International Journal of Cooperative Information Systems
Vol. 17, No. 2 (2008) 223–255

SERVICE-ORIENTED COMPUTING: A RESEARCH ROADMAP

MICHAEL P. PAPAZOGLOU
Tilburg University, INFOLAB/CRISM
Tilburg, The Netherlands
mikep@uvt.nl

PAOLO TRAVERSO
Istituto per la Ricerca Scientifica e Tecnologica (IRST)
Trento, Italy
traverso@itc.it

SCHAHRAM DUSTDAR
Technical University of Vienna
Information Systems Institute
Vienna, Austria
dustdar@infosys.tuwien.ac.at

FRANK LEYMANN
University of Stuttgart
Institute of Architecture of Application Systems
Stuttgart, Germany
Frank.Leymann@informatik.uni-stuttgart.de

Service-Oriented Computing (SOC) is a new computing paradigm that utilizes services as the basic constructs to support the development of rapid, low-cost and easy composition of distributed applications even in heterogeneous environments. The promise of Service-Oriented Computing is a world of cooperating services where application components are assembled with little effort into a network of services that can be loosely coupled to create flexible dynamic business processes and agile applications that may span organizations and computing platforms. The subject of Service-Oriented Computing is vast and enormously complex, spanning many concepts and technologies that find their origins in diverse disciplines that are woven together in an intricate manner. In addition, there is a need to merge technology with an understanding of business processes and organizational structures, a combination of recognizing an enterprise's pain points and the potential solutions that can be applied to correct them. The material in research spans an immense and diverse spectrum of literature, in origin and in character. As a result research activities are very fragmented. This necessitates that a broader vision and perspective be established — one that permeates and transforms the fundamental requirements of complex applications that require the use of the Service-Oriented Computing paradigm.

This paper provides a Service Oriented Computing Roadmap and places on-going research activities and projects in the broader context of this roadmap. This research roadmap launches four pivotal, inherently related, research themes to Service-Oriented

Computing: service foundations, service composition, service management and monitoring and service-oriented engineering.

Keywords: Service-oriented computing; service-oriented architecture; business process; service composition; service management.

1. Overview

Today's business climate demands a high rate of change with which Information Technology (IT)-minded organizations are required to cope. Organizations face rapidly changing market conditions, new competitive pressures, new regulatory fiats that demand compliance, and new competitive threats. All of these situations and more drive the need for the IT infrastructure of an organization to respond quickly in support of new business models and requirements. Only in this way can an organization gear towards the real world of fully automated, complex electronic transactions. As most enterprise applications were not designed to enable rapid adaptation of application functionality, this adds another level of intricacy to an already complex IT landscape. At the same time, it increases infrastructure complexity and limits its ability to quickly change application features or functions.

Integration and infrastructure management are the key elements of an on demand operating IT environment. Integration enables the efficient and flexible combination of resources to optimize operations across and beyond the boundaries of an organization and enables them to interoperate seamlessly. Integration is about seamlessly interlinking on one hand people and on the other hand processes and information that may transcend organizational boundaries despite the existence of multiple — and possibly heterogeneous — platforms and protocols, and numerous access devices, while leveraging the potential of the Internet. Infrastructure management address two objectives: automation and virtualization of the environment. Automation of the environment is achieved by the capability to reduce management complexity to enable better use of assets, improve availability and resiliency, and reduce costs based on business policy and objectives. Virtualization of the environment is achieved by the capability to provide easy access to and a single consolidated view of all available resources in a network — no matter where the resources or information reside. Service orientation provides the underlying implementation that can make an on demand IT operating environment a reality by supporting the functions of both integration and infrastructure management.[1]

Service-Oriented Computing (SOC) utilizes services as the constructs to support the development of rapid, low-cost and easy composition of distributed applications. Services are autonomous, platform-independent computational entities that can be used in a platform independent way. Services can be described, published, discovered, and dynamically assembled for developing massively distributed, interoperable, evolvable systems. Services perform functions that can range from answering simple requests to executing sophisticated business processes requiring peer-to-peer

relationships between possibly multiple layers of service consumers and providers. Any piece of code and any application component deployed on a system can be reused and transformed into a network-available service. Services reflect a "service-oriented" approach to programming, based on the idea of composing applications by discovering and invoking network-available services rather than building new applications or by invoking available applications to accomplish some task.[2] Services are most often built in a way that is independent of the context in which they are used. This means that the service provider and the consumers are loosely coupled.

This "service-oriented" approach is independent of specific programming languages or operating systems. It allows organizations to expose their core competencies programmatically over the Internet or a variety of networks, e.g. cable, UMTS, XDSL, Bluetooth, etc. using standard (XML-based) languages and protocols, and implementing a self-describing interface. Web Services are the current most promising technology based on the concept of Service Oriented Computing.[3] Web services provide the basis for the development and execution of business processes that are distributed over the network and available via standard interfaces and protocols. Web services may use the Internet as the communication medium (as well as other transport protocols) and open Internet-based standards, such as the Simple Object Access Protocol (SOAP) as transmission medium, the Web Services Description Language (WSDL) for service definition and the Business Process Execution Language (BPEL) for orchestrating services.

The visionary promise of services technologies is a world of cooperating services where application components are assembled with little effort into a network of services that can be loosely coupled to create dynamic business processes and agile applications that span organizations and computing platforms.[4] Services hold the promise of moving beyond the simple exchange of information — the dominating mechanism for application integration today — to the concept of accessing, programming and integrating application services that are encapsulated within old and new applications. An important economic benefit of the Service Oriented Computing paradigm is that it enables application developers to dynamically grow application portfolios more quickly than ever before, by creating compound application solutions that use internally existing organizational software assets which they appropriately combine with external components possibly residing in remote networks. Previously isolated Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), Supply Chain Management (SCM), Human Resource Management (HRM), financial, and other legacy systems can now be converted to service enabled architectures and integrated more effectively than when relying on custom, point-to-point coding or proprietary Enterprise Application Integration technology. The end result is that it is then easier to create new composite applications that use pieces of application logic and/or data that reside in the existing systems. This represents a fundamental change to the socio-economic fabric of the software developer community that improves the effectiveness and productivity in

software development activities and enables enterprises to bring new products and services to the market more rapidly.[5]

Key to this concept is the service-oriented architecture (SOA). SOA is a logical way of designing a software system to provide services to either end-user applications or to other services distributed in a network, via published and discoverable interfaces. A well-constructed, standards-based Service Oriented Architecture can empower a business environment with a flexible infrastructure and processing environment. SOA achieves this by provisioning independent, reusable automated business process and systems functions as services and providing a robust and secure foundation for leveraging these services. Efficiencies in the design, implementation, and operation of SOA-based systems can allow organizations to adapt far more readily to a changing environment.

Services technologies are being shaped by, and increasingly will help shape, modern society as a whole, especially in vital areas such as dynamic business, health, education and government services. Applying services technologies leads to reduced complexity and costs, exposing and reusing core business functionality, increased flexibility, resilience to technology shifts and improving operational efficiency. For all these reasons, it is expected that the Service Oriented Computing paradigm will exhibit a steeper adoption curve, as it solves expensive and intractable business and technology problems, and will infiltrate more of the applications portfolio, than previous application technologies.

In this paper, we first motivate the need for a Service-Oriented Computing Roadmap and subsequently place on-going research activities and projects in the broader context of this roadmap. This research roadmap launches four pivotal, inherently related, research themes to Service-Oriented Computing: service foundations, service composition, service management and monitoring and service-oriented engineering. Each theme is introduced briefly from a technology, state of the art and scientific challenges standpoint. From the technology standpoint a comprehensive review of the state of the art, standards, and current research activities in each key area is provided. From the state of the art, the major open problems and bottlenecks to progress are identified. Several of these obstacles arise due to the current lack of interdisciplinary research in the field, which is considered to be a major impediment that limits added economic growth through deployment and use of services technology. Finally, the scientific challenges that tackle the found obstacles are formulated. These are long-term visions that serve as integration platforms and demonstrators for an holistic approach to Service-Oriented Computing in the identified key areas.

2. Need for a Services Research Roadmap

As Service-Oriented Computing is very much an emerging field, there is no such thing as a "general audience" for Service-Oriented Computing — there are many people (researchers and practitioners) with many different (and probably conflicting) levels of understanding and uses for Service-Oriented Computing.

The subject of Service-Oriented Computing is vast and enormously complex, spanning many concepts, protocols and technologies that find their origins in disciplines such as distributed computing systems, computer networking, computer architectures and middleware, grid computing, software engineering, programming languages, database systems, security, artificial intelligence and knowledge representation that are interwoven in an intricate manner. In addition there is a need to merge technology with an understanding of business processes and organization structures, a combination of recognizing an enterprise's pain points and the potential solutions that can be applied to correct them. The material in research spans an immense and diverse spectrum of literature, in origin and in character. As a result research activities are very fragmented and do not contribute to a mutually acceptable, joint research agenda.

Only through adaptation of an *holistic approach* to Service-Oriented Computing research it is considered likely that new industries and economic growth factors can be provided. Thus to unleash the full potential of SOC research, a broader vision and perspective is required — one that permeates and transforms the fundamental requirements of complex applications that require the use of the Service-Oriented Computing paradigm. This will further enhance the value proposition of SOC and will facilitate the formulation of a Services Research Roadmap leading to more effective and clearly inter-related solutions and better exploitation of research results.

Purpose of the Services Research Roadmap is to facilitate efficient and effective use of research funds by consolidating, streamlining and strategically inter-relating the current research results and agenda and prioritizing attention to gaps, encouraging interdisciplinary research that might otherwise be overlooked, and coordinating existing and future research work and projects.

3. Research Roadmap for Service-Oriented Computing Research

Services Research Roadmap introduces a stratified logical service-based architecture (known as extended Service-Oriented Architecture[2,6]) to create a reactive and adaptive IT environment. This environment has the ability to represent detailed business policies and rules abstracted from fixed functional and operational capabilities and delivering these abstracted capabilities in the form of customizable service-oriented solutions. Its objective is to provide facilities for ensuring consistency across the organization, high availability of services, security of non-public services and information, orchestration of multiple services as part of mission-critical composite applications — all essential requirements for business-quality services. Thus, it strives to improve systems and business visibility and provide greater control and flexibility in defining and adjusting business rules and parameters.

The architectural layers in the Services Research Roadmap, which are depicted in Fig. 1, describe a logical separation of functionality in such a way that each layer defines a set of constructs, roles and responsibilities and leans on constructs of its preceding layer to accomplish its mission. The logical separation of functionality

is based on the need to separate basic service capabilities provided by a services middleware infrastructure and conventional SOA from more advanced service functionality needed for dynamically composing (integrating) services and the need to distinguish between the functionality for composing services from that of the management of services and their underlying infrastructure.

As shown in Fig. 1, the Services Research Roadmap has three planes with the bottom plane utilizing the basic service middleware and architectural constructs and functionality for describing, publishing and discovering services, while the service composition and management planes are layered on top of it. The perpendicular axis indicates service characteristics that cut across all three planes. These include semantics, non-functional service properties and Quality of Service (QoS). As cross cutting concerns permeate all three planes (see Fig. 1) we shall introduce them briefly below before focussing on the Services Research Roadmap planes in the following subsections.

Quality of Service encompasses important functional and non-functional service quality attributes, such as performance metrics (response time, for instance), security attributes, (transactional) integrity, reliability, scalability, and availability. Delivering QoS on the Internet is a critical and significant challenge because of its dynamic and unpredictable nature. Applications with very different characteristics and requirements compete for all kinds of network resources. Changes in traffic patterns, securing mission critical business transactions and the effects

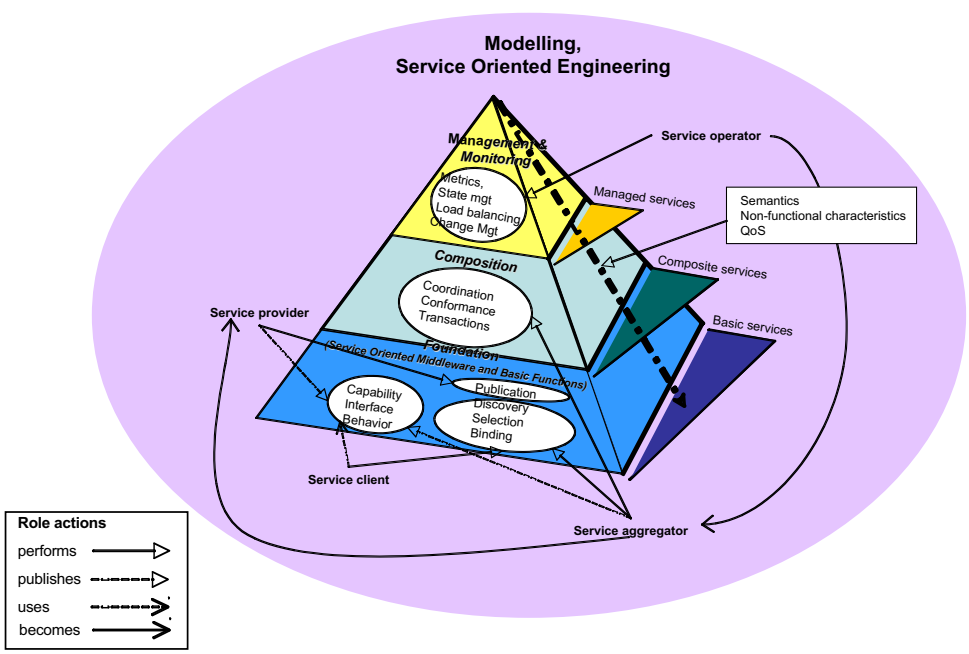

Fig. 1. The research planes in the Services Research Roadmap.

of infrastructure failures, low performance of Web protocols and reliability issues over the Web create a need for Internet QoS standards. Often, unresolved QoS issues cause critical transactional applications to suffer from unacceptable levels of performance degradation. Traditionally, QoS is measured by the degree to which applications, systems, networks, and all other elements of the IT infrastructure support availability of services at a required level of performance under all access and load conditions. While traditional QoS metrics apply, the characteristics of Web services environments bring both greater availability of applications and increased complexity in terms of accessing and managing services and thus impose specific and intense demands on organizations, which QoS must address.

For Web services to interact properly with each other as part of composite applications that perform more complex functions by orchestrating numerous services and pieces of information, the requester and provider entities must agree both on the service description (WSDL definition) and semantics that will govern the interaction between them. A complete semantic solution requires that semantics are addressed not only at the terminology level but also at the level that Web services are used and applied in the context of business scenarios, viz. at the business process-level. This implies that there must be agreement between a service requester and provider as to the implied processing of messages exchanged between interacting services that are part of a business process.

Finally, Fig. 1 illustrates that service modeling and service oriented engineering, i.e. service-oriented analysis, design and development techniques and methodologies, are crucial elements for the developing meaningful services and business process specifications and an important requirement for SOA applications that leverage Web services. Service-oriented engineering activities help develop meaningful services, service compositions and techniques for managing services. In other words they apply to the three service planes shown in Fig. 1.

The Services Research Roadmap in Fig. 1 is also shown to define several roles. In addition to the classical roles of service client and provider, it also defines the roles of service aggregators and service operator (which will be explained in the material that follows).

In the following, we shall concentrate on the individual planes found in the Services Research Roadmap, motivate and explain them, introduce current standards, state of the art as well as characteristic research activities within each plane, highlight open problems and describe major emerging trends and opportunities, identify relevant technological driving forces and finally concentrate on proposing a number of challenging research activities for the near future. Table 1 summarizes our findings regarding the state of the art and grand challenges in Services Research.

3.1. *Service foundations*

The bottom plane in the Services Research Roadmap is the service foundations plane that provides a service oriented middleware backbone that realizes the

Table 1. Overview of state of the art and grand challenges in Services Research.

	State of the Art	Grand Challenges
Service Foundations	Enterprise Service Bus: — Open standards message backbone — Implementation, deployment, management — Set of infrastructure capabilities implemented by middleware technology — Implementation backbone for SOA	— Dynamically (re)-configurable run-time architectures — Dynamic connectivity capabilities — Topic & content-based routing — End-to-end security solutions — Infrastructure support for process, information & application integration — Advanced service discovery mechanisms
Service Composition	Service orchestration — Service interaction at message-level — Point to point compositions from perspective & control of a single endpoint — Executable business processes	— Composability analysis for repleceability, compatibility & conformance — Autonomic composition of services — QoS aware service composition — Business-driven composition — Composition of resources, humans & organisations in the form of services
Service Management	— Web Services Distributed Management (WSDM) — Management Using Web Services (MUWS) — Management of Web Services (MOWS)	— Self-configuring services — Self-healing services — Self-optimizing services — Self-protecting services
Service Engineering (Service Design & Development)	— Porting of existing components using wrappers — Component-based development	— Design principles for engineering service applications — Flexible gap analysis techniques — Service governance techniques — Design principles for service adaptation

runtime SOA infrastructure that connects heterogeneous components and systems, and provides multiple-channel access to services, e.g. via mobile devices, hand held devices, over variety of networks including the Internet, cable, UMTS, XDSL, Bluetooth, and so on. This runtime infrastructure allows defining basic interactions involving the description, publishing, finding and binding of services.

In a typical service-based scenario employing the service foundations plane a service provider hosts a network accessible software module (an 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 client (requestor) discovers a service (endpoint) and retrieves the service description directly from the service (through meta-data exchange) or from a registry or repository (like UDDI); it uses the service description to bind with the service provider and to invoke the service or to interact with the service. Service provider and service client roles are logical constructs and a service may exhibit characteristics of both. In Fig. 1, service aggregators group services that are provided by other service providers into a distinct value-added service and may themselves act as service providers. For reasons of conceptual simplicity, in Fig. 1, we assume that service clients, providers and aggregators can act as service brokers or service discovery agencies and publish the services they deploy. The role actions in this figure also indicate that a service aggregator can become (or rather is a special type of) provider.

3.1.1. *State of the art*

The requirements to provide an appropriately capable and manageable integration infrastructure for Web services and SOA are coalescing into the concept of the Enterprise Service Bus (ESB). There are two key ideas behind this approach[7]: loosely couple the systems taking part in the integration and break up the integration logic into distinct easily manageable pieces.

The Enterprise Service Bus is an open-standards based message backbone designed to enable the implementation, deployment, and management of SOA-based solutions. An ESB is a set of infrastructure capabilities implemented by middleware technology that enable an SOA and alleviate disparity problems between applications running on heterogeneous platforms and using diverse data formats. It supports service, message, and event-based interactions with appropriate service levels and manageability. In other words, the ESB provides the distributed processing, standards-based integration, and enterprise-class backbone required by the extended enterprise. The ESB is designed to provide interoperability between larger grained applications and other components via standards-based adapters and interfaces. The bus functions as both transport and transformation facilitator to allow distribution of these services over disparate systems and computing environments.

Conceptually, the ESB has evolved from the store-and-forward mechanism found in middleware products and now is a combination of Enterprise Application Integration, e.g. application servers and integration brokers, Web services, XSLT, and orchestration technologies.[8] An ESB provides an implementation backbone for an SOA that treats applications as services. It establishes proper control of messaging as well as applies the needs of security, policy, reliability and accounting, in an SOA.

One model that is emerging as appropriate and successful for the ESB is the *container model*. In this model, there is a "container" for the service implementation

taking care of exposing the service functionalities and non-functional properties to the external world via the network. The basic (core) functions of the container are as follows:

- establishing connectivity and Message Exchange Patterns (MEPs)
- providing support and provision facilities such as transactions, security, performance metrics, etc., in a declarative and composable manner,
- providing support for dynamic configuration,
- monitoring of internal behavior and state to management systems (services)
- performing data and protocol adaptation,
- providing support for services discovery.

Figure 2 shows a simplified view of an ESB that integrates a J2EE application using JMS, a .NET application using a C# client, an MQ application that interfaces with legacy applications, as well as external applications and data sources using Web services. An ESB, as represented in Fig. 2, enables the more efficient value-added integration of a number of different application components, by positioning them behind a service-oriented façade and by applying Web services technology to the problem. In Fig. 2, a distributed query engine, which is normally based on XQuery or SQL, enables the creation of data services to abstract the complexity of underlying data sources. As shown in Fig. 2, a primary use case for ESB is to act as the intermediary layer between a portal server and the backend data sources that the portal server needs to interact with. A portal in Fig. 2 is a user-facing aggregation point of a variety resources represented as services, e.g. retail, divisional, corporate employee, and business partner portals.

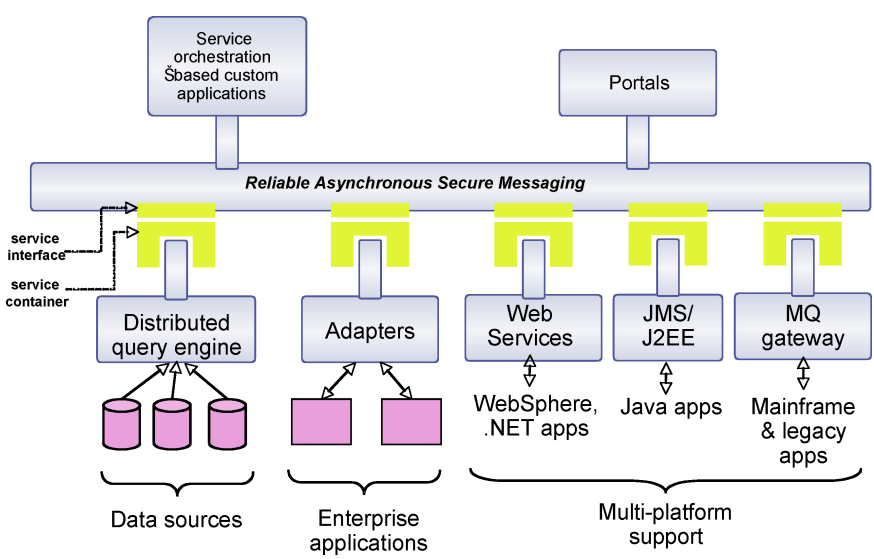

Fig. 2. Enterprise service bus connecting diverse applications and technologies.

To design effective SOA solutions it is desirable that service developers build self-configuring service architectures that can use distributed components to dynamically create an optimal service-based architectural run-time style according to particular application requirements and existing system characteristics. To this end, the grid services community has attempted to address the support of dynamically reconfigurable service architectures by directing research efforts into two categories of problems. Service specific architectures that are designed for particular classes of services/applications.[9–11] Examples include resource selection for resource-intensive applications and resource allocation for services consisting of a set of multi-fidelity applications. Others proposed generic architectures that can compose different services using "type-based architectural composition".[12,13] Components have well-defined input/output (requires/provides) interfaces, so a service composition module can automatically generate a service configuration providing the requested interface(s) — all in all, the overall direction to compose applications from services is accepted in this domain too.[5]

Other research activities in the services foundation layer to date have targeted mostly formal service description language(s) for enhanced service definitions addressing, besides functional aspects, also behavioral as well as non-functional aspects associated with services.[14–16] Research activities have also concentrated on providing an open, modular, extensible framework for service discovery, publication and notification mechanisms across distributed, heterogeneous, dynamic (virtual) organizations as well as unified discovery interfaces and query languages for multiple pathways.[17–20]

The AI and semantic Web community has concentrated their efforts in giving richer semantic descriptions of Web services that describe the properties and capabilities of Web services in an computer-interpretable form.[21] Such activities target the use of formal languages for semantically describing all relevant aspects of Web services in order to facilitate the automated discovery, combination and invocation of services over the Web.[22,23]

3.1.2. *Grand challenges*

Major research challenges for the near future include:

- *Dynamically (re-)configurable run-time architectures*: The run-time service infrastructure should be able to configure itself and be optimized automatically in accordance with specific application requirements and high-level policies — representing business-level objectives, for example — that specify what is desired (such as particular security and privacy requirements) and not how it is to be accomplished. A self-reconfiguring services architecture can automatically leverage distributed service components and resources to create an optimal architectural configuration according to both the requirements of a particular user and application characteristics. For instance, the run-time environment must posses the critical ability to route service interactions through a variety of protocols, and

to transform from one protocol to another where necessary. Another important aspect is the ability to support diverse service messaging models consistent with the SOA interfaces, and capable of transmitting and homogenizing the required interaction context, such as security, transaction, or message correlation information. In particular, the run-time environment should be able to configure itself in accordance with an extensible set of QoS properties and policies for security, for transactional behavior, and so on.

- *Dynamic connectivity capabilities*: Dynamic connectivity is the ability to connect to Web services dynamically without using a separate static API or proxy for each service. Service-based applications today operate on a static connectivity mode, requiring some static piece of code for each service. Dynamic service connectivity is a key capability for a successful run-time environment. The dynamic connectivity API is the same regardless of the service implementation protocol (Web services, JMS, EJB/RMI, etc.).
- *Topic and content-based routing capabilities*: The run-time service infrastructure should be equipped with routing mechanisms to facilitate not only topic-based routing but also, more sophisticated, content-based routing. Topic-based routing assumes that messages can be grouped into fixed, topical classes, so that subscribers can explicate interest in a topic and as a consequence receive messages associated to that topic while content-based routing on the other hand, allows subscriptions on constraints of actual properties (attributes) of business events. Content-based routing forwards messages to their destination based on the context or content of the service.
- *End-to-end security solutions*: Validating the security aspects in SOA-based applications requires a full system approach to test end-to-end security solutions from both network level and application level security angles. This requires the development of a set of services security technologies can create a unifying approach for dealing with protection for messages exchanged in service-based environments. Their purpose is to construct authentication, authorization, auditing, privacy and trust, as well as higher-level key exchange mechanisms, while providing integrating abstraction framework allowing systems and applications to surmount different security systems and technologies. Similar considerations can be found in the WS-Roadmap jointly developed by IBM and Microsoft.[24]
- *Infrastructure support for application integration*: The run-time environment should possess the ability to support service-based application integration by enabling better-structured integration solutions that deliver applications comprised of interchangeable parts, evolutionary application portfolios that protect investment and can respond rapidly to new requirements and business processes and facilitate "best of breed" portfolio strategies which automatically combine legacy applications, acquired packages, external application subscriptions and newly built components.
- *Infrastructure support for data integration*: The run-time environment should possess the ability to provide consistent access to all the data by all the applications

that require it, in whatever form they need it, without being restricted by the format, source, or location of the data. This requirement might involve self-configurable adapters and transformation facilities, aggregation services to merge and reconcile disparate data, e.g. merging two customer profiles, and validation to ensure data consistency, e.g. minimum income should be equal to or exceed a certain threshold.

- *Infrastructure support for process integration*: The run-time environment should possess the ability to provide automated facilities that provide solutions for business processes, integration of applications into processes, and integrating processes with other processes. Process-level integration may include the integration of business processes and applications within the enterprise context (viz. Enterprise Application Integration solutions) and should also support the integration of end-to-end processes involving external sources, such as supply chain management or financial services that span multiple institutions (viz. e-Business integration solutions). The service level provides the necessary infrastructure that enables effective process compositions (see Sec. 3.2).
- *Semantically enhanced service discovery*: The main challenge of service discovery is the use of automated means for accurate discovery of services in a manner that demands minimal user involvement. Improving service discovery requires explicating the semantics of both the service provider and the service requester. Improving service discovery involves adding semantic annotations and including descriptions of QoS characteristics (for example in DAML/OWL or other semantic markup languages) to service definitions in WSDL and then registering these descriptions in registries. The use of standard ontologies that support shared vocabularies and domain models for use in the service description also facilitates service discovery by making the semantics implied by structures in service descriptions explicit. To achieve automated discovery of services, the needs of service requesters have to be explicitly stated. We expect such needs to be expressed as goals, which correspond to the description of what services are sought, in some formal request language.

3.2. *Service composition*

The service composition plane in the Services Research Roadmap encompasses necessary roles and functionality for the aggregation of multiple services into a single composite service. Resulting composite services may be used by service as basic services in further service compositions or may be offered as complete applications/solutions to service clients. Service aggregators accomplish this task. Service aggregators thus become service providers by publishing the service descriptions of the composite service they create. Service aggregators develop specifications and/or code that permit the composite service to perform functions that are based on features such as meta-data descriptions, standard terminology and reference models and service conformance. Service aggregators perform service coordination to

control the execution of the composite services (viz. processes), services transactions and manage both the dataflow as well as the control flow between composite services. They also enforce policies on aggregate service invocations.

3.2.1. *State of the art*

The full potential of Web services as a means of developing dynamic e-Business solutions will only be realized when applications and business processes are able to integrate their complex interactions into composite added value services. Services technologies offer a viable solution to this problem since they support coordination and offer an asynchronous and message oriented way to communicate and interact with application logic. However, when looking at Web services, for example, it is important to differentiate between the baseline specifications of SOAP, UDDI and WSDL that provide the infrastructure that supports publishing, finding and binding operations in the service-oriented architecture and higher-level specifications required for e-Business integration. These higher-level specifications provide functionality that supports and leverages services and enables specifications for integrating automated business processes.

Currently, there are competing initiatives for developing business process definition specifications, which aim to define and manage business process activities and business interaction protocols comprising collaborating services. The terms "orchestration" and "choreography" have been widely used to describe business interaction protocols comprising collaborating services.

Orchestration describes how services can interact with each other at the message level, including the business logic and execution order of the interactions from the perspective and under control of a single endpoint. Orchestration refers to an executable business process that may result in a long-lived, transactional, multistep process model. With orchestration, the business process interactions are always controlled from the (private) perspective of one of the business parties involved in the process.

Choreography is typically associated with the public (globally visible) message exchanges, rules of interaction and agreements that occur between multiple business process endpoints, rather than a specific business process that is executed by a single party. Choreography is more collaborative in nature than orchestration. It is described from the perspectives of all parties (common view), and defines the complementary observable behavior between participants in business process collaboration. Choreography offers a means by which the rules of participation for collaboration can be clearly defined and agreed to, jointly. Choreography tracks the sequence of messages that may involve multiple parties and multiple sources, including customers, suppliers, and partners, where each party involved in the process describes the part they play in the interaction and no party "owns" the conversation.

Orchestration is targeted by a family of XML-based process standard definition languages most representative of which is the Business Process Execution Language

for Web Services.[25] Service choreography is targeted by Web Services Choreography Description Language (WS-CDL), which specifies the common observable behavior of all participants engaged in business collaboration. This sharp distinction between orchestration and choreography is rather artificial and the consensus is that they should both coalesce in the confines of a single language and environment.

On the research front, activities have mainly concentrated on dynamic compositions,[26] on modularizing compositions,[26,27] on enhancing service descriptions (with, for instance, compositional assertions) so that compositions can be assessed and formally verified[28] and on providing context aware services to enable compositions. In the AI field there has been some work in the area of applying AI planning techniques to automate the retrieval and composition of Web service[29–34] verification[35] and monitoring of service oriented applications[36] and so forth, but these efforts are still either at the specification-level or at very preliminary stage of development. Many of the existing approaches towards service composition largely neglect the context in which composition takes place. It is only recently that research approaches have focussed on developing context-aware methodologies that take into account the business and social context of service compositions as the basis for process specification and verification.[37]

3.2.2. *Grand challenges*

One of the major challenges for industry-wide adoption of the service-oriented approach is the automated composition of distributed business processes, i.e. the development of technology, methods and tools that support an effective, flexible, reliable, easy-to-use, low-cost, dynamic, time-efficient composition of electronic distributed business processes. Standards such BPEL and WS-CDL that operate at the service composition plane in the Services Research Roadmap provide the basis for the composition of services and the integration of business processes that are distributed among the most disparate entities, both within an organization (e.g. different departments) and across organizational borders (e.g. consumers interacting with different businesses or government departments providing complementary services).

However, so far, the automated composition of distributed business processes is still far from being achieved: no effective, easy-to-use, flexible support is provided that can cope with the life cycle of distributed business processes, with their inevitable evolution and required adaptation to changes in, e.g. business strategies and markets, customers and providers relationships, interactions, and so on. Service composition is today largely a static affair. All service interactions are anticipated in advance and there is a perfect match between output and input signatures and functionality. More *ad hoc* and dynamic service compositions are required very much in the spirit of lightweight and adaptive workflow methodologies. These methodologies will include advanced forms of coordination, instance-based modification of process models, less structured process models, and automated planning techniques as part of the integration/composition process. On the transactional front, although

standards like WS-Transaction, WS-Coordination and the Web Service Composite Application Framework (WS-CAF) are a step in the right direction, they fall short of describing different types of atomicity needs for e-Business and e-government applications. These do not distinguish between transaction phases and conversational sequences, e.g. negotiation. Another area that is lacking research results is advanced methodologies in support for the service composition lifecycle. Some of the major limitations of state-of-the-art technologies that prevent effective automated composition are:

(1) Lack of tools for supporting the evolution and adaptation of business processes. It is hard to define compositions of distributed business processes that work properly under all circumstances. Misunderstandings in the agreement between different organizations, as well as errors in the specification and implementation of the interaction protocols, easily occur, especially for complex processes and interaction protocols. Typical problems are business processes that wait forever, or for too long, to receive an answer from another process, or that expect a different answer; or, business processes that fail to invoke another process as required and do not allow the distributed business to correctly proceed. Moreover, even in the case business, interactions are initially correctly defined and implemented. They frequently stop working when some processes involved in the interactions are autonomously redefined by an external organization; this kind of evolution is very common in distributed, highly dynamic environments.
(2) Lack of integration of business requirements in the business process life cycle. While BPEL and WS-CDL are adequate for the specification of the detailed message exchanges in orchestrations and choreographies, there is the need for languages that define both the internal business needs of an organization and its requirements over external services, and for a systematic way of linking them to business processes. Indeed, without explicit requirements, it is not possible to motivate the choices that lead to the specification of a certain flow of activities within a business processes and of its interactions with other processes. Traceability, i.e. determining how a process is related to and affect business requirements and needs, cannot be supported if the two are not linked, which is of utmost importance in supporting legal requirements by IT.[38] Finally, and most importantly, if requirements are not accessible, there is no way to drive the automated composition of distributed business processes so that it could support the evolution and adaptation of the processes.

Some of the most notable research challenges for the near future include:

- *Composability analysis for replaceability, compatibility, and conformance for dynamic and adaptive processes*: Service conformance ensures the integrity of a composite service by matching its operations with those of its constituent component services, imposes semantic constraints on the component services (e.g. to ensure enforcement of business rules), and ensures that constraints on data exchanged by component services are satisfied. Service conformance comprises

both behavioral conformance as well as semantic conformance. The former guarantees that composite operations do not lead to spurious results and that the overall process behaves in a correct and unambiguous manner. The latter, by annotating services and operations with (possibly domain-specific) semantic descriptions, ensures that they preserve their meaning when they are composed and can be formally validated.

- *Adaptive and emergent service compositions*: Automated composition task is traditionally described as the problem of supporting the aggregation of component services that are available and published, e.g. in the Web, as they are. However, most often, when different organizations, e.g. companies, financial or public administration bodies, decide to cooperate, rarely their services or their business processes, can be aggregated without a change, an adaptation of the local services and processes. In such a distributed environment, with autonomous actors, knowledge exchange/sharing and process interoperability/composition cannot be often forced in a top-down manner. They emerge after a negotiation process where each actor has to deal with two opposite and driving forces: the "impedance" to change its business assets and the need to evolve towards common and shared assets. A typical example nowadays in Europe is the financial sector, where different banks have to merge and integrate their own business processes in order to be more competitive and offer better services to clients. Networks of small medium enterprises are a further significant example, where each company has its own local business needs and strategies, and at the same time it has the need to join in a network to be more competitive on the market. The result, the shared knowledge and the interoperable processes, the common models and objectives, are not known *a priori*. They emerge from a distributed negotiation process where each actor accepts to give up some of its assets, if this is compensated by some advantage due to being part of a network. This negotiation has to be human-driven, since it often concerns the strategic interests of the involved actors. All of this requires techniques and tools supporting the process of defining the emergent common model, processes and objectives and supporting the negotiation among actors.
- *Autonomic composition of services*: One of the main fundamental ideas of Service-Oriented Computing is that applications should be developed by composing services that are available, e.g. on the Web. Given some business level and strategic requirements for the composition, the idea is to automatically generate the electronic business process implementing it. In this framework, the challenge is the *autonomic composition of services*, e.g. service composition that are self-configuring, self-optimizing, self-healing, and self-adapting. Self-configuring compositions are, e.g. composite services that are capable of automatically discovering new partners to interact with, to automatically select among available suppliers, to choose among different options available for contracts, etc. Self-optimizing Web service compositions should automatically select partners and options that would, e.g. maximize benefits and reduce costs. Self-healing compositions should

be able to automatically detect that some business composition requirements are no longer satisfied by the implementation and react to requirement violations. Self-adapting service compositions should be able to function despite changes in behaviors of external composite services. They should reduce as much as possible the need of human intervention for adapting services to subsequent evolutions.

- *QoS-aware service compositions*: To be successful, service compositions need to be QoS-aware, i.e. understand and respect each other's policies, performance levels, security requirements, SLA stipulations, and so forth. For example, knowing that a new business process adopts a Web services security standard such as one from the stack of WS-Security specifications is not enough information to enable successful composition. The client needs to know if the services in the business process actually require WS-Security, what kind of security tokens they are capable of processing, and which one they prefer. Moreover, the client must determine if the service should communicate using signed messages. If so, it must determine what token type must be used for the digital signatures. Finally, the client must decide on when to encrypt the messages, which algorithm to use, and how to exchange a shared key with the service. For example, a purchase order service in an order management process may indicate that it only accepts username tokens that are based signed messaged using X.509 certificate that is cryptographically endorsed by a third party.
- *Business-driven automated compositions*: One of the main ideas of service oriented applications is to abstract away the logic at the business level from its non-business related aspects, the "system level", e.g. the implementation of transaction, security, and reliability policies. This abstraction should make easier and effective the composition of distributed business processes. However, the provision of automated composition techniques, which make this potential advantage real, is still an open problem. Business-driven automated compositions should exploit business and system level separation in service compositions. According to this view, service composition at the business level should pose the requirements and the boundaries for the automatic composition at the system level. While the service composition at the business level should be supported by user-centred and highly interactive techniques, system level service compositions should be fully automated and hidden to the humans. System level compositions should be QoS-aware, should be generated and monitored automatically, and should also be based on autonomic computing principles (see also challenges for service management and monitoring in Sec. 3.3.2).

3.3. *Service management and monitoring*

Managing loosely coupled applications in an SOA is an absolute requirement. Composite service developments necessitate the use of mechanisms that provide them insights into the health of systems that implement Web services and into the status and behavior patterns of loosely coupled applications. Failure or change of a single application component can bring down numerous interdependent enterprise

applications. The addition of new applications or components can overload existing components, causing unexpected degradation or failure of seemingly unrelated systems. Application performance depends on the combined performance of cooperating components and their interactions. To counter such situations, enterprises need to constantly monitor the health of their applications. The performance should be in tune, at all times and under all load conditions. A consistent management and monitoring infrastructure is thus essential for production-quality Web services and applications and provided by the management and monitoring plane in Services Research Roadmap. The rationale is very similar to the situation in traditional distributed computing environments, where systems administrators rely on programs/tools/utilities to make certain that a distributed computing environment operates reliably and efficiently.

The management and monitoring in Services Research Roadmap requires that a critical characteristic be realized: that services be managed and monitored. Service management encompasses the control and monitoring of SOA-based applications throughout their life cycle. Service management spans a range of activities from installation and configuration to collecting metrics and tuning to ensure responsive service execution. It includes many interrelated functions such as Service-Level Agreement negotiation, management, auditing, monitoring, and troubleshooting, service lifecycle/state management, performance management, services and resources provisioning, and includes aspects like scalability, availability and extensibility and others.

3.3.1. *State of the art*

Service operations management typically gathers information about the managed service platform, services and business processes and managed resource status and performance, and supporting specific management tasks (e.g. root cause failure analysis, SLA monitoring and reporting, service deployment, and life cycle management and capacity planning). Operations management functionality may provide detailed application performance statistics that support assessment of the application effectiveness, permit complete visibility into individual business processes and transactions, guarantee consistency of service compositions, and deliver application status notifications when a particular activity is completed or when a decision condition is reached. Considerations need also be made for modeling the scope in which a given service is being leveraged individual, composite, part of a long-running business process, and so on. Service monitoring allows monitoring events or information produced by the services/processes, monitoring instances of business processes, viewing process instance statistics, including the number of instances in each state (running, suspended, aborted or completed), viewing the status, or summary for selected process instances, suspend, and resume or terminate selected process instances. Of particular significance is the ability to be able to spot problems and exceptions in the business processes and move toward resolving them as soon as they occur. We refer to the role responsible for performing such operation

management functions as the service operator (see Fig. 1). Depending on the application requirements a service operator could be a service client or service aggregator.

It is increasingly important for service operators to define and support active capabilities versus traditional passive capabilities. For example, rather than merely raising an alert when a given service is unable to meet the performance requirements of a given service-level agreement, the management framework should be able to take corrective action itself. This action could take the form of rerouting requests to a backup service that is less heavily loaded, or automatically provisioning a new application server with an instance of the software providing the service if no backup is currently running and available.

Finally, service operations management also provides global visibility of running processes, comparable to that provided by Business Process Management tools. Management visibility is expressed in the form of real-time and historical reports, and in triggered actions. For example, deviations from key performance indicator target values, such as the percent of requests fulfilled within the limits specified by a service level agreement, might trigger an alert and an escalation procedure, or might propose changes to affected process models enabling them to achieve their goals.

Figure 3 highlights the elements of a conceptual architecture that combines a service management and an application channel developed in accordance to SOA principles.[39] This architecture provides a continuous connection between the Web services application channel and directs it into the management channel. Example management applications include availability and performance

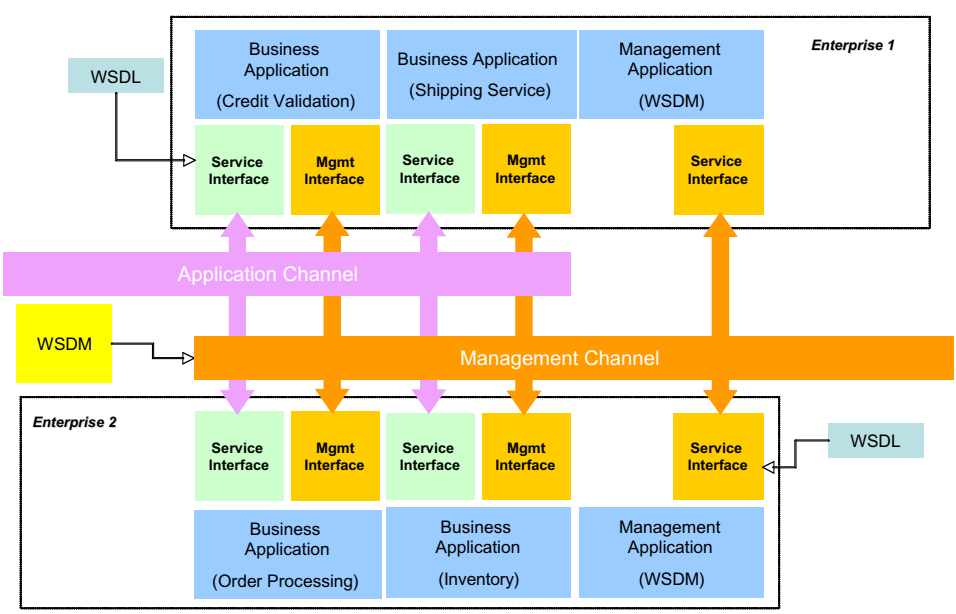

Fig. 3. Web services management architecture.

management, configuration management, capacity planning, asset protection, job control, and problem determination.

Service management in this conceptual architecture involves a collection of services that communicate with each other — passing data to each other or coordinating some activity together — all with the aim of facilitating the delivery of one or more business services. In fact, this architecture does not prescribe a particular management protocol or instrumentation technology because it needs to work with the various computing technologies and standards that exist in the industry today, such as Simple Network Management Protocol (SNMP), Java Management Extensions (JMX), WBEM, as well as future technologies.

In Fig. 3, manageable resources are as usual hardware and software resources, both physical and logical, e.g. software applications, hardware devices, servers, and so on, whose management capabilities are exposed as Web services that implement various management interfaces, such as those defined in Web Services Distributed Management specification (see below). A management interface of a resource is described by a WSDL document, resource properties schema, meta-data documents, and potentially a set of management related policies. Manageable resources can be accessed directly by resource managers, as part of a business processes and/or a management processes. In Fig. 3, a business process is composed of integrating basic services such as credit validation, shipping, order processing, and inventory services originating from two collaborating enterprises.

At the level of standards services management considers consistent management of end-to-end Web services. Such activities are the target of the Web Services Distributed Management (WSDM) specification. WSDM essentially defines a protocol for interoperability of management information and capabilities in a distributed environment via Web services. WSDM focuses on two distinct tasks in its attempt to solve distributed system management problems.[40] The first activity area, called Management Using Web Services (MUWS) addresses the use of Web services technologies as the foundation of a modern distributed systems management framework. This includes using Web services to facilitate interactions between managed resources and management applications. In particular, MUWS defines how to describe the manageability capabilities of managed resources using WSDL documents. Expressing capabilities enables more efficient discovery of and introspection of resources since managers, typically focus on a particular management task or domain, and therefore need to be able to easily and efficiently determine the relevant capabilities of a manageable resource.[39] In addition, WSDM addresses the specific requirements for managing Web services themselves just like any other resource. This activity is called Management of Web Services (MOWS).

The most recent wave of management product categories does not have the business-awareness that services management will require. The finer grained nature of services (as opposed to applications) requires evaluating processes and transactions at a more magnified rate and in a more contextually aware manner. Research activities have concentrated on assessing the impact of service execution from a

business perspective and, conversely, to adjust and optimize service executions based on stated business objectives.[41] This is a crucial issue as corporations strive to align service functionality with business goals.

One crucial aspect of management entails monitoring. Here research activities traditionally focus on dynamic monitoring techniques that are capable of employing monitoring rules governing the control of composite services,[42] e.g. such as those generated using BPEL processes. Other approaches concentrate on capturing and monitoring negotiations that incorporate security policies and policy models that facilitate service life-cycle management.[43]

The ability to gauge the quality of a service is critical if we are to achieve the Service-Oriented Computing paradigm. Many techniques have been proposed and most of them attempt to calculate the quality of a service by collecting quality ratings from the users of the service, and then combining them in one way or another. Collecting quality ratings alone from the users is not sufficient for deriving a reliable or accurate quality measure for a service. To this end, research activities have concentrated on using QoS metrics for selecting Web-services and for establishing trust between trading partners.[44]

Ideally, services are collaborating in highly distributed environments, naturally cutting across various enterprise boundaries. This environment demands that contracts are set up, stipulating agreements between services regarding their collaboration, both at the functional and non-functional level, in a concise manner. These contracts may serve as the basis for process monitoring and adaptation. Research activities in this front concentrate on standardizing on agreements between enterprise domains offering agreement templates, and facilities to dynamically check the state of an agreement.[45]

3.3.2. *Grand challenges*

Seeding autonomic capabilities for service level management is an evolutionary service level management approach where autonomic computing capabilities anticipate IT system requirements and resolve problems, with minimal human intervention. The function of any autonomic capability is a control loop that collects details from the system and acts accordingly.[4,40,46] Although there can be numerous types of control loops in a system, they can be naturally divided into five categories: self-configuring, self-adapting, self-healing, self-optimizing and self-protecting, each of which represents a major research challenge for future research in services management and monitoring. Some of the most notable research challenges for the near future include:

- *Self-configuring management services* — Self-configuring services configure themselves automatically to adapt to different environments in which they can be installed and can operate to optimize for particular kinds of their use.
- *Self-adapting management services* — Self-adapting services adapt dynamically to changes in the environment, market and so on, using policies provided by the

IT professional. Such changes could include the deployment of new instances of a particular kind of services or the removal of existing ones, or dramatic changes in the system characteristics.

- *Self-healing management services* — Self-healing services can discover, diagnose and react to disruptions. They can detect system malfunctions and initiate policy-based corrective action without disrupting the IT environment. Corrective action could involve a product altering its own state or effecting changes in other components in the environment. In this way, service-based solutions as a whole become more resilient because day-to-day operations are less likely to fail.
- *Self-optimizing management services* — Self-optimizing services can monitor and tune resources automatically. Self-optimizing components can tune themselves to meet end-user or business needs. The tuning actions could mean reallocating resources — such as in response to dynamically changing workloads — to improve overall utilization, or ensuring that particular business transactions can be completed in a timely fashion. Self-optimization helps provide a high standard of service for both the system's end users and a business's customers. Without self-optimizing functions, there is no easy way to divert excess server capacity to lower priority work when an application does not fully use its assigned computing resources. In such cases, customers must buy and maintain a separate infrastructure for each application to meet that application's most demanding computing needs.
- *Self-protecting management services* — Self-protecting services can anticipate, detect, identify and protect against threats. Self-protecting components can detect hostile behaviors, e.g. unauthorized access and use, virus infection and proliferation, and denial-of-service attacks, as they occur and take corrective actions to make themselves less vulnerable. Self-protecting capabilities allow businesses to consistently enforce security and privacy policies.

3.4. *Service design and development (Service-oriented engineering)*

Service-oriented applications start from the premise that all businesses have a business design. A business design describes how that business works — the processes that it performs; the organizational structure of the people and finances within that business; the business' near-term and long-term goals and objectives; the economic and market influences that affect how that business achieves its goals; the rules and policies that condition how the business operates.[47] The foundations of business design are business processes that are part of the fabric of a business and contribute to how the business functions and responds to its customers, opportunities, internal and external threats.[48]

Service orientation utilizes services as the constructs to support the development of rapid, low-cost and easy composition of distributed applications.[2] Key to this concept is the service-oriented architecture (SOA), which is a logical way of designing a software system to provide services to either end-user applications

or to other services distributed over a network, via published and discoverable interfaces. A well-constructed SOA can empower a business environment with a flexible infrastructure and processing environment by provisioning independent, reusable automated business processes (as services) and providing a robust foundation for leveraging these services. SOAs rely on an evolutionary software engineering approach that partly builds upon earlier development processes including component-based development and business process modeling.[49] Older software development paradigms for object-oriented and component-based development[50,51] cannot be blindly applied to SOA and Web services as they do not address the key elements of SOA: services, flows of information and components realizing services.[52] While relatively simple Web services may be built that way, a service-based development methodology is of critical importance to specify, construct, refine and customize highly volatile business processes from internally and externally available Web services.

What is required is a service-engineering methodology that allows enterprises to effectively design and deploy services and which can more easily embed changes into service-based applications at the rate and pace of change in the business design. It is from this correspondence that Service-oriented Architectures deliver on the promise of more flexible businesses through a more flexible IT environment. This correspondence is represented as the service-oriented engineering methodology, in which business processes are modeled, analyzed, assembled (possibly out of pre-existing components), deployed and monitored in a continuous and iterative manner. Figure 1 also illustrates that the Services Research Roadmap planes require the support of a SOA lifecycle methodology (service-oriented engineering), which starts with analyzing and modeling the business environment including key performance indicators of business goals and objectives, translating that model into service design, deploying that service system and managing that deployment.

3.4.1. *State of the art*

A service-oriented design and development methodology focuses on business processes, which it considers as reusable elements that are independent of applications and the computing platforms on which they run. This promotes the idea of viewing enterprise solutions as federations of services connected via well-specified contracts that define service interfaces in the context of SOAs.

Many researchers and developers in their early use of SOA, think that they can port existing components to act as Web services just by creating wrappers and leaving the underlying component untouched. Since component methodologies focus on the interface, many developers assume that these methodologies apply equally well to service-oriented architectures. Thus, implementing a thin SOAP/WSDL/UDDI veneer on top of existing applications or components that implement the Web services is by now widely practiced by the software industry. Yet, this is in no way sufficient to construct commercial strength enterprise applications. Unless the nature of the component makes it suitable for use as a Web service, and most are

not, it takes serious thought and redesign effort to properly deliver components functionality through a Web service. While relatively simple Web services may be effectively built that way, a methodology is of critical importance to specify, construct, refine and customize highly volatile business processes from internally and externally available Web services.

Conventional development methodologies such as Object-Oriented Development (OOD) and Component-Based Development (CBD) do not address the three key elements of an SOA: services, service assemblies (composition), and components realizing services. These methodologies can only address part of the requirements of service-oriented computing applications. These practices fail when they attempt to develop service-oriented solutions while being applied independently of each other. For instance, although both CBD and service-oriented computing offer a "separation-of-internal and external perspectives" and the motivation for both components and services is often expressed in terms of reusability, composability and flexibility, they are quite diverse in nature. Components and services present differences along the dimensions of type of communication, type of coupling, type of interface, type of invocation, and type of request brokering. However, in so far as development is concerned, they also differ fundamentally in the way that they approach flexibility and reusability. Services are subject to continuous maintenance and improvement in scope and performance so that they can be offered to an ever-increasing number of consumers. The selection of a service is usually done dynamically on the basis of a set of policies. Use of installed components does not allow for the same kind of reuse and dynamic behaviour. Moreover, the view that components are merely distributable objects, deployed on some middleware server, carries with it all the difficulties of object modeling and yet multiplies the complexity by increasing the scale of the model.[53] Let alone if models are extended across enterprise boundaries.

Service-oriented design and development requires an inter-disciplinary approach fusing elements of object-oriented and component design with elements of business modeling. OOD and CBD can contribute general software architecture principles such as information hiding, modularization, and separation of concerns. On the other hand, business modeling can contribute conventions that help analyze the structuring of value-chains and improve processes, help define amongst other things standardized business processes and operating procedures, and create a shared understanding of how a business functions so that workflow implementations are tested before design and implementation.

As regards to research, it is only recently that we see some initial results. Activities have mainly concentrated on two fronts on developing a methodology for service-oriented engineering and on design-time models.

On the first activities concentrated on proposing elements of a methodology for services design and development. Research activities have concentrated on how to provide sufficient principles and guidelines to specify, construct and refine and customize highly volatile business processes choreographed from a set of internal and external Web services.[53,54]

On the second front activities have concentrated on developing design-time models using goal-oriented requirements analysis techniques.[55] According to these, requirements analysis begins by identifying stakeholder goals (needs). The goals are refined through AND/OR decompositions to identify collections of actions which together can satisfy each root-level goal. Web services are designed for each one of these actions. The goal models can then be made available at run-time to augment UDDI and other discovery infrastructure in identifying and selecting what Web services to compose in order to fulfil a user need.[56,57]

3.4.2. *Grand challenges*

Business processes and services in a service-oriented application are implemented as components in terms of financial and operational functions and data available from resources such as ERP, databases, CRM and other systems. It is thus convenient to distinguish between business processes and business services as comprising the *logical part of services development life cycle* and the *physical part of services development life cycle* that comprises infrastructure services and component implementations that map logical services to existing resources (this is analogous to the distinction between business and system level separation in Sec. 3.2.2). The logical part is shown in Fig. 4 to comprise business processes, and business services, such

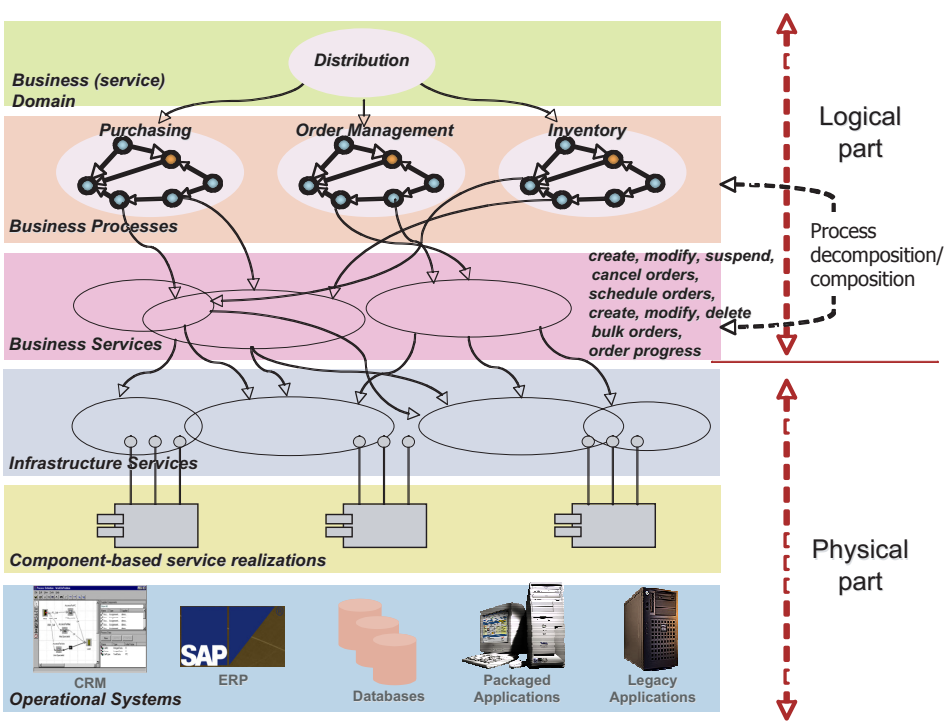

Fig. 4. The Web Services Development Life Cycle hierarchy.

as an order management process that provide business services (which are indivisible services) for creating, modifying, suspending, canceling, querying orders, and so on. Infrastructure services are usually management and monitoring services woven in by a container and in the services management and monitoring infrastructure such as those providing technical utility such as logging, security, transactionality or authentication, and those that manage resources. These services provide the infrastructure, enabling the integration of services through the introduction of a reliable set of capabilities, such as intelligent routing, protocol mediation, and other transformation mechanisms, often considered as part of the Enterprise Service Bus. The implementation part provides through component implementations the financial and operational functions and data available from resources such as ERP, databases, CRM and other systems, which lie at the bottom of the service lifecycle development hierarchy, which automatically populate the service domains, and business processes. Both business services and their implementation components need to be designed with the appropriate level of granularity.

The above issues raise a plethora of interesting research challenges that need to be addressed in order to develop sound service engineering methodologies. Major research challenges for the near future include:

- *Engineering of Service Compositions*: One of the main challenges in the development of Services-oriented systems is the provision of methodologies that support the specification and design of compositions of services. Traditional software engineering methodologies hardly apply in this scenario, where the environment is highly dynamic, the uncertainty is high and several decisions cannot be taken at design time but must be postponed at run-time, where the control is highly distributed, and we have different stakeholders with possibly conflicting business needs. Novel techniques must be developed to support the refinement from the early phases of requirement analysis to the final steps of implementation and deployment. Similarly, novel techniques must be devised to construct compositions of Web services that at run-time can provide feedback and significant information to business analysis and stakeholders, who can use this information to devise new business strategies or take strategic decisions at design time.
- *Associating a services engineering methodology with standard software development and business process modeling techniques*: Most representative current conventional software development techniques is the Rational Unified Process (RUP) whose aim is to support the analysis and design of iterative software development. RUP has the principles of object-oriented analysis and design and CDB as its foundation, and therefore, does not lend itself readily to be aligned to SOA design.[58] Research activities could focus on how to blend several of the RUP milestones and connect them with business modeling approaches such as the Supply Chain Operations Reference (SCOR) Framework proposed by the Supply Chain Council (http://www.supply-chain.org/) to provide a sound basis for supporting the corresponding phases of a services development methodology. SCOR is a modeling approach that provides standard guidelines for companies that help to

examine the configuration of their supply chains, identify and measure metrics in the supply chain. In addition, it should be examined how emerging business modeling notations such as BPMN can aid in capturing software development activities rather than using UML.

- *Flexible gap analysis techniques*: Gap analysis is a technique that purposes a business process and services realization strategy by incrementally adding more implementation details to an abstract service/process interface. Gap analysis commences with comparing candidate service functionality with available software service implementations that may be assembled within the enclosures of a newly conceived business process. A gap analysis strategy may be developed in stages and results in a recommendation to do development work, reuse or purchase services. It considers several service realization possibilities such as green field development, top-down and bottom-up development, meet in the middle development and development on the basis of reference models.
- *Design principles for engineering service applications*: In order to design useful and reliable business processes that are developed on the basis of existing or newly coded services, we need to apply sound design principles that guarantee that services are self-contained and come equipped with clearly defined boundaries and service end-points to allow for service composability and loose coupling. Key principles that serve as the foundation for service design: service coupling, cohesion and granularity.
- *Automated, transparent user centred support to the entire business process lifecycle of composed services*: The challenge is to automatically perform the time consuming and error prone task of analyzing business processes in detail, discovering and selecting suitable external services, detecting problems in the interactions, searching for possible alternative solutions, monitoring execution step by step, and so on. There is a need for techniques that operate in a transparent and user centred way by suggesting solutions that can be adopted, refused, or refined by stakeholders, managers, business analysts, designers, and programmers.
- *Design techniques for managing service versioning and adaptivity*: Adaptive service capabilities need to be seeded into services and processes so that they can continually morph themselves to respond to environmental demands and changes without compromising on operational and financial efficiencies. Business processes could be analyzed in detail instantaneously, discovering and selecting suitable external services, detecting problems in the service interactions, searching for possible alternative solutions, monitoring execution step by step, upgrading and versioning themselves, and so on. An integral part of adaptive services is service version control. A service interface version is a specific instance of a service interface at a particular point in time that came into existence due to a revision or a change. Currently, versioning has not been built into the Web services architecture. This means that in situations where the interface of a particular service needs to change with a new version WSDL cannot convey the change to the service requester. If a developer makes a change to a service interface, all

older requestors would fail, and the failure would be undetectable to the Web services infrastructure.

- *Service Governance*: The cross-organizational nature of end-to-end business processes that are composed out of variety of service fragments that may need to be maintained separately by different organizations makes service governance a challenging issue. The potential composition of services into business processes across organizational boundaries can function properly and efficiently only if the services are effectively governed for compliance to requirements dictated by QoS factors. Services that flow between enterprises must have defined owners with established ownership and governance responsibilities. These owners are responsible for gathering requirements, development, deployment, the boarding process, and operations management for any mission critical or revenue generating service.[59] The service must meet the functional and QoS objectives within the context of the business unit and the enterprises within which it operates. Ownership of a specific service is usually associated with a business scope. Typical examples of such business scopes are customer relationship management, customer information and entitlements, order management, financing, taxes, and so forth. Consequently, identifying, specifying, creating, deploying enterprise services, and overseeing their proper maintenance and growth needs SOA governance to oversee the entire life cycle of an enterprise's service portfolio.[60]

4. Summary

Service-oriented computing is a novel computing paradigm that promotes the idea of assembling application components with little effort into a network of services that can be loosely coupled to create flexible dynamic business processes and agile applications that may span organizations and computing platforms.

The subject of Service-oriented computing is vast and enormously complex, spanning many concepts and technologies that find their origins in diverse disciplines that are woven together in an intricate manner. As a result research activities are very fragmented. This necessitates that a broader vision and perspective be established — one that permeates and transforms the fundamental requirements of complex applications that require the use of the Service-oriented computing paradigm. In this paper, we have provided a Service-oriented computing Roadmap and places on-going research activities and projects in the area of SOC in the broader context of this roadmap.

The SOC research roadmap launches four pivotal, inherently related, research themes to Service-oriented computing: service foundations, service composition, service management and monitoring and service-oriented engineering.

Service foundations provide a service oriented middleware backbone that realizes the run-time SOA infrastructure that connects heterogeneous components and systems, and provides multiple-channel access to services, e.g. via mobile devices, hand held devices, over variety of networks including the Internet, cable, UMTS, XDSL, Bluetooth, and so on. This run-time infrastructure allows defining basic

interactions involving the description, publishing, finding and binding of services. Major research challenges for the service foundations plane include: dynamically (re-)configurable run-time architectures, dynamic connectivity, topic and content-based routing capabilities, end-to-end security solutions, infrastructure support for application integration, data integration and process integration, and semantically enhanced service discovery.

Service composition encompasses the necessary roles and functionality for the aggregation of multiple services into a single composite service. Resulting composite services may be used by service as basic services in further service compositions or may be offered as complete applications/solutions to service clients. Some of the most notable research challenges for include: composability analysis for replaceability, compatibility, and conformance for dynamic and adaptive processes, adaptive and emergent service compositions, autonomic composition of services, QoS-aware service compositions, business-driven automated compositions.

Service management encompasses the control and monitoring of SOA-based applications throughout their life cycle. Service management spans a range of activities from installation and configuration to collecting metrics and tuning to ensure responsive service execution. It includes many interrelated functions such as Service-Level Agreement negotiation, management, auditing, monitoring, and troubleshooting, service lifecycle/state management, performance management, services and resources provisioning, and includes aspects like scalability, availability and extensibility and others. Some of the most notable research challenges include seeding autonomic capabilities in service management and monitoring such as self-configuring, self-adapting, self-healing, self-optimizing and self-protecting capabilities.

Service-oriented engineering allows enterprises to effectively design and deploy services which can more easily embed changes into service-based applications at the rate and pace of change in the business design. It is from this correspondence that SOAs deliver on the promise of more flexible businesses through a more flexible IT environment. This correspondence is represented as the service-oriented engineering methodology, in which business processes are modeled, analyzed, assembled (possibly out of pre-existing components), deployed and monitored in a continuous and iterative manner. Major research challenges for the near future include the engineering of service compositions, associating a services engineering methodology with standard software development and business process modeling techniques, flexible gap analysis techniques, design principles for engineering service applications, automated, transparent user centred support to the entire business process lifecycle of composed services, design techniques for managing service versioning and adaptivity, and finally, service governance.

References

1. F. Leymann, The (Service) bus: Services penetrate everyday life, *3rd Int. Conf. on Service-Oriented Computing ICSOC'2005*, LNCS, Vol. 3826 (Springer-Verlag Berlin Heidelberg, Amsterdam, The Netherlands, December 13–16, 2005).

2. M. P. Papazoglou and G. Georgakapoulos, Service-oriented computing, CACM, **46**(10) (October 2003).
3. S. Weerawarana, F. Curbera, F. Leymann, T. Storey and D. F. Ferguson, *Web Services Platform Architecture* (Prentice Hall, 2005).
4. F. Leymann, Combining web services and the grid: Towards adaptive enterprise applications, in *Proc. CAiSE/ASMEA'05* (Porto, Portugal, June 2005).
5. F. Leymann, Web Services: Distributed applications without limits, in *Proc. BTW'03 Lecture Notes in Informatics*, Vol. P-26 (Leipzig, Germany, February 26–28, 2003).
6. M. P. Papazoglou, Extending the service oriented architecture, *Business Integration Journal*, February 2005.
7. S. Graham *et al.*, *Building Web Services with Java*, 2nd edn. (SAMS Publishing, 2005).
8. M. P. Papazoglou and W. J. van den Heuvel, Service-oriented architectures, *VLDB Journal* **16** (July 2007).
9. J. L'opez and D. O'Hallaron, Evaluation of a resource selection mechanism for complex network services, in *Proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing* (Aug. 2001).
10. C. Liu, L. Yang, I. Foster and D. Angulo, Design and evaluation of a resource selection framework for grid applications, *IEEE International Symposium on High Performance Distributed Computing (HPDC-11)* (July 2002).
11. V. Poladian, D. Garlan, M. Shaw and J. P. Sousa, Dynamic configuration of resource-aware services, in *Proceedings of the 26th International Conference on Software Engineering* (May 2004).
12. A.-A. Ivan, J. Harman, M. Allen and V. Karamcheti, Partitionable services: A framework for seamlessly adapting distributed applications to heterogeneous environments, in *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing* (2002).
13. S. R. Ponnekanti and A. Fox, SWORD: A Developer Toolkit for Web Service Composition, 11th World Wide Web Conference (Web Engineering Track), May 2002.
14. V. Deora *et al.*, A quality of service management framework based on user expectations, in *Proceedings of the First International Conference on Service-Oriented Computing* (*ICSOC03*) (Springer, 2003).
15. S. Ran, A Model for Web Services Discovery with QoS, SIGecom Exchange **4**(1) (2003).
16. V. Deora *et al.*, Incorporating QoS specifications in service discovery, WISE Workshops, Lecture Notes of Springer Verlag (2004).
17. Y. Wang and E. Stroulia, Semantic structure matching for assessing web-service similarity, *1st International Conference on Service-Oriented Computing* (*ICSOC03*) (Springer-Verlag, 2003).
18. M. C. Jaeger and S. Tang, Ranked matching for service descriptions using DAML-S, in *Proceedings of CAiSE'04 Workshops* (Riga, Latvia, June 2004).
19. X. Ding *et al.*, Similarity search for web services, *30th VLDB Conference* (September 2004).
20. O. D. Sahin *et al.*, SPiDeR: P2P-Based Web Service Discovery, *3rd International Conference on Service-Oriented Computing* (Springer-Verlag, Amsterdam, December 2005).
21. M. Hepp, F. Leymann, J. Domingue, A. Wahler and D. Fensel, Semantic business process management: Using semantic web services for business process management, in *Proc. IEEE ICEBE 2005* (Beijing, China, October 18–20, 2005).
22. A. A. Patil *et al.*, Meteor-S: Web Service Annotation Framework, WWW'04: *Proceedings of the 13th International Conference on World Wide Web* (ACM Press, 2004), pp. 553–562.

23. D. Roman, *Web Service Modeling Ontology, Applied Ontology* **1**(1) (IOS Press, 2005).
24. Security in a Web Services World: A Proposed Architecture and Roadmap, IBM Developer Works, April 2002.
25. T. Andrews *et al.* (eds), Business Process Execution Language for Web Services (May 2003), available at http://www.ibm.com/developerworks/library/ws-bpel.
26. J. Yang and M. P. Papazoglou, Service components for managing the life-cycle of service compositions, *Information Systems* **28**(1) (2004).
27. A. Charfi and M. Mezini, Hybrid Web service composition: Business processes meet business rules, *Int. Conf. on Service-Oriented Computing* (*ICSOC 2004*), New York (Dec 2004).
28. M. Solanki, A. Cau and H. Zedan, Augmenting semantic web service descriptions with compositional specification, *WWW '04: 13th International Conference on World Wide Web* (ACM Press, NY, USA, 2004).
29. J. L. Ambite *et al.*, Argos: An Ontology and Web Service Composition Infrastructure for Goods Movement Analysis, National Conference on Digital Government Research, Seattle, Washington, May 2004.
30. M. Paolucci *et al.*, Semantic matching of web services capabilities, *1st International Semantic Web Conference* (2002).
31. A. Lazovik, M. Aiello and M. P. Papazoglou, Associating assertions with business processes and monitoring their execution, *Int. Conf. on Service-Oriented Computing* (*ICSOC 2004*), New York (Dec 2004).
32. A. Lazovik, M. Aiello and M. P. Papazoglou, Planning and monitoring the execution of web service requests, *Int. Journal on Digital Libraries* to appear in 2006.
33. P. Traverso and M. Pistore, Automatic composition of semantic web services into executable processes, *International Semantic Web Conference (ISWC) 2004*.
34. M. Pistore, P. Traverso, P. Bertoli and A. Marconi, Automated Synthesis of Executable Web Serivce Compositions from BPEL4WS Processes, Special Track at the International World Wide Web Conference (WWW) 2005.
35. R. Kazhamiakin and M. Pistore, A Parametric Communication Model for the Verification of BPEL4WS Compositions, International World Wide Web Conference (WWW) 2006.
36. F. Barbon, P. Traverso, M. Pistore and M. Trainotti, Run-Time Monitoring the Execution of Web Service Compositions, The International Conference on Planning and Scheduling (ICAPS) 2006.
37. E. Colobo, J. Mylopoulos and P. Spoletini, Modeling and analyzing context-aware composition of services, *3rd International Conference on Service-Oriented Computing* (Springer, Amsterdam, December 2005).
38. R. Agrawal, Ch. Johnson, J. Kiernan and F. Leymann, Taming compliance with Sarbanes-Oxley internal controls using database technology, *22nd Int. Conf. on Data Engineering ICDE'2006*, Altanta, GA, USA (April 2006).
39. M. P. Papazoglou and W. J. van den Heuvel, Web Services Management: A Survey, *IEEE Internet Computing* (November/December 2005).
40. H. Kreger *et al.*, Management Using Web Services: A Proposed Architecture and Roadmap, IBM, HP and Computer Associates (June 2005), available at www-128.ibm.com/developerworks/library/specification/ws-mroadmap.
41. F. Casati *et al.*, Business-Oriented Management of Web Services, *Communications of the ACM* **46**(10) (2003).
42. L. Baresi and S. Guinea, Towards dynamic monitoring of WS-BPEL processes, in *Proceedings of the 3rd International Conference on Service-Oriented Computing* (*ICSOC 2005*) (Springer, Amsterdam, December 2005).

43. H. Skogsrud, B. Benatallah and F. Casati, Trust-serv: Model-driven lifecycle management of trust negotiation policies for web services, *WWW '04: Proceedings of the 13th International Conference on World Wide Web* (ACM Press, NY, USA, 2004), pp. 53–62.
44. E. M. Maximilien and M. P. Singh, Toward autonomic web services trust and selection, *Int. Conf. on Service-Oriented Computing* (*ICSOC 2004*), New York (Dec 2004).
45. H. Ludwig, A. Dan and R. Kearney, Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements, *Int. Conf. on Service-Oriented Computing* (*ICSOC 2004*), New York (Dec 2004).
46. J. O. Kephart and D. M. Chess, The vision of autonomic computing, *IEEE Computer* (January 2003).
47. M. P. Papazoglou and P. M. A. Ribbers, e-*Business: Organizational and Technical Foundations* (J. Wiley & Sons, Ltd, February 2006).
48. F. Leymann and D. Roller, *Production Workflow — Concepts and Techniques* (PTR Prentice Hall, 2000).
49. P. Harmon, Second generation business process methodologies, *Business Process Trends* **1**(5) (May 2003).
50. F. Bachmann *et al.*, Technical Concepts of Component-Based Software Engineering, Technical Report, Carnegie-Mellon Univ., CMU/SEI-2000-TR-008 ESC-TR-2000-007, 2nd Edition, May 2000.
51. F. Bachmann *et al.*, Technical Concepts of Component-Based Software Engineering, Technical Report, Carnegie-Mellon Univ., CMU/SEI-2000-TR-008 ESC-TR-2000-007, 2nd Edition, May 2000.
52. A. Arsanjani, Service-oriented Modeling and Architecture, IBM developerworks (Novemeber 2004), available at http://www-106.ibm.com/developerworks/library/ws-soa-design1/.
53. M. P. Papazoglou and W. van den Heuvel, Business Process Development Lifecycle Methodology to Communications of ACM, October 2007.
54. C. Ghezzi, Service-Oriented Computing: Where does it come from? A software engineering perspective, keynote address at *Int. Conf. on Service-Oriented Computing* (*ICSOC 2005*), Amsterdam (Dec 2005).
55. A. van Lamsweerde, Requirements Engineering in the Year 2000: A Research Perspective, *22nd International Conference on Software Engineering*, Limerick, Ireland, (May 2000).
56. R. S. Kaabi, C. Souveyet and C. Rolland, Eliciting service composition in a goal driven manner, *Int. Conf. on Service-Oriented Computing* (*ICSOC 2004*), New York (Dec 2004).
57. L. Penserini, A. Perini, A. Susi and J. Mylopoulos, From Stakeholder Needs to Service Requirements Specifications, Technical Report, ITC-IRST, Automated Reasoning Systems, 2006.
58. O. Zimmerman, P. Korgdahl and C. Gee, Elements of Service-oriented Analysis and Design, IBM developerworks (June 2004), available at http://www-106.ibm.com/developerworks/library/ws-soad1/.
59. N. Bieberstein *et al.*, Impact of service-oriented architecture on enterprise systems, organizational structures, and individuals, *IBM Systems Journal* **44**(4) (2005).
60. T. Mitra, A Case for SOA Governance, IBM developerworks, (August 2005), available at http://www-106.ibm.com/developerworks/Webservices/library/ws-soa-govern/index.html.