Service -Oriented Computing: Concepts, Characteristics and Directions

Mike P. Papazoglou
Tilburg University
INFOLAB,
Dept. of Information Systems and Management,
PO Box 90153, Tilburg 500 LE, The Nethrlands
mikep@uvt.nl

Abstract

Service-Oriented Computing (SOC) is the computing paradigm that utilizes services as fundamental elements for developing applications/solutions. To build the service model, SOC relies on the Service Oriented Architecture (SOA), which is a way of reorganizing software applications and infrastructure into a set of interacting services. However, the basic SOA does not address overarching concerns such as management, service orchestration, service transaction management and coordination, security, and other concerns that apply to all components in a services architecture.

In this paper we introduce an Extended Service Oriented Architecture that provides separate tiers for composing and coordinating services and for managing services in an open marketplace by employing grid services.

1 Introduction

Service-Oriented Computing (SOC) is the computing paradigm that utilizes services as fundamental elements for developing applications/solutions. Services are self-describing, platform-agnostic computational elements that support rapid, low-cost composition of distributed applications. Services perform functions, which can be anything from simple requests to complicated business processes. Services allow organizations to expose their core competencies programmatically over the Internet (or intra-net) using standard (XML-based) languages and protocols, and be implemented via a self-describing interface based on open standards.

Because services provide a uniform and ubiquitous information distributor for wide range of computing devices (such as handheld computers, PDAs, cellular telephones, or appliances) and software platforms (e.g., UNIX or Windows), they constitute the next major step in distributed computing.

Services are offered by service providers - organizations that procure the service implementations, supply their service descriptions, and provide related technical and business support. Since services may be offered by different enterprises and communicate over the Internet, they provide a distributed computing infrastructure for both intra- and cross-enterprise application integration and collaboration. Clients of services can be other solutions or applications within an enterprise or clients outside the enterprise, whether these are external applications, processes or customers/users. Consequently, to satisfy these requirements services should be:

- *Technology neutral:* they must be invocable through standardized lowest common denominator technologies that are available to almost all IT environments. This implies that the invocation mechanisms (protocols, descriptions and discovery mechanisms) should comply with widely accepted standards.
- *Loosely coupled:* they must not require knowledge or any internal structures or conventions (context) at the client or service side.
- Support *location transparency:* services should have their definitions and location information stored in a repository such as UDDI and be accessible by a variety of clients that can locate and invoke the services irrespective of their location.

Services come in two flavors: *simple* and *composite services*. The unit of reuse with services is functionality that is in place and readily available and deployable as services that are capable of being managed to achieve the required level of service quality. Composite services involve assembling existing services that access and combine information and functions from possibly multiple ser-

Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE'03)
0-7695-1999-7/03 $17.00 © 2003 IEEE

vice providers. For example, consider a collection of simple services that accomplish a specific business task, such as order tracking, order billing, and customer relationships management. A enterprise may offer a composite web service that composes these services together to create a distributed e-business application that provides customized ordering, customer support, and billing for a specialized product line (e.g., telecommunication equipment, medical insurance, etc). Accordingly, services help integrate applications that were not written with the intent to be easily integrated with other distributed applications and define architectures and techniques to build new functionality while integrating existing application functionality.

Service-based applications are developed as independent sets of interacting services offering well-defined interfaces to their potential users. This is achieved without the necessity for tight coupling of distributed applications between transacting partners, nor does it require predetermined agreements to be put into place before the use of an offered service is allowed.

While the services encapsulate the business functionality, some form of inter-service infrastructure is required to facilitate service interactions and communication. Different forms of this infrastructure are possible because services may be implemented on a single machine, distributed across a set of computers on a local area network, or distributed more widely across several wide area networks. A particularly interesting case is when the services use the Internet as the communication medium and open Internet-based standards. A *web service* is a specific kind of service that is identified by a URI and exhibits the following characteristics:

- It exposes its features programmatically over the Internet using standard Internet languages and protocols, and
- It can be implemented via a self-describing interface based on open Internet standards (e.g., XML interfaces which are published in a network-based repositories).

Interactions of web-services occur as SOAP calls carrying XML data content and the service descriptions of the web-services are expressed using WSDL [15] as the common (XML-based) standard. WSDL is used to publish a web service in terms of its ports (addresses implementing this service), port types (the abstract definition of operations and exchanges of messages), and bindings (the concrete definition of which packaging and transportation protocols such as SOAP are used to inter-connect two conversing end points). The UDDI [14] standard is a directory service that contains service publications and enables web-service clients to locate candidate services and discover their details.

Web services share the characteristics of more general services, but they require special consideration as a result of using a public, insecure, low-fidelity mechanism for inter-service interactions.

This paper is organized as follows. In section 2 we introduce the concept of software as a service, while in section 3 we describe the basic service oriented architecture. Section 4 describes an extended service architecture materialized by grid services and section 5 introduces the concept of service bus for open service marketplaces. Finally, section 6 presents our conclusions.

2 A view of software as a service

The concept of software-as-a-service espoused by SOC is revolutionary and appeared first with the ASP (Applications Service Provider) software model. An ASP is a third party entity that deploys, hosts and manages access to a packaged application and delivers software-based services and solutions to customers across a wide area network from a central data center. Applications are delivered over networks on a subscription or rental basis. In essence, ASPs were a way for companies to outsource some or even all aspects of their information technology needs.

By providing a centrally hosted Intent application, the ASP takes primary responsibility for managing the software application on its infrastructure, using the Internet as the conduit between each customer and the primary software application. What this means for an enterprise is that the ASP maintains the application, the associated infrastructure, and the customer's data and ensures that the systems and data are available whenever needed.

Although the ASP model introduced the concept of software-as-a-service first, it suffered from several inherent limitations such as the inability to develop highly interactive applications, inability to provide complete customizable applications [7]. This resulted in monolithic architectures, highly fragile, customer-specific, non-reusable integration of applications based on tight coupling principles. Today we are in the midst of another significant development in the evolution of software-as-a-service architected for loosely-coupled asynchronous interactions on the basis of XML-based standards with intention to make access to and communications of applications over the Internet easier.

The SOC paradigm allows the software-as-a-service concept to expand to include the delivery of complex business processes and transactions as a service, while permitting applications be constructed on the fly and services to be reused everywhere and by anybody. Perceiving the relative benefits of (web) services technology many ASPs are modifying their technical infrastructures and business models to be more akin to those of web service providers.

3 The basic service oriented architecture

To build integration-ready applications the service model relies on the service-oriented architecture (SOA). SOA is a way of reorganizing a portfolio of previously siloed software applications and support infrastructure into an interconnected set of services, each accessible through standard interfaces and messaging protocols. Once all the elements of an enterprise architecture are in place, existing and future applications can access these services as necessary without the need of convoluted point-to-point solutions based on inscrutable proprietary protocols. This architectural approach is particularly applicable when multiple applications running on varied technologies and platforms need to communicate with each other. In this way, enterprises can mix and match services to perform business transactions with minimal programming effort.

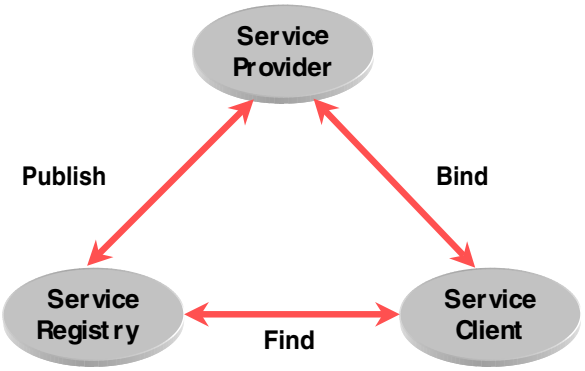

Figure 1. The basic Service Oriented Architecture.

SOA is a logical way of designing a software system to provide services to either end-user applications or other services distributed in a network through published and discoverable interfaces. The basic SOA defines an interaction between software agents as an exchange of messages between service requesters (clients) and service providers. Clients are software agents that request the execution of a service. Providers are software agents that provide the service. Agents can be simultaneously both service clients and providers. Providers are responsible for publishing a description of the service(s) they provide. Clients must able to find the description(s) of the services they require and must be able to bind to them.

The basic SOA is not an architecture only about services, it is a relationship of three kinds of participants: the *service provider*, the *service discovery agency*, and the *service requestor* (*client*). The interactions involve the *publish*, *find* and *bind* operations [4], see Figure-1. These roles and operations act upon the service artifacts: the service description and the service implementation. In a typical service-based scenario a service provider hosts a network accessible software module (a implementation of a given service). The service provider defines a service description of the service and publishes it to a client or service discovery agency through which a service description is published and made discoverable. The service requestor uses a find operation to retrieve the service description typically from a the discovery agency, i.e., a registry or repository like UDDI, and uses the service description to bind with the service provider and invoke the service or interact with service implementation. Service provider and service requestor roles are logical constructs and a service may exhibit characteristics of both.

The fundamental logical view of a service in the basic SOA is that it is a service interface and implementation. A service is usually a business function implemented in software, wrapped with a formal documented interface that is well known and known where to be found not only by agents who designed the service but also by agents who do not know about how the service has been designed and yet want to access and use it. Black box encapsulation inherits this feature from the principles of modularity in software engineering, e.g., modules, objects and components. Services are different from all of these forms of modularity in that they represent complete business functions, they are intend to be reused and engaged in new transactions not at the level of an individual program or even application but at the level of the enterprise or even across enterprises. They are intended to represent meaningful business functionality that can be assembled into a larger and new configurations depending on the need of particular kinds of users particular client channels.

The interface simply provides the mechanism by which services communicate with applications and other services. Technically, the service interface is the description of the signatures of a set of operations that are available to the service client for invocation. The service specification must explicitly describe all the interfaces that a client of this service expects as well as the service interfaces that must be provided by the environment into which the service is assembled/composed. As service interfaces of composed services are provided by other (possibly singular) services, the service specification serves as a means to define how a composite service interface can be related to the inter-

faces of the imported services and how it can be implemented out of imported service interfaces. This is shown in Figure 2. In this sense the service specification has a mission identical to a composition meta-model that provides a description of how the web-service interfaces interact with each other and how to define a new web-service interface (<PrortType>) as a collection (assembly) of existing ones (imported <PrortType>s), see Figure-2. A service specification, thus, defines the encapsulation boundary of a service, and consequently determines the granularity of replaceability of web-service interface compositions. This is the only way to design services reliably using imported services without knowledge of their implementations. As service development requires that we deal with multiple imported service interfaces it is useful to introduce this stage the concept of a *service usage interface*. A service usage interface is simply the interface that the service exposes to its clients [10] . This means that the service usage interface is not different from the imported service interfaces in Figure-2, it is, however, the only interface viewed by a client application.

Figure-2 distinguishes between two broad aspects of services: *service deployment*, which we examined already, versus *service realization*. The service realization strategy involves choosing from an increasing diversity of different options for services, which may be mixed in various combinations including:

- In house service design and implementation. Once a service is specified, the design of its interfaces or sets of interfaces and the coding of its actual implementation happens in-house.

- Purchasing/leasing/paying for services. Complex web-services that are used to develop trading applications are commercialisable software commodities that may be acquired from a service provider, rather than implemented internally. These types of services are very different from the selling of shrink-wrapped software components, in that payment should be on an execution basis for the delivery of the service, rather than on a one-off payment for an implementation of the software. For complex trading web-services, the service provider may have different charging policies such as payment per usage, payment on a subscription basis, lifetime services and so on.

- Outsourcing service design and implementation. Once a service is specified, the design of its interfaces or sets of interfaces and the coding of its actual implementation may be outsourced. Software outsourcings are advantageous in the case of organizations that have become frustrated with the shortcomings of their internal IT departments.

- Using wrappers and/or adapters. Non-component implementations for services may include database functionality or legacy software accessed by means of adapters or wrappers. Wrappers reuse legacy code by converting the legacy functionality and encapsulating it inside components. Adapters use legacy code in combination with newly developed code. This newly developed code may contain new business logic and rules that supplement the converted legacy functionality.

Service descriptions are used to advertise the service capabilities, interface, behavior, and quality. Publication of such information about available services (on a service registry) provides the necessary means for discovery, selection, binding, and composition of services. In particular, the service interface description publishes the service signature while the service capability description states the conceptual purpose and expected results of the service. The (expected) behavior of a service during its execution is described by its service behavior description (e.g., as a workflow process). Finally, the Quality of Service (QoS) description publishes important functional and non-functional service quality attributes, such as service metering and cost, performance metrics (response time, for instance), security attributes, (transactional) integrity, reliability, scalability, and availability.

Service implementation can also be very involved because in many occasions many organizations rely on single monolithic programs to represent the single service or service method implementation. But very often in order to fulfil the functions of a service multiple programs are involved, e.g., programs that belong to multiple applications of new programs that belong to multiple applications. Application composition and integration very often is involved in fulfilling the service. At the logical level of the service we do not pay any attention to this. All we need to know is that there is a business function implemented in software somehow and this is the interface to it. At development time we care, however, how the service is implemented. more specifically, what are the methods and the internal construction of the implementation.

The service in the basic SOA is designed in such a way that it can be invoked by various service clients and is logically decoupled from any service caller (*loose coupling*). Services can be reused and one does not have to look inside the service to understand what it does. There are no assumptions of any kind in the service as to what kind of service consumer is using and for what purpose and in what context. In SOA the service is not coupled with its callers, in fact it knows nothing about them. However, the service callers are very much coupled with the service as they know what the services are what they call and what they can accomplish . In summary, in SOA service consumers make targeted

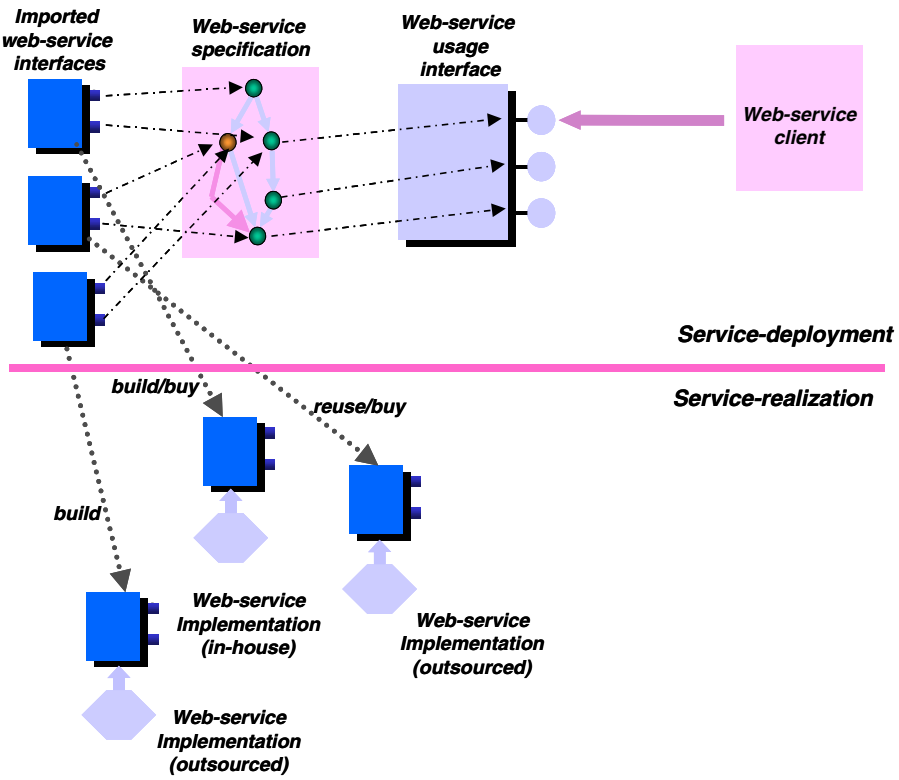

Figure 2. Service interfaces and implementation.

named calls, they target specific services through specific interfaces that are exposed by the services and therefore they are very much dependent on availability and the version of the service that they are calling.

4 Grid services and the extended service oriented architecture

The basic SOA does not address overarching concerns such as management, service orchestration, service transaction management and coordination, security, and other concerns that apply to all components in a services architecture. Such concerns are addressed by the extended SOA (ESOA) [11] that is depicted in Figure-3. This layered architecture utilizes the basic SOA constructs as its bottom layer.

The *service composition layer* in the ESOA encompasses necessary roles and functionality for the consolidation of multiple services into a single composite service. Resulting composite services may be used by *service aggregators* as components (i.e., basic services) in further service compositions or may be utilized as applications/solutions by service clients. Service aggregators thus become service providers by publishing the service descriptions of the composite service they create. A service aggregator is a service provider that consolidates services that are provided by other service providers into a distinct value added service. Service aggregators develop specifications and/or code that permit the composite service to perform functions that include the following:

- Coordination: controls the execution of the component services, and manages dataflow among them and to the output of the component service (e.g., by specifying workflow processes and using a workflow engine for run-time control of service execution).
- Monitoring: allows subscribing to events or information produced by the component services, and publish higher-level composite events (e.g., by filtering, summarizing, and correlating component events).
- Conformance: ensures the integrity of the composite service by matching its parameter types with those of its components, imposes constraints on the component services (e.g., to ensure enforcement of business rules), and performs data fusion activities.
- QoS composition: leverages, aggregates, and bundles the component's QoS to derive the composite QoS, including the composite service's overall cost, per-

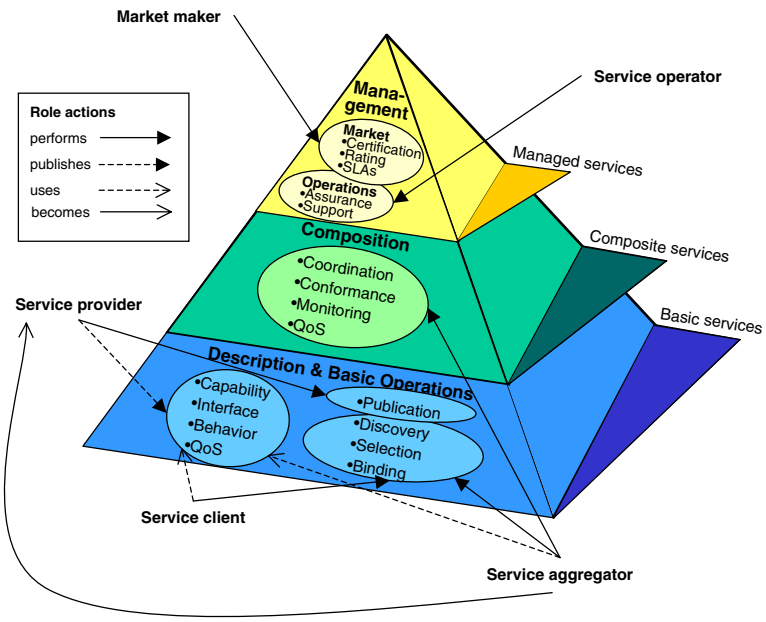

Figure 3. The Extended Service Oriented Architecture.

formance, security, authentication, privacy, (transactional) integrity, reliability, scalability, and availability.

The recently proposed standard Business Process Execution Language for web services (BPEL) [5] is an XML-based effort to addresses the definition of a new web service in terms of compositions of existing services. A BPEL process is defined "in the abstract" by referencing and interlinking portTypes specified in the WSDL definitions of the web services involved in a process.

Managing e-business critical applications requires in-depth administration capabilities and integration across a diverse, distributed environment. Any downtime of key e-business systems has a negative impact on business to the extent of throwing you out of the market. To counter such a situation, enterprises need to constantly monitor the health of their applications. The performance should be in tune, at all times and under all load conditions. Application management is thus an indispensable element of the ESOA that includes performance management and business/application specific management.

To manage critical applications/solutions and specific markets, ESOA provides managed services in the *service management layer* depicted at the top of the ESOA pyramid. In particular, ESOA's *operations management* functionality is aimed at supporting critical applications that require enterprises to manage the service platform, the deployment of services and the applications. Operations management functionality may provide detailed application performance statistics that support assessment of the application effectiveness, permit complete visibility into individual business transactions, and deliver application status notifications when a particular activity is completed or when a decision condition is reached. We refer to the organization responsible for performing such operation management functions as the *service operator*. Depending on the application requirements a service operator may be a service client or aggregator.

Operations management is a critical function that can be used to monitor the correctness and overall functionality of aggregated/orchestrated services thus avoiding a severe risk of service errors. In this way one can avoid typical errors that may occur when individual service-level agreements (SLAs) are not properly matched. This fact was illustrated by the failure of the rail network operator in the UK, apparently triggered in part by a complete mismatch between the SLAs imposed on the track repair subcontractors and the SLAs and legitimate safety expectations of the train companies [3]. Proper management and monitoring provides a

strong mitigation of this type of risk, since the operations management level allows business managers to check the correctness, consistency and adequacy of the mappings between the input and output service operations and aggregate services in a service composition.

Another aim of ESOA's service management layer is to provide support for open service marketplaces. Currently, there exist several vertical industry marketplaces, such as those for the semiconductor, automotive, travel, and financial services industries. Open service marketplaces operate much in the same way like vertical marketplaces, however, they are open. Their purpose is to create opportunities for buyers and sellers to meet and conduct business electronically, or aggregate service supply/demand by offering added value services and grouping buying power (just like a co-op). The scope of such a service marketplace would be limited only by the ability of enterprises to make their offerings visible to other enterprises and establish industry specific protocols by which to conduct business. Open service marketplaces typically support supply chain management by providing to their members a unified view of products and services, standard business terminology, and detailed business process descriptions. In addition, service marketplaces must offer a comprehensive range of services supporting industry-trade, including services that provide business transaction negotiation and facilitation, financial settlement, service certification and quality assurance, rating services, service metrics such as number of current service requesters, average turn around time, and manage the negotiation and enforcement of SLAs. ESOA's service management layer includes market management functionality (as illustrated in Figure-3) that is aimed to support these marketplace functions. The marketplace is created and maintained by a *market maker* (a consortium of organizations) that brings the suppliers and vendors together. The market maker assumes the responsibility of marketplace administration and performs maintenance tasks to ensure the administration is open for business and, in general, provides facilities for the design and delivery of an integrated service that meets specific business needs and conforms to industry standards.

The ESOA service management functions can rely on grid computing as it targets manageability. One of the aims of grid computing is the ability to manage ever-growing and ever more complex networks without overheads. The grid service domain architecture is a high level abstraction model that describes the common behaviors, attributes, and operations and interfaces to allow a collection of services to function as an integral unit and collaborate with others in a fully distributed, heterogeneous, grid-enabled environment. Service grids constitute a key component of the distributed services management as the scope of services expands beyond the boundaries of a single enterprise to encompass a broad range of business partners, as is the case in open marketplaces. For this purpose *grid services* can be used to provide the functionality of the ESOA's service management layer [6], [13].

The principal strengths of web and grid services are complementary, with web services focusing on platform-neutral description, discovery and invocation, and grid services focusing on the dynamic discovery and efficient use of distributed computational resources. This complementarity of Web and Grid Services has given rise to the proposed Open Grid Services Architecture (OGSA) [6] [13], which makes the functionality of grid services available through web service interfaces. Grid services are stateful services that provide a set of well-defined interfaces and follow specific conventions to facilitate coordinating and managing collections of web service providers/aggregators. The grid service indicates how a client can interact with it and is defined in WSDL. The state of the service is exposed to its clients as a standard interface that addresses web service filtering, discovery, routing, aggregation, selection, data and context sharing, notification and life-time management.

5 The service grid bus

Grid services used in the ESOA's service management layer to provide an enabling infrastructure for systems and applications that require the integration and management of services with the context of dynamic virtual marketplaces. Grid services provide the possibility to achieve end-to-end qualities of service and address critical application and system management concerns. To this end grid services provide the grid infrastructure over which services interact, aggregate, and coordinate through a distinct architectural tier. This infrastructure is called the *service grid bus*. The service grid bus (SGB) provides a high-level abstraction architecture and management facilities to allow services (within an open service marketplace) to function as an integral unit and collaborate with other services. The SGB architecture provides facilities for registration, discovery, selection/routing, business rules, filtering, routing, aggregation, fail-over, and topological mapping of service instances. The business rules govern the SGB's automatic processing for incoming service requests over aggregated service instances.

The service bus is a logical construct that cares very little where or on what platform a service provider runs. A user interface, designed to exactly match the business requirements may consume the services provided by the service bus, leaving the enterprise a free hand to choose the most efficient service provider.

An SGB is designed to provide a single service connectivity and a management tier that addresses the following concerns:

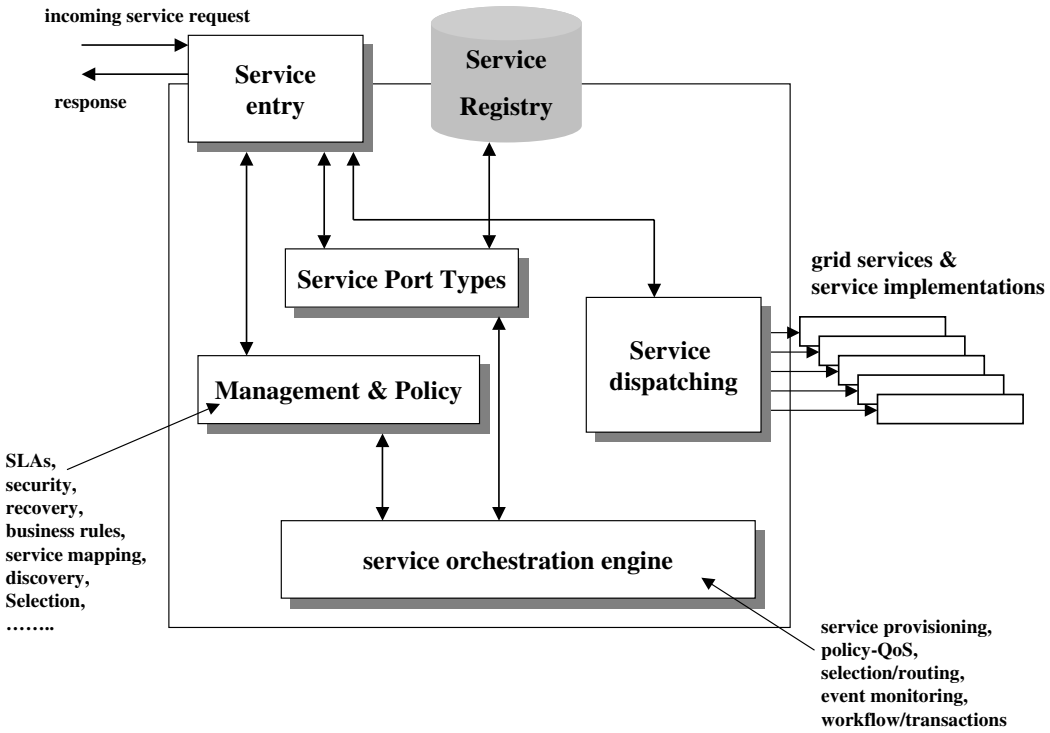

Figure 4. The service grid bus.

- Reach and robustness: The SGB allows to locate services anywhere in an open service marketplace and insulate them from connection failures, errors, and barriers such as firewalls, proxies, and caches. It guarantees that each service always sees an errorfree connection to any other service. The application as a whole should be robust under transient or long-term failure of one or more service components.

- Policy and security management: Services need to describe their capabilities and requirements to their environment and potential users. A collection of capabilities and requirements is referred to as a policy [8]. A policy may express such diverse characteristics as service transactional capabilities, security, response time, pricing, etc. For example, a policy of a service may specify that all interactions must be invoked under transaction protection, that incoming messages have to be encrypted, that outgoing messages will be signed, that responses may only be accepted within a specific time interval, etc. Finally, services must be restricted to authorized producers and consumers. The SGB enforces the security policy uniformly and universally across the entire marketplace.

- Development time and cost: The SGB provides the facilities that allow services to be easily aggregated into composite, higher-level services that match the factoring of an application. New services and clients will be added throughout the lifetime of the application. That process must be managed quickly, efficiently and cost effectively.

- Scalability and performance: The SGB must be able to perform well despite slow components and long, unpredictable network latencies.

The SGB lets service components interact over any network connection, handling all network errors, barriers, and transient or extended off-line conditions. It provides support for a business transaction model and support mechanisms for advanced transactional behavior of complex service-oriented business processes that span organizations [12]. The transaction model allows expressing unconventional atomicity criteria, e.g., payment atomicity, conversation atomicity, contract atomicity, and possesses the ability to express collaborative agreements and business conversation sequences that rely on transactional support. The model relies on a phased approach to business transactions so that all exchange of information between partners on the terms they could commit to, e.g., to fix price and quantity, are kept outside the ”pure” transaction protocol. This results

in enhancing flexibility and reducing latency and expensive transaction compensations and rollbacks in business interactions. The SGB's asynchronous communications makes the application robust under transient service node failures, and allows transparent rerouting in the event of prolonged failure.

The SGB provides standard, high-level services for security, management, service interaction, and aggregation, greatly accelerating application development and deployment. The SGB should also be consistent with emerging Web services workflow and transaction standards such as Web Services Transactions [1], Web Services Coordination [2] and the Business Transaction Protocol (BTP) [9]. Finally, the SGB supports linear scalability and high performance by offering native support for asynchronous interaction, and allowing service endpoints to be managed for load balancing, workload distribution, and fail-over. The SGB itself must be scalable and capable of supporting peak loads.

Figure-4 describes a service sharing and aggregation SGB model for open service marketplaces. The SGB model allows services and the resources they use to be more easily shared by different constituencies within an open service marketplace. With its service grid foundation an open service marketplace provides the notion of a *business service grid* that automatically dispatches the best service available from a pool of dynamically assembled service providers in order to meet the user's need. Selection of a service is not just based on availability, but can also be based on QoS characteristics, as specified in SLAs and business arrangements. Selection of a service is performed automatically based on service policy, and the features provided by the SGB enable service providers to conceal the implementation complexity required to handle multiple client requests over heterogenous environments.

The SGB invokes a service aggregation module to maintain its policy and states while serving external service requests. This module also performs selection and dispatching to find a service instance to execute a service request. The service aggregation module is also decomposed into modular functional units so that it is customizable to meet the special needs of diverse platforms, operations logic and so on.

One of the major benefits of introducing a distinct integration tier in the form of the SGB to implement the ESOA's service management layer is the ability to couple in value-added services that provide packaged solutions for common development needs. The SGB provides an asynchronous interaction model that lets users and applications initiate and monitor multiple tasks and data feeds simultaneously, and immediately alerts them when a transaction completes or a critical event occurs. In this regard, the SGB allows dynamic data feeds and transaction events to be delivered to the users or applications immediately when they occur. The SGB delivers to users applications that let them view multiple key performance indicators in real-time, collaborate with other users, and take action when critical events take place.

6 Conclusions

In this paper we introduced the concepts behind Service Oriented Computing and explained how the basic Service Oriented Architecture helps deliver service-based applications. We argued that in order to provide the advanced functionality needed to deliver sophisticated e-business applications an Extended Service Oriented Architecture is necessary. This architecture includes a service composition tier to offer necessary roles and functionality for the consolidation of multiple services into a single composite service. It also provides a tier for service operations management that can be used to monitor the correctness and overall functionality of aggregated/orchesteated services and support for open service marketplaces. Finally, we explained how grid services can be used to implement the service management tier of the Extended Service Oriented Architecture by means of the service grid bus.

References

[1] F. Cabrera et al, "Web Services Coordination (WS-Coordination)", August 2002, http://www.ibm.com/developerworks/library/ws-coor/.

[2] F. Cabrera et al, Web Services Transaction (WS-Transaction), August 2002, http://www.ibm.com/developerworks/library/ws-transpec/.

[3] CBDI Journal Modeling for SOA www.cbdiforum.com/, Feb. 2002.

[4] M. Champion, C. Ferris, E. Newcomer, and D. Orchard Web Services Architecture W3C Working Draft, www.w3.org/TR/ws-arch/, Nov. 2002.

[5] F. Curbera, Y. Goland, J. Klein, F. Leyman, D. Roller, S. Thatte, and S. Weerawarana Business Process Execution Language for Web Services(BPEL4WS) 1.0," August 2002, http://www.ibm.com/developerworks/library/ws-bpel.

[6] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid Services for Distributed System Integration. IEEE Computer, 35(6), 2002.

[7] J. Goepfert, M. Whalen An Evolutionary View of Software as a Service. IDC White paper, www.idc.com, 2002.

[8] F. Leymann Web Services: Distributed Applications without Limits - An Outline Procs. Of Database Systems for Business, Technology and Web, 2003.

[9] OASIS Committee Specification Business Transaction Protocol, version 1.0, May 2002.

[10] M. P. Papazoglou, J. Yang Design Methodology for Web Services and Business Processes Procs. of the 3rd VLDB-TES Workshop, Hong-Kong, 2002.

[11] M. P. Papazoglou, D. Georgakopoulos Service Oriented Computing Communications of the ACM, October 2003.

[12] M. P. Papazoglou Web Services and Business Transactions World Wide Web Journal, vol. 6, March 2003.

[13] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, and C. Kesselman Grid Service Specification. Technical report, Open Grid Service Infrastructure WG, Global Grid Forum, 2002. Draft 5, November 5, 2002.

[14] UDDI.org *UDDI Technical White paper*, $http://www.uddi.org/$, 2001

[15] Web Service Definition Language. http://www.w3.org/TR/wsdl.