, which influence the selection of underlying hardware, middleware etc. to satisfy the objectives. Dynamic provisioning (Section 3.2) and using software as a service (Section 3.3) will make use of these kinds of points of variability to negotiate SLAs and set up the overall environment appropriately (see [8] and [13] for more details).

4.3 Outsourcing

Since BPEL itself is portable across environments customers can specify their business processes in BPEL and run them anywhere in the environment. The services needed by the orchestration are selected by the bus based on deployment information specified for the orchestration. This selection can be influenced by preferences of the provider of the hosting infrastructure of the orchestration, i.e. the provider may itself offer the corresponding services or may have special contracts with other providers of those services. Thus, a customer may outsource a business process completely, even without taking care about the providers of the services composed by the corresponding orchestration. I.e. the utility computing model applies to complete business processes and applications.

5 Conclusion

The current Web is an infrastructure for accessing content everywhere ("content Web"). Web service technology will likely provide an infrastructure for accessing services everywhere ("service Web"). Since Web service technology is not restricted to Web protocols access to services over any kind of suitable protocols, across heterogeneous environments will be supported. Quality of services used from today's application servers will be supported by the service bus. Composition of services from other services available on the bus will be the way of building new applications. These applications can be hosted anywhere on the bus resulting in a utility model for IT artifacts. As a consequence, outsourcing and off-shoring of IT will become ubiquitous allowing companies to focus on their core business.

References

1. G. Alonso, F. Casati, H. Kuno, V. Machiraju. Web Services, Springer 2004.
2. K. Appleby, S.B. Calo, J.R.Giles, K.-W.Lee. Policy-based automated provisioning, IBM Systems Journal 43(1) (2004).
3. 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).
4. I. Foster, C. Kesselmann. The Grid 2, Morgan Kaufmann 2004.
5. A.G. Ganek, T.A. Corbi. The dawning of the autonomic computing area, IBM Systems Journal 42(1) (2003).
6. M. Hepp, F. Leymann, J. Domingue, A. Wahler, D. Fensel. Semantic Business Process Management: Using Semantic Web Services for Business Process Management, Proc. IEEE ICEBE 2005 (Beijing, China, October 18-20, 2005).
7. D. Karastoyanova, A. Houspanossian, M. Cilia, F. Leymann, A. Buchmann. Extending BPEL for Run Time Adaptability, Proc. EDOC'2005, (Enschede, The Netherlands, September 19 – 23, 2005).
8. A. Keller, R. Badonnel. Automating the Provisioning of Application Services with the BPEL4WS Workflow Language, Proc. DSOM 2004 (Nancy, France, November 2004).
9. F. Leymann, D. Roller. Workflow based applications, IBM Systems Journal 36(1) (1997) 102-123.
10. F. Leymann, D. Roller. Production Workflow: Concepts and Techniques, Prentice Hall 2000.
11. F. Leymann. Web Services: Distributed applications without limits, Proc. BTW'03 (Leipzig, Germany, February 2003), Springer 2003.
12. F. Leymann. The Influence of Web Services on Software: Potentials and Tasks, Proc. 34th Annual Meeting of the German Computer Society (Ulm, Germany, September 20 – 24, 2004), Springer 2004.
13. F. Leymann, Combining Web Services and the Grid: Towards Adaptive Enterprise Applications, Proc. CAiSE/ASMEA'05 (Porto, Portugal, June 2005).
14. M.A. Rappa. The utility business model and the future of computing services, IBM Systems Journal 43(1) (2004).
15. M. Reichert, P. Dadam. ADEPTflex - Supporting Dynamic Changes of Workflows Without Losing Control, Journal of Intelligent Information Systems 10(2) (1998).

16. S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D.F. Ferguson. Web Services Platform Architecture, Prentice Hall 2005.
17. G. Wiederhold, P. Wegner, S. Ceri. Towards Megaprogramming: A paradigm for component-based programming, Comm. ACM 35(22) 1992, 89 – 99.

Links: (followed on September 17, 2005)
18. K. Czajkowski, D. Ferguson, I. Foster, J. Frey, F. Leymann, M. Nally, T. Storey, S. Tuecke, S.Weerawarana. Modeling stateful resources with Web services, Globus Alliance & IBM, 2004, http://www.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf
19. M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt, I. Trickovic, WS-BPEL Extension for People (BPEL4People), IBM, SAP 2005 http://www-128.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/
20. 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
21. Business Process Execution Language For Web Services V1.1, BEA, IBM, Microsoft, SAP & Siebel, 2003, http://www-128.ibm.com/developerworks/library/specification/ws-bpel/
22. OASIS BPEL Technical Committee, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel
23. Web Services Choreography Description Language, W3C Working Draft, 2004, http://www.w3.org/TR/ws-cdl-10/
24. Web Services Resource Framework, IBM, Globus, Computer Associates, Fujitsu, Hewlett-Packard http://www-106.ibm.com/developerworks/library/ws-resource/
25. OASIS Resource Framework Technical Committee, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf
26. Web Services Notification, Akamai, Computer Associates, Fujitsu, Globus, Hewlett-Packard, IBM, SAP, Sonic Software, TIBCO Software 2004, http://www-106.ibm.com/developerworks/library/specification/ws-notification/
27. OASIS WS-Notification Technical Committee, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn
28. OASIS Web Services Distributed Management (WSDM) http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm