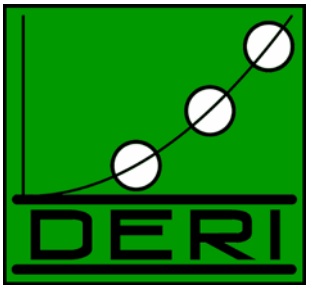
DERI – Digital Enterprise Research Institute



Semantically Enabled Service-Oriented Architectures:

A Manifesto and a Paradigm Shift in Computer Science

Michael Brodie[1]
Christoph Bussler[2]
Jos de Brujin[3]
Thomas Fahringer[4]
Dieter Fensel[3,5]
Martin Hepp[3]
Holger Lausen[3]
Dumitru Roman[3]
Thomas Strang[3,6]
Hannes Werthner[7]
and Michal Zaremba[5]

[1] Verizon, USA
[2] Cisco Systems, Inc., USA
[3] DERI Innsbruck, Leopold-Franzens Universität Innsbruck, Austria
[4] Institute for Computer Science, Leopold-Franzens Universität Innsbruck, Austria
[5] DERI Galway, National University of Ireland, Galway, Ireland
[6] German Aerospace Center (DLR), Oberpfaffenhofen, Germany
[7] Department for Information Systems and e-tourism, Leopold-Franzens Universität Innsbruck, Austria

Technical Report TR-2005-12-26

26 December 2005

DERI Galway
National University of Ireland
Galway
Ireland
www.deri.ie

DERI Innsbruck
University of Innsbruck
Technikerstrasse 21a
Innsbruck
Austria
www.deri.at

DERI Seoul
Seoul National University
Yeonggun-Dong, Chongno-Gu
Seoul
Korea
www.deri.org/korea

DERI Stanford
Stanford University
Serra Mall
Stanford
USA
www.deri.us

Table of Contents

Abstract 6

1 Introduction 7

2 The SESA Vision 10

2.1 Service-Orientation 11

2.2 What does Service-Oriented mean? 13

2.3 Need for a Semantic Extensions of the SOA Framework 17

2.4 Implications of our Approach 23

2.4.1 Resource Management 24

2.4.2 Architecture as an Emergent Property 26

2.4.3 Example Problem Solving Scenario 28

2.4.4 The Economics of IT 33

3 Problem Solving Layer 38

4 Common Service Layer 43

4.1 WSMO 45

4.2 WSML 49

4.3 WSMX 52

4.4 Triple Space 58

5 Resource Layer 64

5.1 Ubiquitous and Grid Computing as part of the Resource Layer 66

5.2 Connections are Services 69

5.3 Semantically Enabled Resource Management 70

6 Progress towards the Vision 73

6.1 Research Efforts 74

6.1.1 The OWL-S Approach 75

6.1.2 The SWSF Approach 77

6.1.3 The WSDL-S Approach 80

6.1.4 The IRS-III Approach ... 82
6.1.5 Relations with W<Triple> ... 84
6.2 Standardization Efforts ... 86
6.3 Industrial Efforts ... 89
6.3.1 The State of SOA Adoption ... 90
6.3.2 Semantic Solutions: Demand and Supply ... 94
6.3.2.1 Demand: Dynamic, Scalable SOA ... 95
6.3.2.2 Domain-Specific Demand: Pharmaceuticals ... 96
6.3.2.3 Demand: Integration with Semantic Assurance ... 98
6.3.2.4 Supply: Semantically Enabled SOA and Integration ... 101
6.3.2.5 Supply: Semantically Enabled Solutions ... 105

7 Conclusion ... 107
7.1 The SESA Challenge ... 109

8 References ... 111

Table of Figures

Figure 1. Three Layers of Semantically Enabled Service-Oriented Architecture (SESA). 9
Figure 2. SOA Values and Functionality. 14
Figure 3. SOA Values over services based over core application platforms. 15
Figure 4. A 2-node federated, distributed SOA or simple SOA GRID. 16
Figure 5. Tourist life cycle and companies' processes – both suppliers and intermediaries. 29
Figure 6. e-commerce framework. 38
Figure 7. e-commerce framework. 40
Figure 8. WSML family of languages. 49
Figure 9. WSMX: A Reference Architecture for SEE. 53
Figure 10. Mapping WSMX to SESA. 55
Figure 11. Four Phases of Computing / Communication Convergence. 58
Figure 12. Resource Management in the Context of an SOA. 65
Figure 13. Top level elements of OWL-S [OWL-S, 2004]. 75
Figure 14. The Layered Structure of SWSL-Rules [SWSF, 2005]. 78
Figure 15. Associating semantics to WSDL elements [Akkiraju et al., 2005]. 80

Abstract

After four decades of rapid advances in computing, we are embarking on the greatest leap forward in computing that includes revolutionary changes at all levels of computing from the hardware through the middleware and infrastructure to applications and more importantly in intelligence. This paper outlines a comprehensive framework that integrates two complimentary and revolutionary technical advances, Service-Oriented Architectures (SOA) and Semantic Web, into a single computing architecture, that we call Semantically Enabled Service-Oriented Architecture (SESA). While SOA is widely acknowledged for its potential to revolutionize the world of computing, that success depends on resolving two fundamental challenges that SOA does not address, integration, and search or mediation. In a services-oriented world, billions of services must be discovered and selected based on requirements, then orchestrated and adapted or integrated. SOA depends on but does not address either search or integration. The contribution of this paper is to provide the semantics-based solution to search and integration that will enable the SOA revolution. The paper provides a vision of the future enabled by our framework that places computing and programming at the services layer and places the real goal of computing, problem solving, in the hands of end users.

1 Introduction

Computer science appears to be in a period of crisis. The globalization trend is moving programming jobs from Europe and the US to low-labour countries such as China and India. This appears to place computer science research and departments at risk of being considered as working on obsolete technology. However, the opposite is true. Computer science is on the verge of a new generation of computing that is leading to innovation not only in computing but also in business, science, and all other endeavours that depend on computing.

Times of crisis are also times of innovation and can lead to paradigm shifts. Computer science is entering a new generation. The previous generation was based on abstracting from hardware. The emerging generation comes from abstracting from software and sees all resources as services in a *Service-Oriented Architecture (SOA)*. "A Service-Oriented Architecture is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. Some means of connecting services to each other is needed."[1] In a world of services, users are concerned only about the services and not about any software or hardware components that implement the service. Service-Oriented Computing is rapidly becoming the dominant computing paradigm.

A service-oriented world will have billions of services. Computation will involve services searching for services based on functional and non-functional requirements and interoperating with those that they select. However, services will not be able to interact automatically and SOAs will not scale without signification mechanization of service discovery, negotiation, adaptation, composition, invocation, and monitoring as well as service interaction which will require further data, protocol, and process mediation. Hence, machine processable *semantics* are critical for the next generation of computing - SOAs - to reach its full potential. Only with semantics can critical subtasks can be automated leaving humans to focus on problem solving.

[1] http://www.service-architecture.com/web-services/articles/ serviceoriented_architecture_soa_definition.html.

The goal of *Semantically Enabled Service-Oriented Architectures (SESA)* is to place semantics at the core of computer science in order to realize the potential of the next generation of computing. SESA will provide a next generation operating system that provides seamless and transparent integration of billions of services on a global scale. Semantic descriptions will enable computing to become a utility, just as electricity is today. In this paper we define the SESA manifesto and challenge the computing industry and research communities to close the SOA semantic gap identified in the SESA manifesto. This paper outlines a comprehensive framework that augments existing SOA frameworks - technologies, standards, and guidelines, to incorporate semantic solutions to address the SOA semantic gap. To resolve the SOA semantic gap, we propose an extension of the SOA framework to the SESA framework that provides a basis for the semantic specification of SOA, the means for grounding semantic specifications in each component of the SOA framework, and, semantically enabled solutions for the components of the SOA framework. In some areas we identify the need for and potential of semantic solutions, in other areas we propose specific directions, and in still others we provide reference implementations. We challenge the community to collaborate in realizing the SESA manifesto.

While a decade of research on the semantic web with its focus on defining semantics of data[2] has led to a deeper understanding of data semantics, the scalable semantic description of services is still in its infancy. We intend to lead in this direction though a better understanding of how semantics can become a critical component in modern computer engineering so that it acts as a new generation of operation system that enables resource sharing at a global scale. Typically in the past semantics has been studied in Artificial Intelligence for isolated computing or human tasks (e.g., simulating or achieving human intelligence with a computer). Our objective is to make semantics a pillar of the software architecture of the next generation of computing. This paper provides a comprehensive framework with which to augment the worldwide movement to service-orientation with semantics in the context of evolving industrial standards and technologies. The framework provides a basis for a manifesto to the industrial and research communities to address the SOA semantic gap so that SOA benefits can be fully realized.

[2] Refer to *Proceedings of the 4th International Semantic Web Conference (ISWC 2005)*, Galway, Ireland, November, 2005, *Lecture Notes in Computer Science (LNCS 3729)*, Springer-Verlag, Berlin. http://iswc2005.semanticweb.org/.

The content of the paper is structured as following. Section 2 provides our vision of semantically enabled Service-Oriented Computing that can result from the current paradigm shift in computer science. Section 3, 4, and 5 discuss the three main elements of a semantically enabled SOA - the problem-solving layer, the common service layer, and the resource layer, depicted in Figure 1. Section 6 enumerates existing steps towards such a computing architecture. Section 7 provides a conclusion and a challenge.

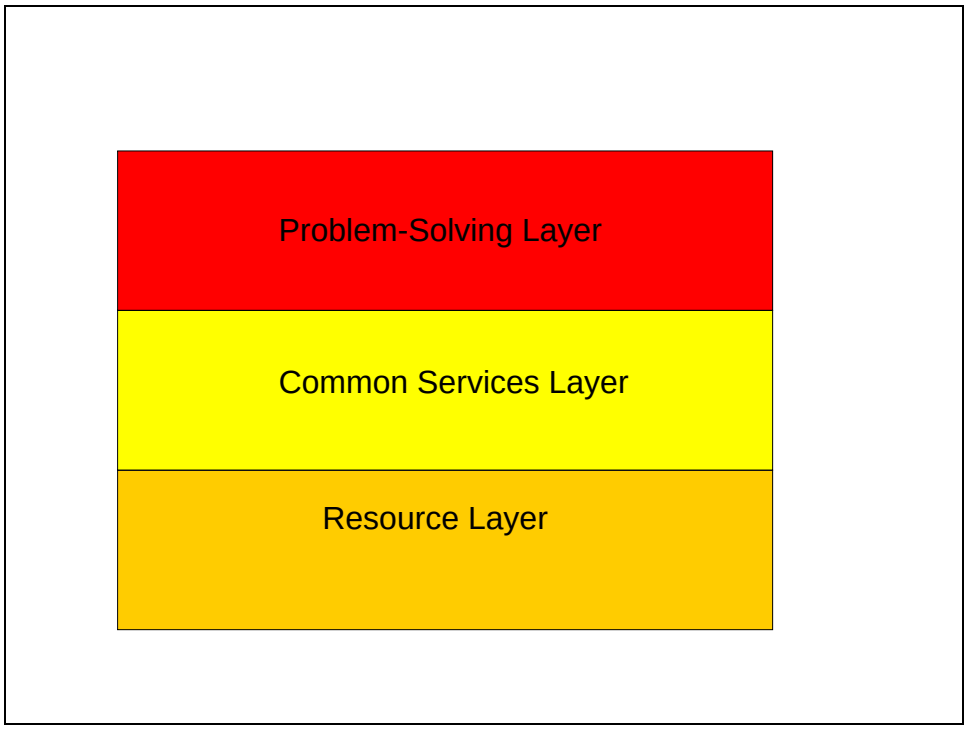


Figure 1. Three Layers of Semantically Enabled Service-Oriented Architecture (SESA).

2 The SESA Vision

This section provides a description of our vision on a new path for computer science. We motivate and define service orientation, argue why semantic enablement is needed, and conclude with some of the implications of our approach.

2.1 Service-Orientation

Conventional computing has provided a basis for economic growth and problem solving unprecedented in human history. After half a century, conventional computing has hit the wall of complexity. Since the early 1950's each decade has produced major breakthroughs in computing and information technology. Computer science has contributed major breakthroughs in software design, programming, and infrastructure. However, due to the incremental nature of information technology, advances have been built one on top of the other, resulting in layer upon layers of legacy technology that reduce major breakthroughs to local optimizations. Each decade's advance, e.g., client-server, could not overcome the cost of migrating from the technology layers to the architectures based on the newest contribution [Brodie et al., 1995]. As a result, full advantage was not taken of new contributions. Indeed, new contributions added complexity to the layered legacy. Hardware advances have vastly outpaced those of software in part due to the lack of complexity, layering, and challenges of legacy software. The demands of modern business on information systems and computing have long outpaced the capabilities of software to respond. This is amply demonstrated by the fact that approximately 95% of IT budgets are devoted to maintenance and enhancement of the legacy. SOA marks the first time in computing history that an advance that will overcome the wall of cost and complexity.

Service-orientation is the largest leap forward in computing in half a century. After thirty years of evolution, SOA offers the promise of a comprehensive computational model that applies at all levels of our current computing hierarchy, including hardware, network, and all software layers up to user tools and interfaces. Hence, SOA offers a complete replacement of the layered software legacy.

From a technical point of view, the primary benefit of SOA is the inherent ability of reusing services in new contexts. This addresses two of the major costs of conventional computing, development and integration. Services are developed once for reuse in multiple contexts. Development of complex services involves the selection of services appropriate for the given context and the orchestration of those services into a composite or complex service to meet the requirements of that context. The same process can be used to modify or extend the complex service. This SOA (re-) development paradigm facilitates the flexibility to create, modify or redesign services in the face of

constantly changing business requirements and constantly evolving supporting services. These technical benefits have resulted in a much greater and higher-level benefit of SOA, that of business flexibility and facilitated the ability of businesses to create, modify, or re-design their businesses rapidly at considerably lower cost than using conventional computing.

These technical and business benefits are generally believed to help overcome the previously insurmountable barriers of the cost of development of the components of the next generation of computing and of the migration from the legacy to the next generation [Brodie et al., 1995]. Every hardware and software vendor is now fully committed to SOA and the movement to SOA. One example of avoiding the complexities and cost of layering new technology on old is Microsoft's VISTA, previously code-named Longhorn. Vista is a complete re-write and replacement of Microsoft's Windows operating system and all supporting infrastructure. Microsoft's Vista will replace previous legacy Windows infrastructure.

2.2 What does Service-Oriented mean?

The fundamental technical characteristics of SOA are a service and remote service invocation. A service is a unit of executable code that can be invoked by another service remotely and inexpensively anywhere in the network. Compared with conventional computing models that require significant human, programming, and computing resources for applications to communicate and interoperate, the service-oriented computational model is based on communication (messaging) and interoperability between services to discover and invoke remote services, based on requirements, and to orchestrate services into composite services or service workflows.

Presumably an SOA is the architecture that supports services-orientated computing; yet there is no architecture inherent in services and service invocation. Indeed, SOA is not an architecture characterization; it is a style of programming, as described in more detail in Section 2.4.2. As there is no standard definition of an SOA we adopt a vendor neutral description of SOA functionality [Heffner, 2005] which is characterized by three values that an SOA supports – *Change*, *Connection*, and *Control*. Each value is provided by a set of functions. Flexible business Change, the key SOA benefit, is enabled by a *Service Life-Cycle Environment* that supports the service life cycle in which services are developed, modified, and maintained. Connection between services, the technical essence of SOA, is realized in the *Service Delivery Network* through which all service interactions take place. Control, which is required to manage services, is enabled by the *Service Control Platform* that provides tools to monitor, govern, and manage service execution. Figure 1 depicts the three function sets surrounding the services that they support. The figures in this section are not conventional computing architectures, i.e., functional components and their relationships. The services surrounded by the three function sets are those services that are active in some service-oriented solution. The Service Repository is a service library and support environment that contains all services that can be used to define services. The Service Repository is supported by a Service Registry with which all services are registered. In this paper we use the term SOA to refer to the sets of functions and supported services depicted in Figure 2, even though no conventional computing architecture is defined.

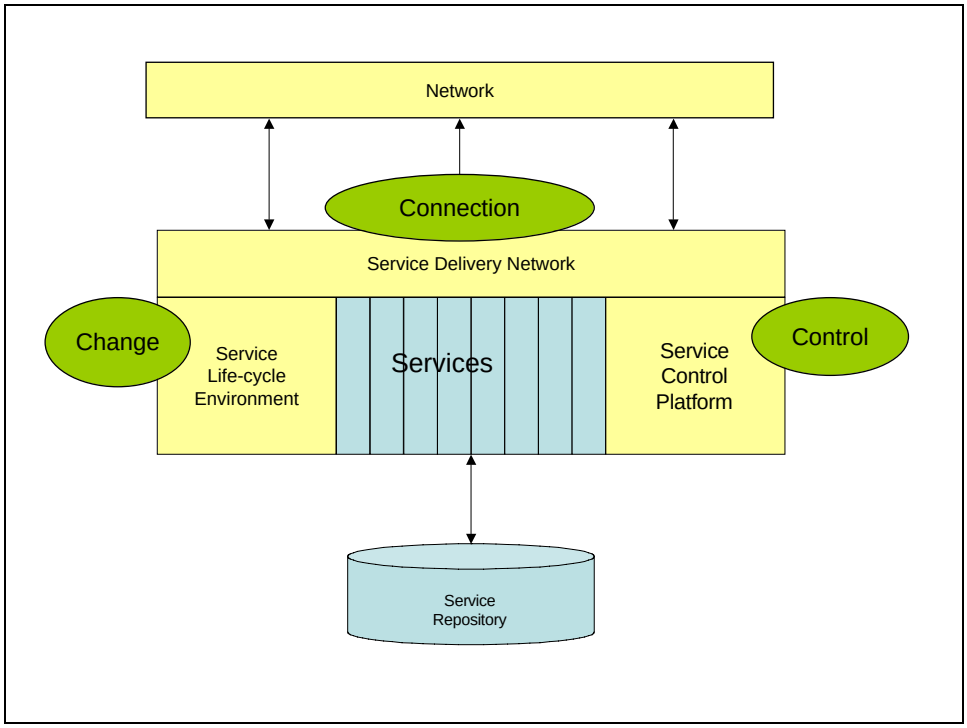


Figure 2. SOA Values and Functionality.

Initially during the migration to SOA, an SOA will mix services with conventional or legacy applications. Figure 2 depicts an SOA that supports pure services. At least initially, SOAs must deal with existing computing resources that are not yet or may never be service-oriented. Figure 3 depicts services as service interfaces that are APIs (Application Programming Interfaces) to functions within conventional applications, called core application platforms that will exist early in the transition to SOA. For the remainder of this paper, we refer only to services. The arguments of this paper apply to services and to conventional applications as depicted in Figure 3.

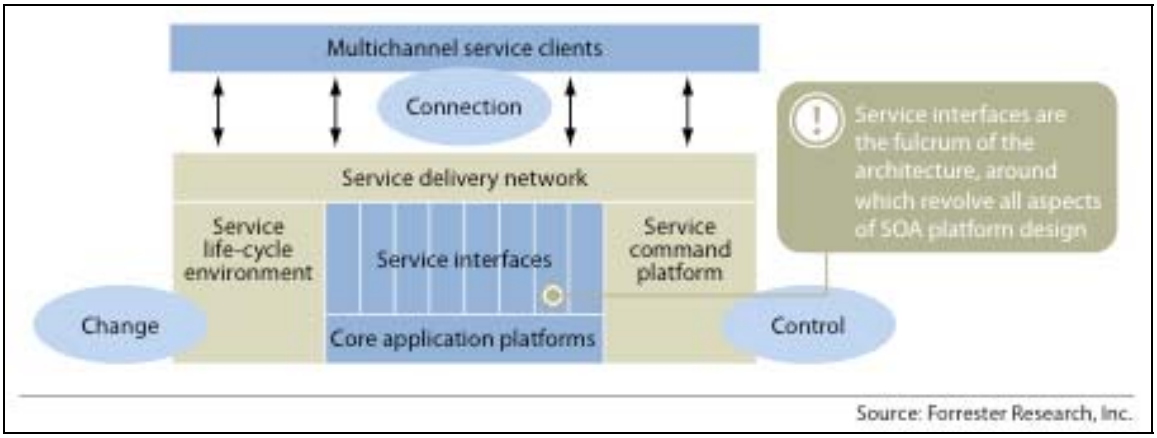


Figure 3. SOA Values over services based over core application platforms.

The SOAs in Figure 2 and Figure 3 appear to be a set of functions and services that reside on a single computing platform. A real SOA will be distributed over many computing platforms. Participating platforms may contain the three function sets and a set of services, as depicted in Figure 4, an SOA Grid consisting of 2 nodes. However, some computing platforms may obtain some or all of their services from other platforms. SOA platforms including their functions and services will be federated, e.g., a global services registry and repository will be implemented as a federation of local registries and repositories. Any one computing platform may have none or all of the functions locally and none or some of the services locally. The SOA Connection functions support transparent distribution and connections not just for end user applications but also for all services in the SOA, including Change, Control, and Connection functions.

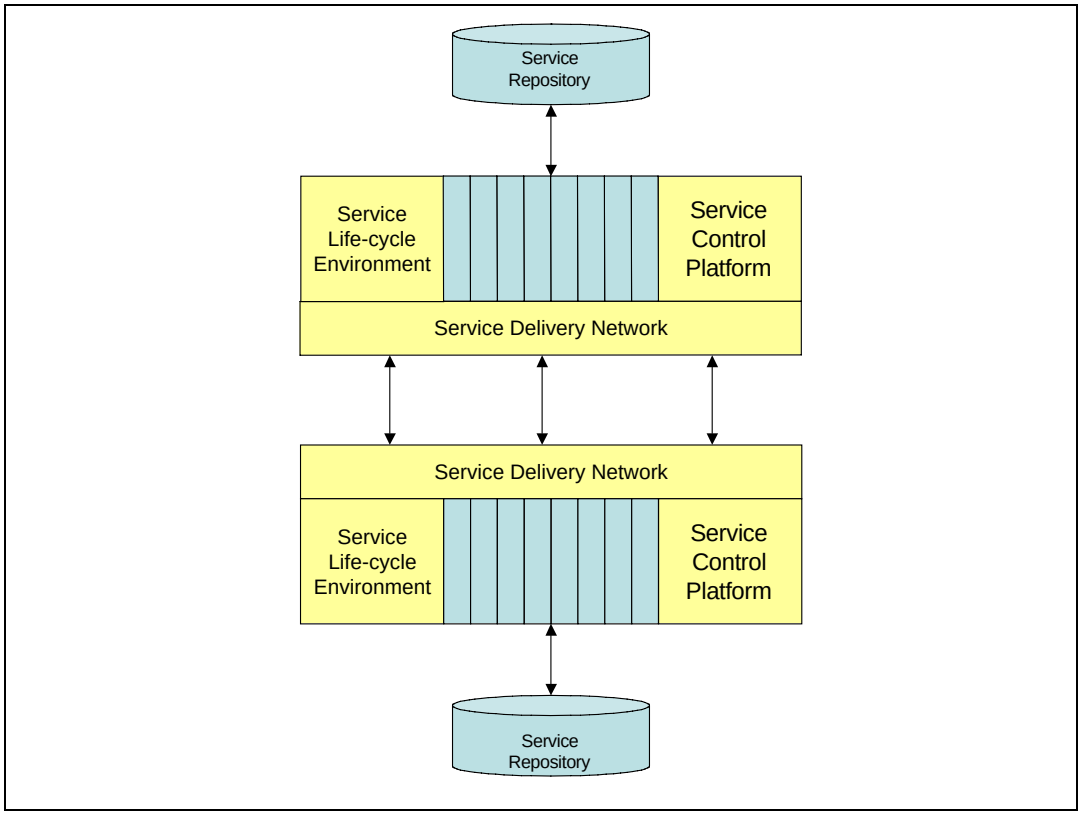


Figure 4. A 2-node federated, distributed SOA or simple SOA GRID.

2.3 Need for a Semantic Extensions of the SOA Framework

The essence of SOA is a computational model based on remote service invocation. SOA is not a computing architecture but a style of programming that has yet to be defined. SOA provides no guidelines as to how services and service invocations are to be used; the granularity of a service; service design; service reuse and so forth. In fact, SOA contains all of the familiar challenges of computer science in the SOA context. Happily, SOA provides a basis for achieving the benefits of the contributions of fifty years of computing. Advances that were previously not fully monopolized, including: abstraction and separation of concerns, i.e. each service can be used to implement a distinct function; *model, pattern and policy driven*, i.e. all services of a specific type, e.g., network management; derive uniform direction from higher level patterns or policies that can be modified to redirect complete domain-specific solutions, such as network management; and integration, i.e. data, protocols, and processes can be mapped uniformly following globally defined mapping definitions. A compelling feature of the new wave of SOA products (see Section 6.3) is the capability to define policies that can be used define many aspects of service-oriented solutions from the service model itself (e.g., instance and transactional behaviour) to business policies in a problem domain such as retail sales in an enterprise.

While SOA provides **a basis** for taking advantage of decades of computer science advances, SOA does not inherently provide those advances or the consequent benefits. Recognizing the need for guidance as to how to “Do” SOA, SOA software vendors and consultants, e.g., IBM, Microsoft, BEA, and Sun, offer evolving frameworks that provide guidance on the design, development, and maintenance of SOA solutions. An *SOA framework* is an SOA populated by models, policies, and patterns that implement and, to some extent, govern the SOA, possibly for specific business (e.g., US Sales Regions) or technical (e.g., network management) domains. An SOA framework could include guidelines for all SOA aspects from the computational models used, service-oriented solution development, the three SOA function sets – Change, Control, and Connection, as well as for each domain-specific problem solution. While it is powerful to define an SOA framework that delineates and governs many aspects of the SOA, it is considerably more powerful for the SOA framework to be dynamic so that the SOA framework dynamically compensates for changing conditions, as could be achieved by semantic enablement of an SOA framework, as described in this paper.

In a service-oriented world, the boundaries of applications may disappear. What is currently achieved by monolithic, inflexible applications may be achieved by “applications” composed, possibly on demand, from other services that in turn are composed from services. Services should be designed so that they can be (re) used in any **meaningful** context. Large legacy applications might well translate into thousands of services. Hence, large enterprises might have millions of services. The idea of a global market of services radically changes current “computing” boundaries of applications and even enterprises leading to global, or definable, SOA environments that will include billions of services. For one service to use or invoke another, mappings must be made between their data structures, protocols, and process specifications. Currently, human programmers are required to define or verify meaningful service mappings. Specification, discovery, and adaptation technology is syntactic, based on the technical specifications of services. Semantics are required to increase the level of automation. Current SOA technology will not scale without semantically based solutions. Let us consider this semantic gap in more detail.

SOA is in its infancy. Since the 1999 declaration by IBM, Microsoft, and Sun of SOA in terms of Web servicess, worldwide research and development has focused on service-orientation. Significant advances have been and continue to be made in technical results and standards (see Section 6.2) focusing on syntactic and engineering solutions to SOA challenges. While vendors want to sell SOA solutions and gain experience from the use of those products and while customers want to take advantage of SOA as soon as possible, SOA technology, standards, products, and methods will evolve over the coming decade. This paper addresses one of the critical gaps in SOA, the SOA semantic gap that must be addressed as an integral part of SOA development.

Two core challenges of conventional computing that are not addressed by SOA are *search* and *integration,* which on its own accounts for half of the development cost of all information systems. Not only does SOA provide a basis for addressing these challenges, SOA significantly and fundamentally depends on solutions to fill the semantic gap to achieve its potential. Consider the following. SOA provides the potential of a global registry in which to search for services anywhere in the network. This is referred to as **service discovery**. SOA provides the potential of invoking remote services to achieve the combined results of those services. This requires that the services interoperate or integrate with respect to their respective data, protocol, and process

syntax and semantics. These actions are called **service orchestration** (or **composition)** and **adaptation** (or **integration).** Service-orientation does not address the challenges of automating discovery, orchestration/composition, or adaptation / integration.

Discovery matches the requirements for a service against the capabilities of all candidate services to find a single service, or composite service, that meets the requirements. In a service-oriented world service requirements include functional and non-functional requirements. Functional requirements of a service require that a match be made based on a meaningful match of the required functions with the capabilities of the functions that will provide the service. That is, the semantics of the functions provided by the service must match those defined in the requirement. The same holds true for non-functional requirements. For service discovery and matching (or selection) to be achieved dynamically, hence automatically, syntactic and semantic aspects of service requirements must be considered. With even thousands of services to consider current search and discovery solutions must be augmented to accommodate semantics. With billions of services, SOA will not work without automated, semantically based discovery. Service selection must be based not only functional requirements but also on non-functional requirements, including the resolution of performance, resource, economic, and other challenges, such as honoring Service Level agreements (SLAs). Each non-functional requirement poses potentially complex challenges in its own domain. Services can be specified in both functional and non-functional terms. Hence, discovery (search) can be based on functional and non-functional requirements. Non-functional requirements matching, e.g., concerning resource usage can be addressed by resource specifications of services and (service-oriented) *resource management* solutions, as discussed later in the paper. Service selection is even more complex since it must also consider orchestration, the composition of services to form a composite service or service workflow. Having matched one or more services that will meet the requirements, services must be selected based on their being able to be composed or orchestrated to form the desired composite service. Can the composition requirements of selected services be met? Does the orchestrated composition make sense? Does it meet the original requirement? And can the services be adapted or integrated meaningfully?

Service integration or *adaptation* maps service protocols, processes, and data so that the meaningful interaction of discovered and orchestrated services is itself meaningful. That is the protocols, processes, and data of one or more services are mapped (or adapted) so that they are meaningfully mapped to those of the service with which it

must interoperate. Integration, an open problem of conventional computing, hence one of its greatest costs, led to SOA but is not resolved by SOA. SOA provides the computing context within which integration or adaptation can be resolved. Unlike conventional computing in which integration is done largely manually between large software components, integration must be resolved largely dynamically between millions of services. This underscores the critical role of semantics in SOA.

The simplest way to consider service discovery (search), selection (match), orchestration (composition, choreography), and adaptation (integration, interoperation) is within the Change function as tools to assist human programmers. This is currently the case in most SOAs in which programmers use browsers to discover, select, orchestrate and adapt. Indeed this would continue to be the case in a SOA that is not enabled by semantic solutions. SESA includes these capabilities to underly all aspects of Service-Oriented Computing including Change, Control, and Connection. The semantically enabled solutions will dramatically increase the level of automation of discovery, selection, orchestration, and adaptation so that they can operate dynamically to support dynamic business flexibility and adaptation. SESA does not enable this dynamism; it provides the technical basis for addressing it in the semantic domain but within the technical limits of computing.

For SOA to achieve its benefits in a reasonable scale, SOA requires discovery, orchestration / composition, and integration / adaptation to be meaningful and dynamic. We call such an SOA a SESA. We call the lack of these capabilities in an SOA the ***SOA semantic gap***. The SOA semantic gap must be closed for SOA to reach the full potential of service-orientation. To close this gap, SESA semantically enables SOA with the following technologies.

The **Web Service Modeling Ontology (WSMO)**[3] ([Roman et al., 2005]) provides a conceptual model for adding semantics to service-oriented solutions including Service-Oriented Architectures. Its main elements are goal definitions of user, service definitions of providers, and ontologies and mediators as declarative and procedural means to facilitate interoperability at the level of data, protocols, and processes.

[3] http://www.wsmo.org/

The **Web Service Modeling Language (WSML)**[4] ([de Bruijn, 2005]) is a family of languages providing formal semantics for WSMO models. Its four major dialects form a lattice based on rule languages and descriptions logics as well as on their minimal and maximal intersection.

The **Web Service Execution Environment (WSMX)**[5] ([Haller et al., 2005]) is a reference implementation of an SESA that is compliant with the semantic specifications of WSMO. WSMX supports semantically enabled change functions such as dynamic discovery, selection, and mediation. WSMX also implements semantically enabled control and connection functions such as service invocation and inter-operation. WSMX is an execution environment for the dynamic discovery, selection, mediation, invocation and inter-operation of the Semantic Web Services in a reference implementation for WSMO. The development process for WSMX includes defining its conceptual model, defining the execution semantics for the environment, describing an architecture and a software design and building a working implementation.

The set of concepts developed by DERI to realize semantically enabled **Triple-Space Computing** is based on a Linda-like ([Ahuja et al., 1986], [Carriero and Gelernter, 1989]) publish-subscribe, globally accessible information / communication space. Currently, Web services require close coupling with the applications that they integrate. Applications communicate via message exchange requiring strong coupling in terms of reference and time. The communication has to be directed to the Web service addressed and the communication must be synchronous. If both parties do not implement and jointly agree on the specific way this mechanism is implemented, then the applications must support asynchronous communication. The web is strongly based on the opposite principles. Information is published in a persistent and widely accessible manner. Any application can access this information at any point in time without having to request the publishing process to directly refer to it as a receiver of its information. While Web services use the Internet as a transport media (relying on protocols such as FTP, SMTP, or HTTP), that is all they have in common with the web. DERI is developing a semantically enabled triple space as described in [Fensel, 2004], [Bussler, 2005].

[4] http://www.wsmo.org/wsml/
[5] http://www.wsmx.org/

These semantic descriptions need to be grounded in existing syntactic means for describing services. SOA services, resources, and computing platforms can be syntactically described using evolving standards such WSDL, WSRF, and OGSA. The *Web Service Description Language (WSDL)*[6] provides means to specify services via the definition of endpoints and operations. The *Web Service Resource Framework (WSRF)*[7] provides for a standardized description of stateful resources (services), i.e., it defines conventions for managing state so that applications discover, inspect, and interact with stateful resources in standard and interoperable ways. Finally, the *Open Grid Services Architecture (OGSA)*[8] represents an evolution towards a Service-Oriented Architecture based on Web services concepts and technologies. Most SOA specification standards, including those above, provide syntactic means for specification. Hence, we call these specifications the syntactic technical kernel of the SOA vision. A SESA is a semantically enabled SOA that includes semantically enabled SOA change, connection, and control functions that in turn enable all SOA aspects with semantics. We propose extending the syntactic technical specifications underlying the SOA vision with means to ground semantic specifications.

[6]http://www.w3.org/2002/ws/desc/
[7]http://www.globus.org/wsrf/
[8]http://www.globus.org/ogsa/

2.4 Implications of our Approach

In this section, we sketch some implications of our approach for the three core elements of a future computing architecture: resources, services, and problem solving. In addition, we discuss what our proposal could mean for the economics of information technology.

2.4.1 Resource Management

In conventional computing, resource management refers to the management of hardware resources such as network elements and involves the optimization of hardware resources, by means such as resource allocation, monitoring and capacity planning. Service-oriented resource management is orders of magnitude more complex than conventional resource management for several reasons.

(1) The usage of markets for the discovery, allocation, use, and management of services introduces the full complexity of economic activity for these activities. This includes various mechanisms for determining the price and remuneration for a service, e.g. auctions, tendering, reverse auctions; or incentive and agency conflicts (e.g. contractual agreements that guarantee that the incentives for the provider of a service and the consumer of the service are in alignment in order to avoid agency conflicts). This is a fundamental difference with SOAs that are employed within an enterprise for a defined context such as Enterprise Application Integration or supply chains. Another factor is the scope of these actions that take place in open SOA environment. It makes a significant difference if these actions take place in a small part of the company or across the entire enterprise. These issues become more critical as businesses automate greater amounts of their business models, to the point that the enterprise becomes identical to its automated representation of its business model.

(2) The role of policies in a service-oriented environment will grow exponentially, since preferences, business strategy, and trade-off decisions must be supported on various levels. Policy matchmaking in today's SOAs is limited to a binary verification that required characteristics, such as security, are met. Such elemental policy management may be adequate with few services and in which it is a major challenge to simply identify a suitable service. However, as the number of suitable services grows and as the capabilities and requirements become more sophisticated, binary matchmaking must be extended to include, amongst other things, ranking of matches and the inclusion of imperfect matches ([Noia et al., 2003], [Hepp, 2005]). These extensions require sophisticated support for the relevant semantics.

(3) In the history of economics, the use of markets for the exchange of resources has always fostered specialization and the division of labor. We can expect the same

for the market of resources in a networked economy implemented that adheres to the SOA principle. The specificity of resources will grow dramatically and thus the need for automated discovery of substitution relationships between offerings.

Resource management in an SOA environment can include many aspects such as a grid of computing platforms, complex distribution and federation requirements, vast numbers of services, and complex, and multi-platform executions of service flows. This new level of complexity is orthogonal to the technical complexity of SOA operations, e.g. the mediation between different message format or service choreographies.

If we lift SOA from an approach for solving intra-organizational integration problems to become the fundamental paradigm of computing in a business context, then the scope of what a resource is expands dramatically from core IT services into every basic activity (i.e. process fragment) or complex business process that can be described. Especially, services on all levels of activity, from basic technical functionality to services on the business perspective can be represented in the very same framework, which means that all aspects of a business problem can be included in the problem solution.

Hence, resource management can be extended from the management of hardware and computing resources to the management of all relevant resources. That is, computer resource management can be integrated within the broader real world domain of resource management. This requires, as with trust domains, that every resource be specified in terms that are necessary for resolving resource management problems and the resource management problem solutions that use those specifications. As with all other aspects of SOA, an SESA will provide semantically enabled capabilities for addressing resource management, as discussed later in the paper.

2.4.2 Architecture as an Emergent Property

In conventional computing a computing architecture of a software component, typically represented in an architectural diagram, consists of all constituent computing components at a given level of detail and the relationships (interfaces) between them. Typically such an architecture is determined not only at compile time but remains fixed for extended periods of time. Replacing an architectural component is a relatively rare event.

In Service-Oriented Computing, components correspond to services. For a given computation, discovery is used to find services that match the functional and non-functional requirements of the computation. The selected services are choreographed and adapted to meet the requirements and context of the requested service. In the initial stages of SOA programmers, supported by the Change function, will do discovery, choreography, and adaptation, during development. This process results in an architecture of services – a set of software components and their interfaces required to execute the computation.

An SOA, enhanced by semantics, would permit a degree of dynamic discovery, choreography and adaptation. Hence, a Service-Oriented Architecture would be *emergent* ; a service architecture would emerge dynamically depending on the policies and models that govern the relevant service. For example, cost and performance policies may cause the components of the service architecture to be changed whenever the relevant conditions change. Alternatively, a policy may ensure that a specific service, a reasoning component in WSMX, remain fixed.

Hence in Service-Oriented Computing, the architecture of a component or service, *S*, consists of the services required to execute *S* and the relationships (interfaces and invocations between) amongst the component services. As with all Service-Oriented Computing, service architectures are extremely flexible. Rather than being pre-defined and fixed over long periods of time, as in conventional computing, the architecture of a service is defined in terms of the requirements specified for the service and the discovery, selection, choreography, adaptation processes applied across all relevant services. These processes, hence the architecture, may also be governed by models, patterns, and policies. For example, reliability and availability policies may strictly limit changes; or patterns may define entire architectures that have been proven to work well in specific

contexts.

Augmenting SOA with semantics dramatically enhances the power of the SESA to specify, discover, select, choreography, and adapt services and indeed entire service-oriented solutions, such as entire architectures as described above. For example, SESA could enable architectures that respond dynamically to circumstances but within defined polices and patterns, i.e., policy-driven architectures.

2.4.3 Example Problem Solving Scenario

In order to describe problem solving within a semantically enabled service-oriented approach we take a business scenario from the liveliest context in e-commerce, i.e. e-tourism ([Werthner, 2003], [Werthner and Ricci, 2004]). Travel and tourism is among the most important application domains in b2c e-commerce (estimates range from 35 to 45 % of the total turnover on b2c e-commerce). For example, in the EU15 82% of accommodations have a Web site (over 30% selling already online). At the demand side one Billion international arrivals in the year 2010 are estimated. Structurally the supply and the demand side form a worldwide network, where both production and distribution are based on cooperation. The wide diffusion of e-commerce and the fast technical development – within 6 years a development from online presence over transaction support to customer retention (3 system generations) – led to an "informatization" of the entire value chain as well as to a changed information, booking and travel behaviour. One can observe an Internet based integration of processes, with a focus not only on process reengineering, but also on network engineering.

Looking into the future one can expect flexible network configurations (cooperation) and the further integration of consumers into (internal) business processes. Adding the tourist life cycle one can draw the following figure of linking tourists' life cycles (taking into consideration the mobility aspect of travellers) with companies' business processes.

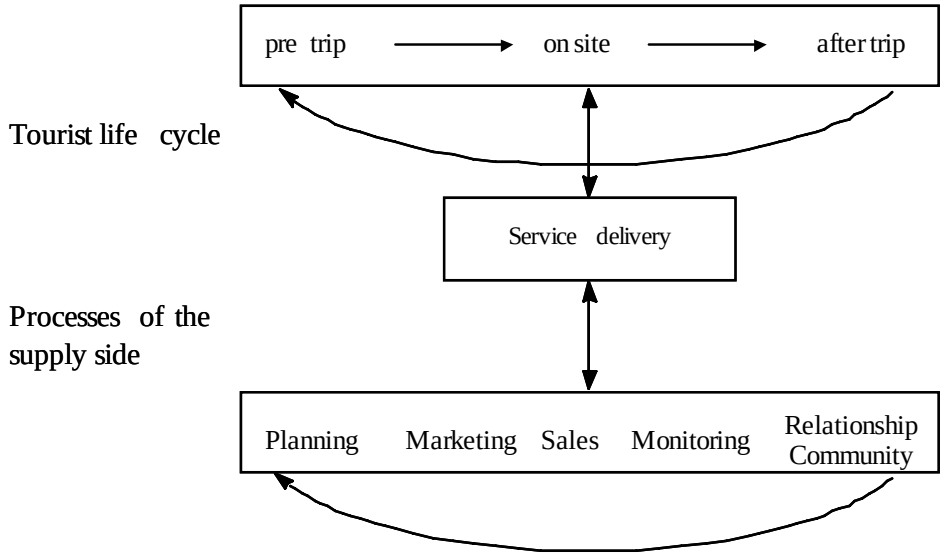


Figure 5. Tourist life cycle and companies' processes – both suppliers and intermediaries.

In this context processes cross company borders, leading to distributed b2b2c applications, requiring cooperation between companies, and supporting also mobile interaction with the consumer. Such a scenario is based on the assumption that technology – based on a common pervasive infrastructure – will become transparent, invisible for the consumer; information will be available at home, the work place and during travel – travellers having access to information whenever and wherever they want.[9] Applications will guide them in the decision and travelling process and allow users to create (ad hoc) product bundles. This requires scalable and flexible IT solutions, providing seamless integration and interoperability (between all stakeholders).

For example, a customer is planning a trip to the city of Vienna and wants to include going to a classical concert, visiting restaurants with at least 20 gourmet points, and booking a hotel with specific features (all within defined budget limits). This combines pre-trip planning with an on-site support (using the user model of the pre-trip planning module). The "application" provides additional multimedia information on the concert in the form of videos and audios, or on the conductor. It gives hints on on-site alternatives, recommends similar events, or it proposes the use of specific transport means in Vienna. Such a vision presumes the access to different information resources

[9] This reflects to the IT research vision "Ambient Intelligence" of the European Commission.

and services, the combination of components independent of their physical location, “just” guided by user (business) needs. The vision is that this should be available (semi-) automatically. Such a service composition steps may look as follows: in a first step identify the means of transportation. These have to be filtered according to their availability, price and so on. Then one needs to find – step by step – compatible services (e.g., same destination and time), according to the user preferences. However, different alternatives may be maintained, since, for example, the overall price may depend on the entire set of services (i.e., bundle), which can only be obtained at the end of the bundling process (different pricing schemes may depend on the concrete bundle). Obviously, services need to be described with some meta-information, guiding this process, and discovered in an ad-hoc manner (e.g., hotel availability). And in a mobile environment a centralized registry of services may not be feasible.

However, when taking the supply side’s point of view, the requirements may become even more complex. This describes the process of dynamic packaging, assuming an iterative approach in the bundling of basic products (triggered either by the system, the user or both), with the need to access legacy systems. In this case functions are needed to enable middlemen to create sets of bundled products (involving different suppliers), to ease the participation of SMEs to larger networks (taking into consideration their very different IS structure), and to facilitate the already described networking. For instance, a middleman (e.g., tour operator or cybermediary) tries to create 1000 packages with a maximum cost of 450 € each, for a specific date (e.g., 05/12/05 – 15/12/05); hotel service should be half board and they should be in Tyrol and bundled with ski-lift tickets. In addition, the hotels are offered together with a flight (thus, flights need to be “interfaced” with the hotels). The middleman has contracts with some of the providers (but not all), and he starts communicating / negotiating with some of them (based on his past experience). In addition, he as well as the contacted suppliers follow specific business rules (e.g., supplier: minimal occupancy / price; middleman: contract rules with trust levels, preferred partners, and his utility function). Such networked business operations transfer the level from composing individual instances of services to composing sets of services; and from ad-hoc integration between two participants to an arbitrary number of cooperating enterprises. The related orchestration requires flexibility for business planning; e.g., techniques of constraint reasoning, multi-value optimization and relaxation might be used to aggregate services and to achieve specific business goals (profit optimization or the equal distribution of income within a tourism destination).

These two scenarios of mobile consumer interaction and dynamic network configuration exemplify the need for semantic enrichment as well as operationalization of semantics. In addition, one may also distinguish between two general “cases” of challenges: the support for “ad hoc” creation of business solutions with support for flexible internal and/or external cooperation; and the “automation” of software engineering tasks to create solutions for given problems. In both cases, a semantically enabled service-oriented approach will support people (or systems) in defining and creating problem solutions which are not only specific to their domain, but probably devoid of computing concepts, terms or artifacts. Service-Oriented Architectures have to be turned into a domain specific problem solving environments, where computing resources, expressed as services, are used effectively and efficiently. And the user should be provided with a transparent interface to the available services as well as set of already existing solutions. In addition, such an approach should also support not only a “machine distributed” environment, but also a distributed work and business environment.

One should note, that this approach implies a transformation of meanings (for the user), from services as they are understood in management science to Web services as defined in computer science. In management science a service is defined as a business economic activity (mostly intangible in nature), offered by one party to another to achieve a certain benefit ([Zeithaml and Bitner, 1996], [Kotler, 1988]), and “generated” by (internal) business processes. In IS a service is a complex (or simple) task executed (within) an organization on behalf of a customer ([O’Sullivan et al., 2002]). In this sense the vision implies the separation of business / process logics (expressed as a workflow or other process description) from the Web services used (as well as the respective mappings), and where the created set of Web services correspond to the implemented (business) solution. However, one should also note that service bundling – using a business term – differs from service composition ([Akkermans et al., 2004]): composition assumes a process description, whereas bundling do not make explicit assumptions about time order, but about service connectivity or puts constraints on service configuration, e.g., bundle of services with overall minimum price. This puts emphasis on non-functional aspects of service descriptions.

The services – needed to build and to execute a domain specific solution – are provided by a set of underlying and reusable common services. Hence, our vision of Service-Oriented Computing distinguishes Problem Solving using service-oriented solutions for domain specific problems from Common Services from which the solutions

are composed. The respective composition will be guided by the domain specific business and process logic, using tools and capabilities of the Services Life-cycle Environment (i.e., change, control, connect). In this context we foresee, that problem solutions will evolve dynamically based on semantically enhanced service specifications and corresponding semantic discovery, selection, configuration, choreography and adaptation capabilities (e.g., if a lower level common service becomes available and is a better match for one already in use, it could dynamically replace the old common service). This leads – based on semantically enhanced service descriptions - to a flexible, decoupled world of independent services, supporting business flexibility and adaptation.

2.4.4 The Economics of IT [10]

Economic factors dominate much of modern computing, particularly in large enterprises where most decisions are based on economic grounds, second only to the business objectives and strategies. Yet financial and accounting methods used to support economic decisions that are ultimately used to run information technology (IT) shops are imprecise and fraught with errors and approximations. What is the cost of developing and maintaining an application including software license and acquisition costs through the cost of each step in the software life cycle? What is the cost of supporting and delivering the services of that application to all consumers, including opportunity cost or lost revenue when the application is not used or when there is insufficient computing resources to meet the current demand? How much increased revenue could be obtained if the application could be made available to others whose requirements might be met by the application or some minor variation of the application? Is an individual application making a profit or a loss, i.e., does the business value generated by the application warrant the total cost of ownership of the application? If there is no service currently available in our enterprise that meets our requirements, should we look outside the enterprise for the service? Indeed, what is the Total Cost of Ownership of any application and does future demand and projected business value warrant enhancements required to support the generate new business?

Economic issues such as supply, demand, cost, opportunity cost, and, more generally, creating and managing an automated market for IT services, pose significant challenges in conventional computing. Total life cycle cost of an application is seldom precisely computed, let alone used in setting the price to the consumer of using the application. Seldom are prices of consumption set for applications except at an extremely high level, e.g., all billing services for a year, based on Service Level Agreements (SLAs). Even then, precise monitoring of SLAs is a challenge. It is equally challenging to make precise estimates of the cost of operating and maintaining an application as a basis for charging consumers, e.g., per unit of consumption. While there are sophisticated methods and software tools to monitor performance, and make adjustments under what is called configuration or resource management in support of capacity planning, these methods are largely passive in that much of the information used in their decision making

[10] We own much of the clarity on economics and none of the errors to Pia Koskenoja, Professor of Transportation Economics, Tampere University of Technology, Tampere, Finland.

is static as opposed to dynamic, reflecting the exact state of the computing environment and all of the competing demands. In simple terms, conventional computing operates as an accounting solution that attempts to optimize resource utilization based on extremely limited information and largely after the fact.

SOA offers the potential to fundamentally change the methods of and precision in dealing with economic issues. This potential is based on several SOA principles. Two fundamental SOA economic principles are: (1) the producer-consumer model of computing, and (2) every hardware, software, and possibly human resource, is a service. Every service is described in terms of functional and non-functional characteristics. At the current state of SOA infancy, these characteristics are very limited by SOA standards bodies and are much richer in any operational SOA such as Verizon's ITW [Havenstein, 2005]. The richness of SOA computing model is being tapped by large-scale SOA implementations that augment their registries or directories with whatever information aids in developing SOA. Service descriptions can include any information about the service such as assigned or computed cost to produce, service levels promised under specific conditions, availability and capacity, projected demand, total cost of ownership, performance level per consumer by SLA, and much more. The two economic principles -- the producer-consumer model of computing model and every resource is a service whose description is accessible dynamically in the global registry – bring about the potential of an automated marketplace. Automated marketplaces have been in existence for some time in logistics, automobile manufacturing, and retail consumer markets. In current automated marketplaces, trusted strategic partners, such as those in a manufacturing or distribution supply chain achieve leaner material flows by exposing their systems to each other. For example, a windscreen parts supplier observes the flow of work on the factory floor and delivers just on time the windscreens needed for the current automobile, eliminating the need for the automobile manufacturer to have an inventory of windscreens. SOA will permit automated marketplaces at a much deeper level and across all industries, as can be seen in the discussion of the producer- consumer (i.e., supply and demand) model below.

Producers create services that they publish in a global registry, making them available as widely as they have defined. All services are characterized in terms of functional and non-functional characteristics. Non-functional characteristics can include cost and related charge-back rules, service levels under which the service is offered, penalties for failing to meet SLAs, restrictions and specific conditions for usage, and

anything that the producer might want to record privately or publish. While publishers can characterize services for the use of all potential consumers, publishers can also characterize services privately for their own purposes, such as the evolving total cost of ownership, estimates of opportunity cost when their service is under utilized, number of current consumers and projected usage, usage by specific consumers that warrants monitoring, service availability gathered dynamically by monitoring and management tools, and much more. As SOA develops, standards will evolve to support common models – global, industry-specific, or enterprise specific, e.g., Wal-Mart and P&G (formerly Proctor and Gamble) often set their own.

Producers can use all the information about a service in reasoning about the management and evolution of the service. Consumers can search the global registry for services that meet their functional and non-functional requirements. Just as the production of a service is complex, so is the selection of a service or the choice of competing services and requires considerable information and reasoning skills.

The global, federated SOA system is, in part, an automated marketplace. By the nature of the SOA model, the global registry, a federated system of distributed registries knows all requests for services as well as the state of all services that declare their state to the system or make their state available via agents or other monitoring technologies. Hence, the SOA system knows all supply and demand and attempts to mediate between supply and demand.

A producer-consumer marketplace of services offers considerable potential for increased precision and automation of economic decision-making in computing, as well as significant challenges and opportunities. All services can be active, continuously publishing or making accessible their current and changing status.

First, consider trust. In principle, the more information shared about a service between the producer and consumer, the greater the level of optimization in its consumption. Yet information sharing depends on the level of trust between the producer and consumer. As described elsewhere in this paper, research is being conducted into trust domains and business networks in which all members of a given domain share a given level of trust and can act freely and openly within that level. Is there complete trust between all organizations within an enterprise? Or are well-defined levels of trust and subsequent behaviour similar to the proverb that “good fences make good neighbours”?

In any case, the legal system can be used to offer remediation in the case of violations of agreements, automated or otherwise.

Second, consider the potential of creating markets. While there may be domains of trust that partition the global SOA community, a producer or set of producers can in principle reach all consumers via the global registry. This vastly expands the conventional consumer base. Enterprises' normal practice of selling IT services internally is highly redundant and inefficient. The SOA benefits of re-use that motivate SOA for enterprises should cross enterprise boundaries. Hence, combined with the Software-As-A-Service trend by vendors, such as Salesforce.com, the ability of publishing services globally will permit the constant creation and re-creation of markets, and at a minimum significantly expand the potential consumer base for any service so offered.

Third, consider opportunity cost – the cost for a resource is the highest monetary value that that resource could fetch if it were not used for the purpose for which it is actually being used. Prof. Koskenoja's definition (above) and comment that "while crisply defined in theory [opportunity cost is] almost impossible to measure in practice.' is the bane of conventional computing. While there are sophisticated tools for configuration and resource management and capacity planning, enterprises seldom know, with any precision, the total demand for a resource or its availability, in such a way as to maximize the use of all resources. While hardware and software heterogeneity is a significant culprit, current solutions simply do not deal with demand or cost / value let alone in an automated market such as is possible in SOA. Another potential in addressing opportunity cost is that of changing service offering costs, e.g., a sale, as well as consumption offers, e.g., a negotiated cost reduction in a weak market. This leads to the final consideration for this section.

Fourth, consider a wealth of economic reasoning normally done by humans, possibly augmented by computing. Consider negotiating producer-consumer agreements, discovering a service that partially meets the requirements but may be "good enough", consider policies to determine economic decisions under specific conditions, and finally consider compensating for changes in conditions when the requirements for one or more parties to an agreement are not being met, but all parties would prefer to proceed under changed circumstances than terminate the agreement. All of these forms of reasoning require significant semantic capabilities beyond conventional computing. While automated solutions are by no means promised or projected, an SOA augmented by

semantics, an SESA, as proposed in this technical report, will provide powerful tools to further automate economic reasoning in an automated marketplace as will arise in SOA.

The reality of automated markets enabled by SOA and semantics is a wonderful vision but is a long way off. SOA itself is in its infancy. This paper proclaims a manifesto to augment SOA with semantics to achieve SESA. But exciting research is required in economics and related areas. According to economists it will take considerable time for SOA- (or SESA)-based automated markets to work as reliably as conventional markets. One example referenced earlier, it that of trust between service producers and consumers. For example, the economic principle of *credible commitment* [Williamson, 1985] involves an enterprise determining whether it should produce a service internally or purchase the service externally. At issue is the risk of the external purchase, characterized by some transaction cost, and whether it can be reduced by some procurement procedure. One form of credible commitment is the service producer's good reputation the value of which may involve continued business with the consumer and with others. Hence, the service producer has a vested interest in maintaining their reputation and looking after the interests of the consumer, e.g., protecting the information that the consumer gives in the conduct of the transaction. This credible commitment lowers the consumer's risk or transaction cost. This issue of trust will distinguish a service producer from his competitors, those who produce an equivalent technical service. Since producing services internally is not risk-free, the issue of credible commitment also exists within an enterprise. The issues raised by one economic principle, credible commitment, pose challenges to automating markets. To what extent could the purchase of a service by an enterprise be automated so as to accommodate the relevant economic principles? As with all such questions, there will be simple, sub-optimal solutions, e.g., manually defined trusted providers. If however, automated markets represent a significant advantage, then automating, to some degree, economic principles, such as credible commitment opens the door to semantically enabled solutions.

Automated markets are already making modest beginnings in the SOA world. We anticipate that automated markets will become powerful management and economic tools in the next decade during which we will see real world economic behaviors manifesting in SOA-enabled, even SESA-enabled automated markets.

3 Problem Solving Layer

The objective of the **problem-solving layer** is to turn a Service-Oriented Architecture into a domain specific problem-solving environment. Following the "layered" approach of our vision the problem solving layer represents the transparent interface to the user(s), where we assume that all computing resources are turned into or expressed as services. In order to provide solutions for distinct business problems – from an IS point of view – the problem solving layer has to support the entire e-commerce framework as described in Figure 6. The objective is efficient and effective "resource allocation" for an enterprise or a set of cooperating enterprises (see also the discussion in 2.4.4.).

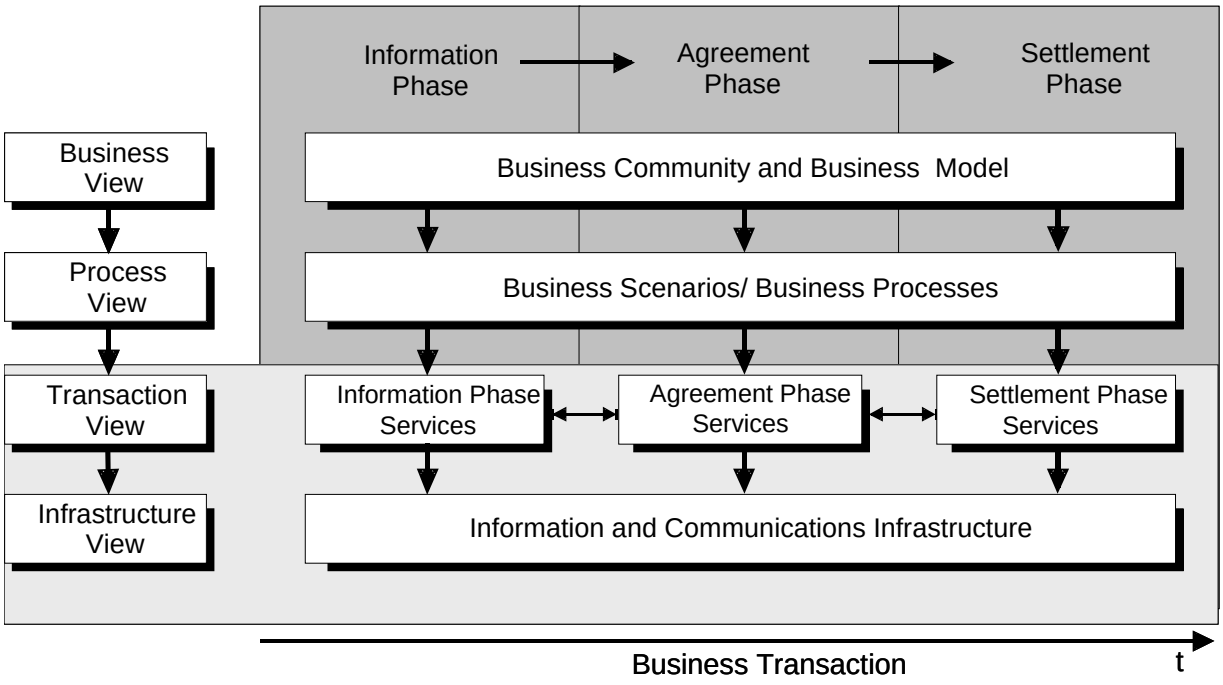


Figure 6. e-commerce framework.

It has to support the information, negotiation and settlement phases of transactions, with different negotiation and contracting possibilities. In this sense it also implements a domain specific economic model, where services would be accompanied by specific functional and non-functional "parameters". The architecture should support the implementation and operation of so-called smart business networks, on the level of

flexible e-business cooperation [Vervest et al., 2005]. They will be based on flexible and loosely coupled network configurations, supporting integrated (cross-) company process integration (including final consumers). There will be a fast connect and disconnect, or otherwise stated “pick, plug and play”. Obviously, such a network can be either “internal” or external/“collaborative” (see section 2.4.3), which imposes that services are most probably provided by different suppliers. The described flexibility (meeting the changing needs of a business / set of businesses) can be achieved by providing a clear separation between the business / process logic and the Web services used. This is the objective of the business / problem solving layer – in effect, the two upper layers of Figure 6.

The approach should support the modeling and implementation of a (collaborative) business model, which is defined as an architecture for product, service and information flows, including a description of the various business actors and their roles (‘organization of the business’); and it contains description of the sources of revenues [Timmers, 2001]. In addition, since no network of businesses operates in an open environment, the vision needs to enable trust domains in which all services are defined in terms of their trust levels and capabilities. Such a business model (based also on a domain specific economic model) as well as trust levels will drive and govern the design and implementation of a specific solution (selection, adaptation and orchestration of services). This must be based not only on functional requirements but also on non-functional requirements covering business and trust aspects (covering issues such a price of a service, type of payment, performance, reliability; or also security levels, authorization, and past history). These have to be included in the description of a service as well as in the requirement definition for service selection. These descriptions should also distinguish between customer and supplier-oriented point of views. They have different roles and views, which imply different ontological commitments [Akkermans et al., 2004]. These requirement functional and non-functional descriptions are the basis of contracts between the respective business partners (or also internal departments), which have to be translated into SLA on the resource level.

Thus, the vision foresees the need to model domain and business logic specific knowledge, plus the respective reasoning means, i.e., a) the design and reasoning over a domain specific ontology (for the domain of the problem area); b) a business (process) modeling ontology (for defining and controlling the business model), including the modeling of business services and transactions; and c) their (semi) automatic mapping

into and generation of business solutions enabled by Semantic Web Services. In other words, formally expressed business (process) models and goals may be translated into a type of service bundling or planning description, which then governs the process of service selection and composition, and finally execution. This is expressed in Figure 7, where a smart business network as a set of cooperating companies where its business objective is the starting point for generating and executing the respective set of services / resources.

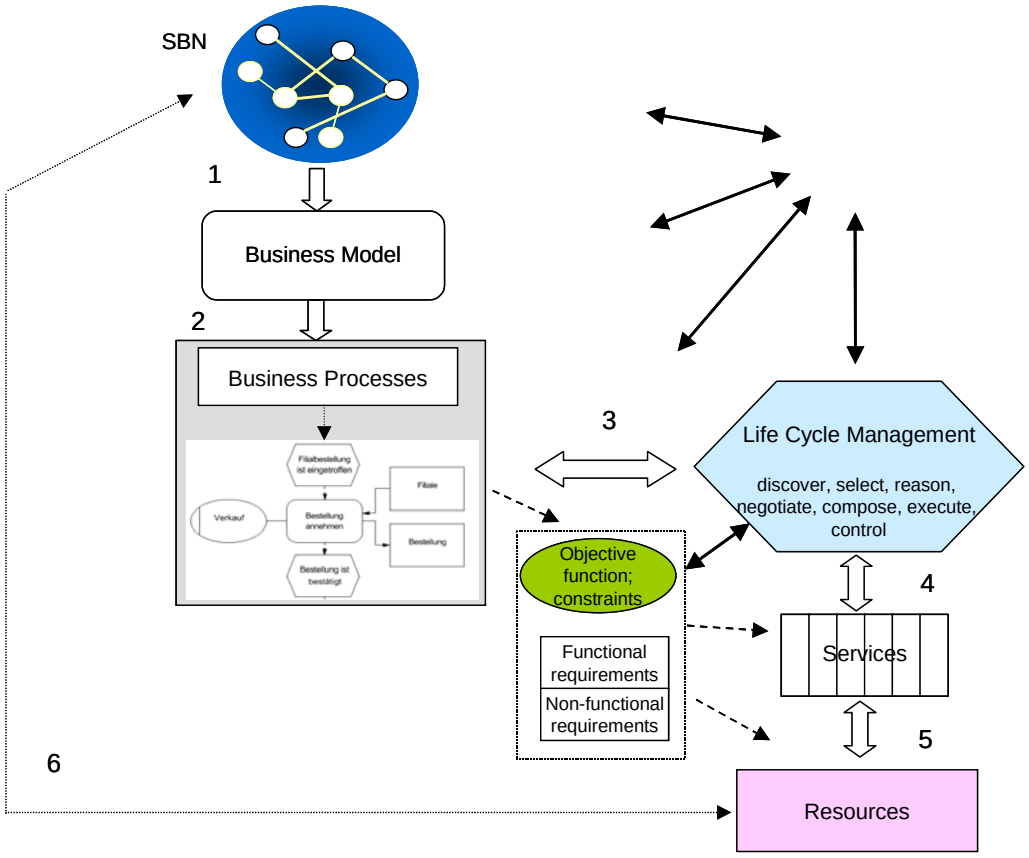


Figure 7. e-commerce framework.

This vision raises some hard research challenges:

- The mentioned mapping of a business model (or in other terms of the business notions of a user) onto the internal Web service “implementation” needs several mappings as well as (modeling) tools, such as business model and service ontologies ([Osterwalder and Pigneur, 2002], [Gordijn et al., 2001]), domain specific ontologies,

business process modeling languages (BPEL4WS), workflow models, business rules or mappings from available descriptions such as EPC by means of EPML ([Mendling and Nüttgens, 2003]).

- Service selection must be based on functional as well as on non-functional requirements, these requirements expressed at the "problem" level need to be translated into Service Level Agreements at the resource level (SLAs). An additional issue is how semantic descriptions (both functional and non functional) can be combined and aggregated on a higher level.
- Service selection is even more complex since it must also consider service bundling and composition. Having matched one or more services that will meet the requirements, services must be selected based on their ability to be combined to form the desired composite service (this will be an iterative process). In this process the customer's and the supplier's view point have to be matched, and the overall business objective has to be taken into consideration.

From the user's perspective – assuming that initially solutions will be created (semi-automatically) by humans – two major roles and related tasks can be identified:

- Problem solvers: they may use specific problem solving tools and already existing solutions. This includes the execution, management and adaptation of a specific problem solution (consider the case of a new service or a new business "participant", either a company or department within a company); users may select, modify and monitor their solution. In consequence, the problem solving layer may consist of a set of such solutions and their management.
- Developers (meta-problem solvers): they develop solutions for the problem solvers, it includes tools and means to create such a solution, i.e., to design, develop and test (or simulating), using the discovery, change, adaptation and composition functions of the underlying common services.

As a starting point may serve a **semantic desktop**, see, for example, ([Decker and Frank, 2004]), MITs Haystack ([Haystack, 2003]), or approaches like Gnowsis and Fenfire. At the moment these applications are focused on personal information management, where the overabundance of information (in the form of e-mails, [multi-media] documents, ..) is managed by treating all this information like semantic web resources (providing structuring and semantic annotation). In addition, such desktops integrate existing applications and extract information on the fly from existing

applications. Developers can also extend the framework and integrate other data sources.

The challenge is to extend this approach a) to support the composition of Web services with filtering and selection support (see, e.g., [Sirin et al., 2004]) and b) to integrate even the entire life cycle of a solution with steps such as design, development, test, deployment, monitoring and modification. All the steps as described in 2.4.3. when composing Web services in order to create a bundle of real-world services need to be supported (e.g., with a service configurator and a workflow editor). In each of these steps semantics play a role and need to modeled (describing the input, output or also their effects). This may also include a simulation or test phase with partial execution.

4 Common Service Layer

As computer science moves to the next period of abstraction, the practice of developing software applications evolves to the modeling of semantically annotated services that can be composed, i.e., can co-operate, to achieve specific tasks. This leads to a flexible, decoupled world of independent services that can be dynamically discovered, combined, and invoked. The *common services layer (CSL)* provides an adaptive execution environment and supporting infrastructure that maps the problem descriptions generated at the Problem Solving Layer to the services that can solve the problems.

The Execution Environment at the heart of the CSL requires components to map problem descriptions at the problem-solving layer to available services at the CSL. Existing architectures (e.g. Open Grid Service Architecture (OGSA) in the Grid area) already support such mappings for components and prototypical interactions, however they operate over purely syntactic descriptions, hence domain specific problem solutions must be coded manually. Besides providing the interpretation of semantic description the CSL needs also be able to execute descriptions and therefore needs to interoperate with standards defined at this lower level. The Web Service Description Language (WSDL) is used to syntactically define the interface of a component using standard web technologies to define means to invoke operations but it does not define notification mechanisms or a standard way of interacting with stateful resources. The Web Service Resource Framework (WSRF) is a standard that extends WSDL in this direction. Initiatives that define syntactic descriptions of resource are orthogonal to the semantically empowered common service layer. The CSL will make use of the former to facilitate the execution of service requests.

The core of our approach is the semantic enrichment of SOAs that implement the Common Service Layer capabilities. This enrichment helps to automate (1) service discovery, service adaptation, negotiation, service composition, service invocation, and service monitoring; as well as (2) data, protocol, and process mediation. This automation is a prerequisite for SOA scalability. To achieve this, we are developing the W<Triple> technology that combines four major building blocks.

- The *Web Service Modeling Ontology (WSMO)*: a conceptual model for structuring semantic annotation of services [Roman et al., 2005],

- The *Web Service Modeling Language (WSML)*: a family of languages providing formal semantics for WSMO models [de Bruijn, 2005],
- The *Web Service Execution Environment (WSMX)*: an execution environment for the dynamic discovery, selection, mediation, invocation and inter-operation of the semantically described Services [Cimpian et al., 2005], and
- Triple Space: [Fensel, 2004] and [Bussler, 2005]: a protocol for the communication of services based on persistent publication of information following the web paradigm.

This work is being conducted in DERI International, which includes institutes in Galway, Innsbruck, Seoul, and Stanford, and in cooperation with large project consortia (e.g. ASG[11], dip[12], Knowledge Web[13], and SEKT[14]) that include many other university groups, small and medium sized companies, government organizations, and large companies such as British Telecom, HP, IBM, and SAP AG.

The remainder of this section describes the building blocks in more detail.

[11] http://asg-platform.org/
[12] http://dip.semanticweb.org/
[13] http://knowledgeweb.semanticweb.org/
[14] http://www.sekt-project.com/

4.1 WSMO

With the Web Service Modeling Ontology (WSMO)[15] [Roman et al., 2005] we developed a conceptual model for structuring the semantic annotation of services. WSMO provides ontological specifications for the core elements of Semantic Web Services. Semantic Web Services aim at an integrated technology for the next generation of the Web by combining Semantic Web technologies and Web services, thereby turning the Internet from an information repository for human consumption into a worldwide system for distributed Web-based computing. Hence, appropriate frameworks for Semantic Web Services, our SESA, need to integrate the basic Web design principles, those defined for the Semantic Web, as well as design principles for distributed, service-orientated computing of the Web. WSMO is therefore based on the following design principles:

- *Web Compliance* - WSMO inherits the concept of URI (Universal Resource Identifier) for unique identification of resources as the essential design principle of the Word Wide Web. Moreover, WSMO adopts the concept of Namespaces for denoting consistent information spaces, supports XML and other W3C Web technology recommendations, as well as the decentralization of resources.
- *Ontology-Based* - Ontologies are used as the data model throughout WSMO, meaning that all resource descriptions as well as all data interchanged during service usage are based on ontologies. Ontologies are a widely accepted state-of-the-art knowledge representation, and have thus been identified as the central enabling technology for the Semantic Web. The extensive usage of ontologies allows semantically enabled information processing as well as support for interoperability; WSMO also supports the ontology languages defined for the Semantic Web.
- *Strict Decoupling* - Decoupling denotes that WSMO resources are defined in isolation, meaning that each resource is specified independently without regard to possible usage or interactions with other resources. This complies with the open and distributed nature of the Web.
- *Centrality of Mediation* - As a complementary design principle to strict decoupling, mediation addresses the handling of heterogeneities that naturally arise in open environments. Heterogeneity can occur in terms of data, underlying ontology,

[15] http://www.wsmo.org/

protocol or process. WSMO recognizes the importance of mediation for the successful deployment of Web services by making mediation a first class component of the framework.

- *Ontological Role Separation* - Users, or more generally clients, exist in specific contexts that will not be the same as for available Web services. For example, a user may wish to book a holiday according to preferences for weather, culture, and childcare, whereas Web services will typically cover airline travel and hotel availability. The underlying epistemology of WSMO differentiates between the desires of users or clients and available services.
- *Description versus Implementation* - WSMO differentiates between the descriptions of Semantic Web Services elements (description) and executable technologies (implementation). While the former requires a concise and sound description framework based on appropriate formalisms in order to provide a concise for semantic descriptions, the latter is concerned with the support of existing and emerging execution technologies for the Semantic Web and Web services. WSMO aims at providing an appropriate ontological description model, and to be complaint with existing and emerging technologies.
- *Execution Semantics* - In order to verify the WSMO specification, the formal execution semantics of reference implementations like WSMX as well as other WSMO-enabled systems provide the technical realization of WSMO. This principle serves as a mean to precisely define the functionality and behaviour of the systems that are WSMO compliant.
- *Service versus Web service* - A Web service is a computational entity that is able to achieve a user goal by invocation. A service, in contrast, is the actual value provided by this invocation [Baida et al., 2005], [Preist, 2004][16]. WSMO provides means to describe Web services that provide access (searching, buying, etc.) to services. WSMO is designed as a means to describe the former and not to replace the functionality of the latter.

The elements of the WSMO ontology are defined in a meta-meta-model language based on the Meta Object Facility (MOF) [MOF, 2002]. MOF defines an abstract language and framework for specifying, constructing, and managing technology neutral meta-models. Since WSMO is meant to be a meta-model for Semantic Web Services,

[16] Note that [Preist, 2004] also distinguishes between a computational entity in general and Web service, where the former does not necessarily have a Web accessible interface. WSMO does not make this distinction.

MOF was identified as the most suitable language/framework for defining the WSMO elements. In terms of the four MOF layers (meta-meta-model, meta-model, model layer, and information layer), the language defining WSMO corresponds to the meta-meta model layer, WSMO itself constitutes the meta-model layer, the actual ontologies, Web services, goals, and mediators specifications constitute the model layer, and the actual data described by the ontologies and exchanged between Web services constitute the information layer (the information layer in this context is actually related to the to the notion of grounding of the semantic descriptions).

WSMO provides three main categories to structure semantic descriptions. First, it provides means to describe *Web services*; second, it provides means to describe user *goals* referring to the problem-solving aspect of our architecture; and third, it provides means to ensure interoperability between the various semantic descriptions of heterogeneous environments: *ontologies* and *mediators*. For complete item descriptions, every WSMO element is described by non-functional properties. These are based on the Dublin Core (DC) Metadata Set [Weibel et al., 1998] for generic information item descriptions, and other service-specific properties related to the quality of service.

Goals provide means to characterize user requests in terms of functional and non-functional requirements. For the former, a standard notion of pre and post conditions has been chosen and the later provides a predefined Ontology of generic properties.

Web service descriptions enrich this by an interface definition that defines access patterns of a service (its choreography) as well as means to express services as being composed from other services (its orchestration). More concretely, a Web service presents:

- a *capability* that is a functional description of a Web service, describing constraints on the input and output of a service through the notions of preconditions, assumptions, post conditions, and effects;
- *interfaces* that specify how the service behaves in order to achieve its functionality. A service interface consists of a *choreography* that describes the interface for the client-service interaction required for service consumption, and an *orchestration* that describes how the functionality of a Web service is achieved by aggregating other Web services.

Ontologies provide a first and important means to achieve interoperability between goals and services as well as between various services themselves. By reusing standard terminologies different elements can be either link directly or indirectly via predefined mapping and alignments.

Mediators provide additional procedural elements to specify further mappings that cannot directly be captured through the usage of ontologies. Using ontologies provides real-world semantics to our description elements as well as machine processable formal semantics through the formal language used to specify them. The concept of mediation in WSMO addresses the handling of heterogeneities occurring between elements that shall interoperate by resolving mismatches between different used terminologies (data level), on communicative behavior between services (protocol level), and on the business process level. A WSMO Mediator connects the WSMO elements in a loosely coupled manner, and provides mediation facilities for resolving mismatches that might arise in the process of connecting different elements defined by WSMO.

4.2 WSML

The Web Service Modeling Language (WSML)[17] [de Bruijn, 2005] is a language for the description of ontologies, goals, Web services and mediators, and which is based on the conceptual model of WSMO. WSML provides a coherent framework that brings together Web technologies with different well-known logical language paradigms. We take Description Logics [Baader et al., 2003], Logic Programming [Lloyd, 1987], and F-Logic [Kifer et al., 1995], as starting points for the development of a number of WSML language variants, based on existing Web standards such as XML Schema and RDF. Its four major dialects form a lattice based on rule languages and descriptions logics as well as on their minimal and maximal intersection (see Figure 8). A major goal in the development of WSML is to investigate the applicability of different formalisms, most notably Description Logics and Logic Programming, in the area of Web services.

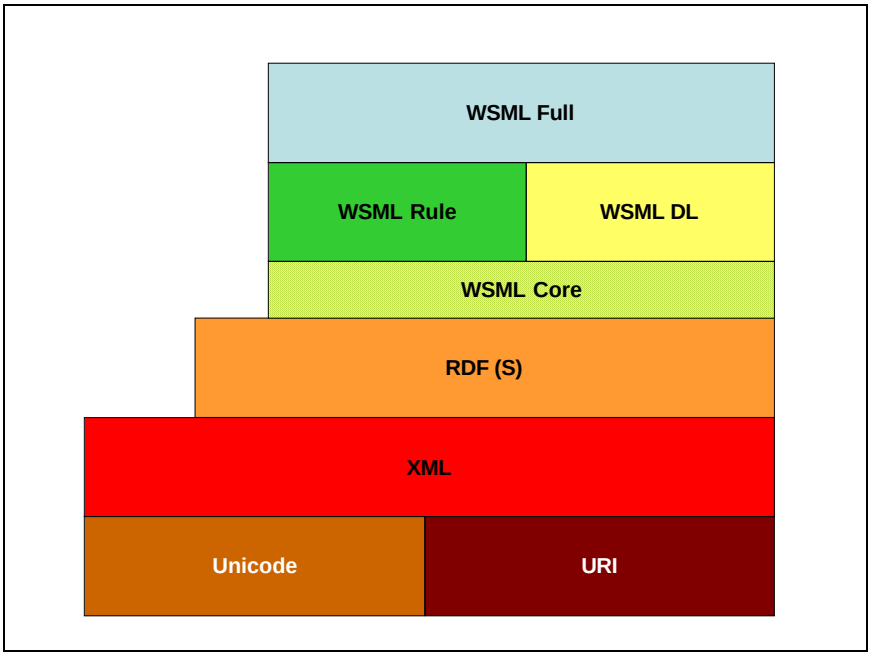


Figure 8. WSML family of languages.

First of all, the languages are layered on top of URIs, name spaces, XML, and RDF

[17] http://www.wsmo.org/wsml/

(S) and therefore reuse existing web-based interchange formats. Second, the reuse of OWL is slightly more complex than in approaches such as OWL-S. WSML-DL, which is similar to OWL, is a syntactic extension of OWL-DL for the description of goals and services. Description Logics (DL) [Baader et al., 2003] is a family of languages that (for the most part) can be seen as subsets of FOPL. We use the Description Logic language SHIQ in WSML since it is an expressive DL and there exists efficient sound and complete reasoning algorithms and implementations for checking concept satisfiability, subsumption, and other reasoning tasks. Furthermore, this has already been applied to Web service discovery and to the Semantic Web in the language OWL [Dean and Schreiber, 2004]. Third, we have a second language called WSML-Rule that is a syntactic extension of the upcoming Web Rule Language[18] for the description of goals and services. Since WRL has not yet been specified, we made informed guesses as to how such a language would look.

Another formal pillar of WSML is Logic Programming. Logic Programming[19] has an extensive body of research results on query answering, as well as many efficient implementations. Furthermore, there exist applications of Logic Programming in the area of Web service for discovery, contracting, and other tasks and there is also a broad interest in applying rules languages to the Web (http://www.ruleml.org/). F-Logic [Kifer et al., 1995] is an extension of FOPL with higher-order style Object Oriented modeling primitives, which stays semantically in a First-Order framework. With F-Logic Programming we mean the Logic Programming language that is obtained from the Horn subset of F-Logic. Using a syntax inspired by F-Logic arguably makes logical expressions easier to write since the modeling vocabulary is not restricted to predicates, as in FOPL, but also includes concepts, instances, attribute typing and attribute values. There are several implementations of F-Logic Programming as well as experience documented in case studies, and commercial products, most notably Ontobroker and FLORA-2. F-Logic Programming can be reduced to regular Logic Programming, thereby benefiting from the research and experience in the area.

[18] See http://www.w3.org/Submission/WRL/ for our proposal.

[19] When talking about Logic Programming we mean purely declarative rules languages, based on the so-called Horn subset of FOPL, with a model-theoretic semantics based on Herbrand models.

Finally, two languages have been developed to provide interoperability between the DL and rule languages. At the lower level, we have WSML-Core that is the intersection of WMSL-DL and WSML-Rule. The intersection is defined as being the maximal sub language that does neither introduce rule elements into a Description Logic nor Descriptions Logic elements into a rule language that would destroy the retrospective interesting computational properties of these two complementary language paradigms. At the higher level, we have WSML-Full, which is defined through the union of WSML-DL and WSML-Rule. On the one hand, it provides full interoperability between DL and rule based descriptions. On the other hand, it requires full First Order Logic extended with non-monotonic aspects as is therefore quite difficult to handle from a computational perspective.

This lattice of languages, which are embedded in current web standards, offers the best that can be achieved with current means. Experience with actual use cases is required to determine the actual value of alternative specification paradigms.

These semantic descriptions have to be aligned (or embedded) in existing syntactic means for describing services such as WSDL for service endpoint descriptions, WSRF as means to describe collection of services and stateful resources, and means as OGSA for describing full-fledged Service-Oriented Architectures. A straightforward and promising proposal has been made by WSDL-S[20]. This simple extension of WSDL provides a grounding mechanism for semantic specifications of otherwise syntactic entities. Currently, efforts are underway to extend WSDL-S to a WSMO compliant grounding mechanism and similar efforts are requested for WSRF and OGSA.

[20] http://www.w3.org/Submission/WSDL-S/

4.3 WSMX

The Web Service Execution Environment WSMX[21] ([Cimpian et al., 2005]) is an execution environment for the dynamic discovery, selection, mediation, invocation and inter-operation of the Semantic Web Services providing a reference implementation for an SOA that uses semantic annotation in all of its major elements. Therefore a general architecture as well as necessary components has been defined and the interfaces and communication of components have been standardized. WSMX is a reference implementation for WSMO and SESA. The development process for WSMX includes defining its conceptual model (which is WSMO), standardizing the execution semantics for the environment, describing the architecture and a software design and building a working implementation.

WSMX will remain a reference implementation of the Semantic Web Services systems. Its primary purpose is to trigger standardization activities in OASIS and to aid European SEMANTIC WEB SERVICE efforts to provide justification for and description of the Semantic Web Services infrastructure in general terms (on the conceptual level) as opposed to the definition of a specific implementation. DERI has contributed WSMX conceptual work and its reference implementation to the open source community as a platform for research and development. WSMX specification has been relabelled and is being developed through OASIS as Semantic Execution Environment (SEE)[22]. The roadmap to the realization of the SEE long-term goal is the OASIS SEE standards activity initiated in November 2005.

[21] http://www.wsmx.org
[22] http://www.oasis-open.org/committees/semantic-ex/

Figure 9 presents the WSMX architecture and its most important components.

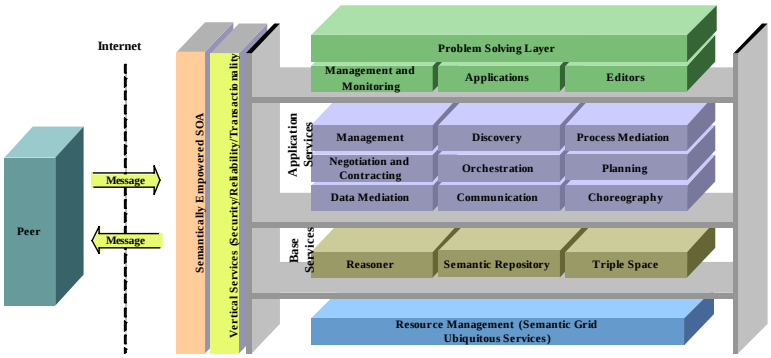


Figure 9. WSMX: A Reference Architecture for SEE.

WSMX is a useful framework for both Web service providers and requesters. As a provider, one may register its service using WSMX in order to make it available to the consumers and, as a requester, one can find the Web services that suits their needs and then invoke them in a transparent, secure and reliable way. WSMX itself is made available as a Web service, so either for finding a Web service or for actually invoking Web services a requester has just to invoke WSMX itself. In the first case, a formal description of the requester goal has to be provided, and in the second case, the actual data the requester wants to use for the invocation. In this way, WSMX can take care of all the other required computations such as heterogeneity reconciliation, composition, security or compensation.

Creating ontologies and semantic descriptions for Web services is useful only if these descriptions can be applied. Infrastructure is vital for a technology to be applied. Web servers and web browsers are the infrastructure that has lead to the success of HTML on the web. WSMX is an execution environment for finding and using Semantic Web Services that are described using WSMO. Considering current Web service technologies there is a large amount of human effort required in the process of finding and using Web services. First the user must browse a repository of Web services to find a service that meets their requirements. Then, once the Web service has been found the user needs to understand the interface of the service, the inputs it requires and outputs it provides. Finally the user would write some code that can interact with the Web service in order to use it. The aim of WSMX is to automate as much of this process as is

possible. The user provides WSMX with a WSMO Goal that formally describes what they would like to achieve. WSMX then uses the Discovery component to find Web services, which have semantic descriptions registered with WSMX that can fulfil this Goal. During the discovery process the users Goal and the Web services description may use different ontologies. If this occurs Data Mediation is needed to resolve heterogeneity issues. Data Mediation in WSMX is a semi-automatic process that requires a domain expert to create mappings between two ontologies that have an overlap in the domain that they describe. Once these mappings have been registered with WSMX the runtime data Mediation component can perform automatic mediation between the two ontologies. Once this mediation has occurred and a given service has been chosen that can fulfil the users Goal, WSMX can begin the process of invoking the service. Every Semantic Web Service has a specific choreography that describes how the user should interact with it. This choreography describes semantically the control and data flow of messages the Web service can exchange. In cases where the choreography of the user and the choreography of the Web service do not match process mediation is required. The Process Mediation component in WSMX is responsible for resolving mismatches between the Choreographies (often referred to as public processes) of the user and Web service.

WSMX is not only a reference implementation based on conceptual model of WSMO, it will become a reference implementation of SESA. This goal has driven the design and architectural decisions described below.

Dynamics. WSMX embarked on the principle of dynamics by interpreting WSML for Semantic Web Service definitions as opposed to a compilation approach. This allows the changes to be made dynamically to a Semantic Web Service without need for recompilation.

Interface vs. implementation. WSMX ensured that Semantic Web Services could be implemented in any language or on any platform by clearly supporting the difference between interface and implementation. While Semantic Web Service interfaces are described in WSML, they can be implemented using any language or technology. This is achieved through an adapter framework that supports linking to existing service implementations.

Grounding. WSMX recognizes existing base technology such as operating systems, databases, and remote communication mechanisms. Hence, we use existing

technology where possible so as to focus the implementation on the semantics aspects.

Dynamic architecture. All functionality is implemented in dedicated components. For example, there are separate components for mediation and choreography. Furthermore, all components are dynamically configured. New components can be added dynamically and existing components can be exchanged or removed. This makes WSMX very flexible and extensible with new functionality that will be provided by WSDL-S, Grid, and other developments.

The SEE/WSMX architecture is being developed and implemented based on the above design decisions. This architecture is the foundation for the broader SESA paradigm that includes resource management, problem solving functionality, as well as grounding in technologies like OGSA and WSRF. The direct mapping of the current WSMX Architecture to the SESA vision is highlighted in Figure 10.

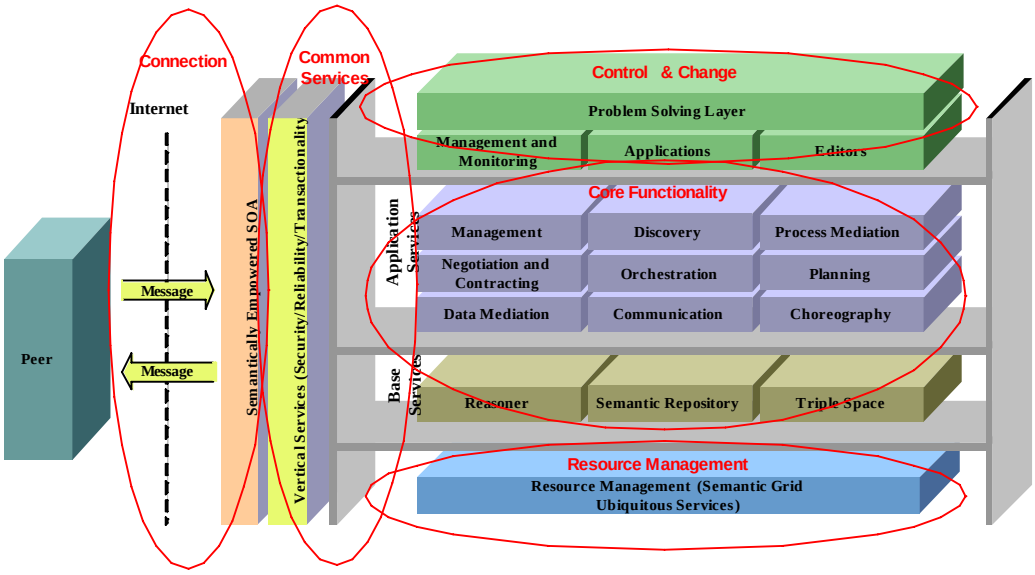


Figure 10. Mapping WSMX to SESA.

Change symbolizes a Service Life-Cycle Environment that supports the service life cycle in which services are developed, modified, and maintained. Control is required to manage services and is enabled by the Service Control Platform that provides tools to monitor, govern, and manage service execution. Connection denotes connection between

services, the technical essence of SESA, and is realized through the Service Delivery Network through which all service interactions take place. WSMX is intended to be a full SOA implementation and its functions are expected to augment SOA with semantics. Some existing WSMX components map loosely onto the core SOA functions, including discovery (search), selection (match), compensation (substitution), choreography (composition), and adaptations of various kinds that underly and support SESA functions.

WSMX supports common B2B and B2C scenarios, acting as an information system representing the central point of a hub-and-spoke architecture. If two partners want to communicate, they define their required services to WSMX, not one to each other. This is an example of a key aspect of SESA, that there is a clear distinction between an interface and its implementation. This allows service registration, discovery, composition, and execution without any knowledge of implementations or their locations. The service-implementation distinction also supports the semantic definition of a service independently of its underlying implementation.

Grounding is another key aspect of the SESA paradigm that is reflected in WSMX. Grounding is the realization of semantic technology based on non-semantic technology. A SESA architecture implementation need not implement the whole underlying technology stack by itself from the hardware on up. Instead, it uses existing technology to the full extent possible. For example, services are invoked remotely over a network. Existing technologies for this functionality are manifold, in the specific context it is the Web Service Definition Language (WSDL) combined with SOAP as the transport mechanism. Technology for WSDL and SOAP exist and so it is advisable to use this technology, even though it is only of syntactic nature. Grounding as a fundamental concept means to map the semantic world to the syntactic world and back. In terms of a service invocation this means that when a Semantic Web Service is invoked, the invoker has the impression to invoke a Semantic Web Service, while the SESA architecture implementation utilizes the non-semantic combination of WSDL and SOAP as the remote service invocation mechanism.

Web services represented a step forward in enabling web-based collaboration between entities by minimizing or eliminating interoperability challenges. B2B and B2C partners can publish and consume each other’s services. Hence information systems based on a Service-Oriented Architecture can interoperate so as to achieve business

partner collaboration, independent of specific implementation details. WSMX is directed at these implementation-independent requirements that arise in B2B and B2C collaborations and serves as reference implementation for both WSMO and SESA.

4.4 Triple Space

The fourth and final building block of the W<Triple> technology is Triple Space, a new paradigm to enable communication and cooperation of services. We are currently entering the fourth phase of the convergence of computing and communication based on Semantic Web Services. The four phases, depicted in Figure 11 below, are defined in terms of the computing and communication realms and how they interact. Communication can involve one individual in communication with another (1: 1) or one individual in communication with many (1: many). In the age where the computer is the network, computing can involve humans interacting with humans, semi-automatically, via the network or can involve machines interacting with machines automatically via the network.

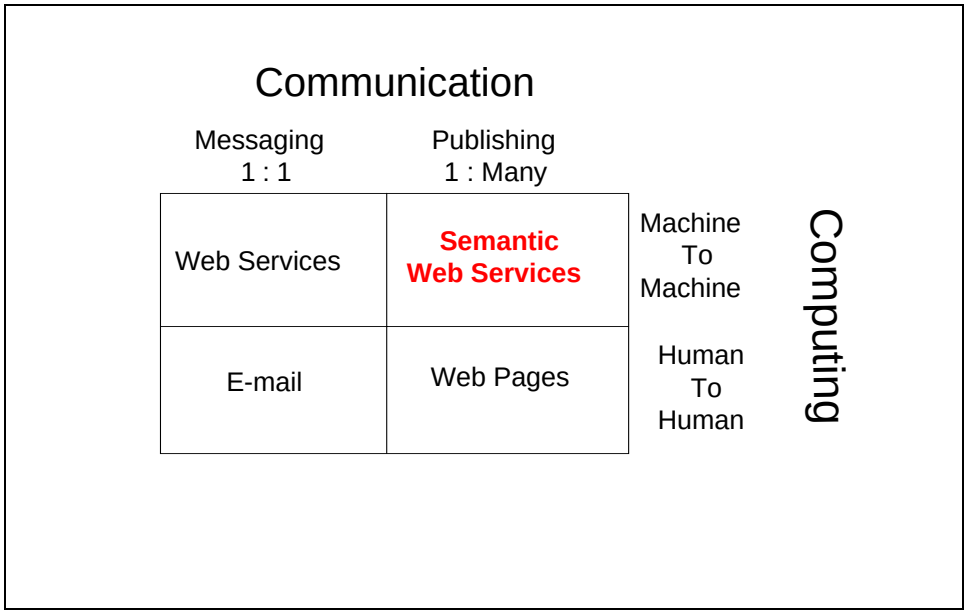


Figure 11. Four Phases of Computing / Communication Convergence.

Phase One, characterized by E-mail, was enabled by the Internet and involved one human communicating with one or a small number of humans. This revolutionized communications and computing and furthered their convergence. Phase One lead to enormous scaling of message-based computing/communications that was achieved by miracles of network and infrastructure engineering. While well suited to messaging or

brief communications, it was not well suited to publishing.

Phase Two, characterized by web pages and enabled by the World Wide Web, permitted a human or a human organization to communicate with the entire networked world. Phase Two led to massive innovation and growth in computing, communication, and their convergence, what we call web (or publishing)-based computing/communications. It led to business innovation and economic growth. Phase Two scaling was also achieved by network and infrastructure engineering.

We are entering Phase Three, characterized by Web services, which is being enabled by Service-Oriented Computing, and which will revolutionize computing and communications based on message-based computing. Message-based computing permits the composition and re-composition of services as required to meet changing requirements in contrast with inflexible architectures of conventional computing. This technical flexibility will contribute to the greater goal of business flexibility – the ability to modify automated business processes with ease. Phase Three message-based computing achieves computing/communication convergence since you can longer compute without communicating and *vice versa*. Phase Three leaves the realm of human-to-human networking and enters the 1:1 quadrant of machine-to-machine networking. In Phase Three, a machine, or a service on a machine, requests services of one or a small number of services (on other machines). The publish-subscribe paradigm of message-based computing falls into the realm of one service, or a small number of services, that subscribes to a single service. Scaling in Phase Three involves enabling 1:1 service interactions - matching a service request to a service offering and adapting (integrating) the service protocols, processes, and data. Scaling will be limited by syntactically based, semi-automatic means for service discovery, selection, orchestration, and adaptation.

In Phase Four, enabled by knowledge technologies, service interactions will be created dynamically by knowledge-based service discovery, selection, orchestration, and adaptation. Interactions will move from the 1:1 realm of message-based computing to the 1:N realm of full publish-subscribe. Scaling in Phase Four over Phase Three will be comparable in order of magnitude to that of the shift from e-mail based to web-based communications. We envisage a growth in technical and business innovation moving from Web service-based computing/communications to Semantic Web Services comparable to that experienced in the movement form the Internet to the World Wide Web. We call the underlying technology W<Triple>. This movement was predicted for

the Web moving to the Semantic Web. The prediction here is for the movement of all computing, e.g., information systems, from Web services to services that can interact dynamically to adjust to changing requirements. Besides their name, current *Web service*s do not have much to do with the web. Let's illustrate this briefly by assuming a time machine would bring us back to the pre-web time. What was the common method of accessing a research paper? One was posting an email kindly asking for the paper and a friendly colleague posting it as an attachment. Dissemination of information was based on message exchange. The communication overhead in publishing and accessing information was high and dissemination was therefore quite limited and slow. Then the web came into being and changed the situation significantly. The author had to publish the paper once by putting it on his web page. After this, he could forget about it and focus on writing new papers. New services such as citeseer[23] even ensure durability of this publication beyond the lifetime of a web page (i.e., they disable the delete operation on the information space). All the potential readers could get instant access to the paper without requiring a two-stage message-exchange process. This tremendously scaled and speeded up the dissemination process of information. When comparing Web services with this essential web principle it becomes quite obvious that Web services are **not** about the web.

Web services require close coupling with the applications that they integrate. Applications communicate via message exchange requiring strong coupling in terms of reference and time. The communication has to be directed to the Web service addressed and the communication must be synchronous. If both parties do not implement and jointly agree on the specific way this mechanism is implemented, then the applications must support asynchronous communication. The web is strongly based on the opposite principles. Information is published in a persistent and widely accessible manner.[24] Any application can access this information at any point in time without having to request the publishing process to directly refer to it as a receiver of it's information. While Web services use the Internet as a transport media (relying on protocols such as FTP, SMTP, or HTTP), that is all they have in common with the web.

Tuple-based computing was introduced in parallel programming languages, such as Linda, to implement communication between parallel processes [Gerlernter, 1992].

[23]http://citeseer.ist.psu.edu/cs

[24]For privacy issues, protected sub-fragments of the web can be defined.

Instead of exchanging messages point to point, a more efficient means of communication was provided. Processes can write, delete[25], and read tuples from a global persistent space.[26] A tuple is a set of ordered typed fields, each of which either contains a value or is undefined. A tuplespace is an abstract space containing all tuples that are visible to all processes. The API for this is extremely simple thus eliminating the complexity of distributed message processing. In reality, the complexity is hidden in middleware that implements the tuplespace. The tuplespace is similar to a blackboard in expert systems, where rules do not send messages to all other rules when they derive a fact. Rather, this is published by adding it to the publicly visible board.

Tuple or space-based computing has one very strong advantage: It de-couples three orthogonal dimensions involved in information exchange: reference, time, and space.

- Processes communicating with each other do not need to know each other explicitly. They exchange information by writing and reading tuples from the tuplespace and do not need to set up an explicit connection channel, i.e., *the processes are completely de-coupled*.
- Communication can be completely asynchronous since the tuplespace guarantees persistent storage of data, i.e., *time-wise the processes are completely de-coupled.*
- The processes can run in completely different computational environments as long as both can access the same tuplespace, i.e., *space-wise the processes are completely de-coupled*.

This strong decoupling in all three relevant dimensions has obvious design advantages for defining reusable, distributed, heterogeneous, and rapidly changing applications like those promised by Web service technology. Also, complex APIs of current Web service technology are replaced by simple read and write operations in a tuplespace. Notice that a service paradigm based on the tuple paradigm also revisits the web paradigm: information is persistently written to a global place where other processes can smoothly access it without starting a cascade of message exchanges.

[25]Actually, deleting tuples may not really be necessary in an exponentially growing space such as the web.
[26]*Global* in the *local* framework of an application that is decomposed by parallel processes.

In side remarks [Johanson and Fox, 2004] report shortcomings of current tuplespace models. They lack the means to name spaces, semantics, and structure in describing the information content of the tuples. The tuplespace provides a flat and simple data model that does not provide nesting, therefore, tuples with the same number of fields and field order, but different semantics, cannot be distinguished. Instead of following their ad-hoc repairs we propose a simple and promising solution that refines the tuplespace into a *triple space, where <subject, predicate, object> describe content and semantics of information*. The object can become a subject in a new triple thus defining a *graph structure* capturing structural information. Fortunately with RDF[27] [Klyne and Carroll, 2004] this space already exists and provides a natural link from the space-based computing paradigm into the semantic web.

The web and the tuplespace have many things in common. They are both global information spaces for persistent publication. Therefore, they share many underlying principles. They differ in their application context. The web is a worldwide information space for the human reader and the tuplespace is a local space for parallel processes in an application. Thus, the web adds some features that are currently lacking in the tuplespace.

- First, with URIs the web provides a well-defined *reference mechanism* that has worldwide scalability to address chunks of information. Tuplespaces lack this mechanism since they were designed mostly for closed and local environments. Johanson and Fox [Johanson and Fox, 2004] already reported this as a bottleneck when applied in their setting of heterogeneity and dynamic change.
- Second, the namespace mechanism of the web allows different applications to use the same vocabulary without blurring their communications. Namespaces help to keep the intended information coverage of identifiers separate even if they are named equally. Namespaces provides a well-defined *separation mechanism* that scales on a worldwide scale to distinguish chunks of information.
- Third, the web is an information space for humans and the tuplespace is an information space for computers, however, the *semantic* web is also for machines. It provides standards to represent machine-processable semantics of data. We already mentioned RDF that provides nested triples as a data model to represent data and their formal semantics on the web. This enables applications to publish and to access information in a machine processable manner. RDF Schema

[27] http://www.w3.org/RDF/

> [Brickley and Guha, 2004] defines classes, properties, domain and range restrictions, and hierarchies of classes and properties on top of RDF. Thus, a richer data model than nested triples can be used to model and retrieve information.

The semantic web has the true potential to become the global space for application integration just as the tuplespace became a means for the local integration of parallel processes. It provides the means for global integration with the inherent complexity stemming from information heterogeneity and dynamic changes. As with tuplespace, it makes problems with protocol and process heterogeneity transparent, by it's uniform and simple means for accessing and retrieving information. Complex message exchanges is replaced by simple read and write operations in a global space.

Having said this, it is also clear that this is not the end but just the beginning of an exercise. No application can quickly check the entire semantic web to find an interesting triple. Conversely, no application would simply publish a triple and then wait forever until another application picks it up. Clever middleware is required that provides a virtual global triplespace without requesting each application either to download or to search through the entire semantic web. The triple space needs to be divided up to provide security and privacy features as well as scalability. However, none of these requirements are really new. They apply to any application that deals with the web on a global scale.

5 Resource Layer

Resources are used to solve problems or more conventionally to execute applications. The resource layer [Fahringer et al., 2005] deals with resource discovery, selection and negotiation for advanced or on-the-fly reservation of resources. The resource layer also covers the deployment and provisioning of physical and logical resources. Resources in the context of an SOA can be subdivided into multiple classes covering, among others, both physical and logical resources. *Physical resources* (e.g. computers, data servers, and networks), which are commonly connected to form a grid of computing and storage platforms; at this level automatic resource management will be facilitated from the perspective of both resource provisioning as well as its lifecycle management. *Logical resources*, such as application components or common services, enabling more advanced composition of applications.

Figure 12 demonstrates how resource management can be integrated with an SOA. At the highest level an application can be orchestrated or composed as a workflow application. Scheduler, reasoner, or mediator services would be provided with a description for the workflow based on which an SLA negotiation could be established with the underlying resource manager. The resource manager has to provide an effective environment to deploy, provide and register resources such as application components (e.g. Web services). After a scheduling service agrees with the resource manager to the usage of specific services, the execution of the workflow can begin. Depending on the dynamic behaviour of the resources during execution of the workflow, re-scheduling and re-negotiation can be launched to adjust to this dynamic behaviour and to incorporate different business models.

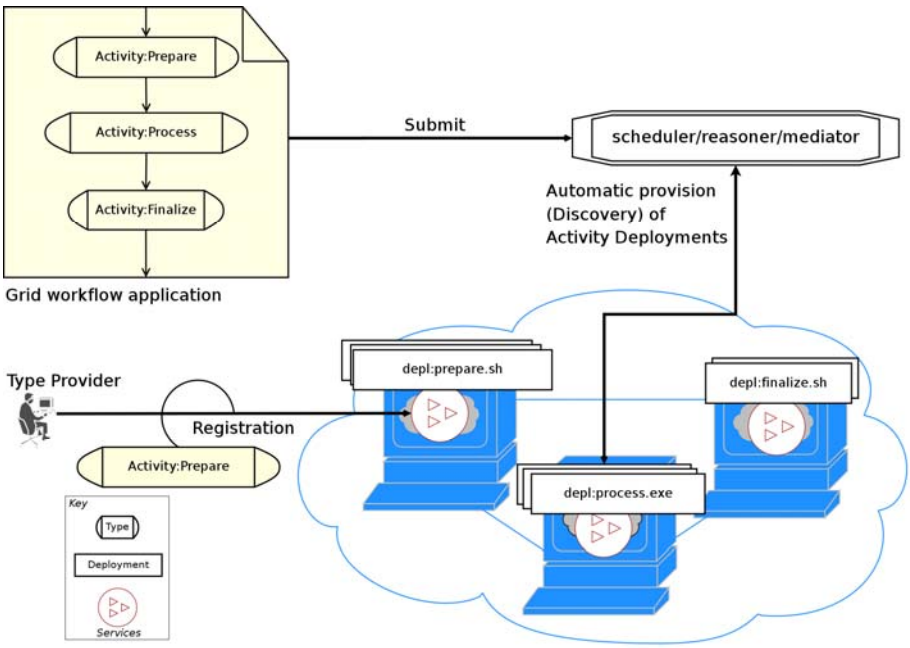


Figure 12. Resource Management in the Context of an SOA.

5.1 Ubiquitous and Grid Computing as part of the Resource Layer

Two of the most prominent and widely discussed areas that deal with distributed resources in the context of Service-Oriented Computing are Ubiquitous Computing and Grid Computing. They can be seen as two endpoints in a continuum where their characteristics are somewhat complementary. Grids rely on a relatively large number of hardware devices ranging from small computers to very powerful devices interconnected with mostly conventional networks (Internet). Ubiquitous Computing environments, on the other hand, are suffering from weak and unreliable connections (due to partial autonomy) between a very dynamic constellation of a high number of mobile devices with limited memory and processing power. Grid nodes are typically general purpose so that a wide variety of possibly complex services can be run on one or more Grid nodes. Ubiquitous Computing nodes are usually of a much more specialized purpose. They may consist only of a single sensor or actuator and some networking facility. Thus the set of services that can be provided on this kind of node is limited, even if each of them may be of high value for selected applications. Other aspects of complementarities include energy constraints and the ability to provide and access central services.

Based on this analysis, we see Ubiquitous Computing and Grid Computing as two endpoints of the Resource Management Layer. This layer is specifically involved in issues caused by data distribution in a Service-Oriented Architecture, which include service description, discovery and composition, availability, autonomy and mobility. Its main task is to provide negotiation and adaptation facilities for functional and non-functional properties of services and components of the framework involved in a service interaction.

Some of the tasks of the Resource Management Layer are more challenging on one end of the continuum than on the other. For instance, only partial reasoning is possible on the Ubiquitous Computing end for several reasons. This is because the limited storage and processing power capacities available on typical Ubiquitous Computing devices do not allow maintenance of a full-fledged knowledge base. Also, required network connectivity to complete missing parts on demand or perform distributed reasoning is not always available. While context-awareness is almost inherently given in Ubiquitous Computing environments, it is more challenging to provide context-awareness to services running on Grid nodes ([Krummenacher et al.,

2005]). This is due to processes such as service personalisation and adaptation to the location where an accurate real-time projection of the sensed environment to a reference model available on the Grid node is required. Other typical service framework tasks such as service compensation[28] is challenging on the Ubiquitous Computing end because of the dynamics in terms of mobility, network reliability and contracting.

On the Grid end, mediation, optimization, reasoning and negotiation is more challenging in terms of the number options that need be considered. Moreover, the widespread use of Grid technology required interoperability and IT support for tooling also becomes problematic ([Hey, 2005]). With the emergence of Web service technology that is currently being upgraded to WSRF, a standard mechanism for describing and accessing Grid resources has been introduced that allows powerful services to be made available over the Internet. This contribution to the evolution of the Grid attracted much attention well beyond the scientific community covering both industry and business applications. Currently much work is devoted to building the components of the Open Grid Service Architecture (OGSA) that defines a Service-Oriented Architecture that resides on top of a low-level Grid operating system and directly targets distributed applications. OGSA comprises services such as scheduler, enactment engine, resource management, information service, security service, monitoring, service discovery, service negotiation, and service composition. On top of an OGSA-based Grid runtime environment, we commonly find a layer for modelling, describing and developing Grid applications ([Fahringer et al., 2005]). The output of that layer represents a specification of a Grid application that can be passed to a Grid runtime environment for execution. Usually the development and execution of a Grid application is sequential which means that an application is first fully built and then executed. In rare cases, however, applications are built and executed simultaneously. The dynamic behaviour of a running application and the Grid may impact the remaining part of an application that has yet to be built and executed.

Many existing Grid projects have developed and deployed a plethora of sophisticated services that build and run business, industry or scientific Grid applications. However, most of these services are described only syntactically which severely hampers the discovery and reuse of existing services on the Grid. Many services are built over and over again because people or machines are unable to automatically locate them. In most

[28] also called *handover* or *hand-off*

cases human beings use conventional Web browsers to search for existing services. Thus adding machine processable semantics to all Grid resources (e.g. Web services, computers, storage systems, networks, etc.) and providing automatic discovery mechanisms is a crucial aspect to enable widespread usage of the Grid, to reduce the cost of the development, and to simplify the negotiation for using existing resources over the Grid.

5.2 Connections are Services

Connectivity is not inherently given in SOAs, it's a service provided through the SOA implementation. Any other service may make use of communication services if and only if they are available. Connections between services -- the technical essence of SOA and base principle of collaboration -- is realized in the Service Delivery Network through which all service interactions take place. Connections are the more "traditional" services provided through an SOA implementation, having a narrow set of functional properties (mainly to provide communication facilities between involved parties) and attributes, but a reasonably wide set of non-functional properties (cost, availability, etc.). Other services adopting the functionality of communication services have to not only deal with the functional properties, but also with the non-functional properties of communication services, meaning that in real-time environments they have to be aware of the fact that there is a wide range of connectivity at any given time, from several high-speed communication channels down to partial or full autonomy. Obviously, connections are first-class resources to be managed on the resource management layer in any SOA.

We can learn from the broad range of functional components that have been designed for communication services. One example are the handover (also called hand-off) procedures that ensure seamless service provisioning (maintaining a connection) while moving around. Communication service handover procedures are very similar to SOA service compensation procedures as shown in [Strang et al., 2003]. It has been shown that substitutability is the relevant property of interoperability for any kind of service handover would benefit in particular from adding semantics.

A TupleSpace can quite naturally be seen as a communication service (in terms of a persistent storage for asynchronous message exchanges). In doing so, the step from TupleSpaces to TripleSpaces lifts a communication service up to a collaboration service.

5.3 Semantically Enabled Resource Management

Many existing systems require manual or semi-manual deployment of common services, as well as force application builders to hardcode specific services deployed on specific resources into their applications. Additionally, currently available information services are not well adapted to store complete description of services, forcing the application builder to use only (name, location) similar information about available common services. As a consequence these applications are difficult to port to different resource platforms, are sensitive towards dynamic changes of a resource platform, and often imply an avoidable failure rate during execution. Such a manual and hard-coded approach forces an application developer to deal with low-level details of the resource platform. In aiming at the *Ambient Intelligence* vision that "people will be surrounded by intelligent and intuitive interfaces embedded in everyday objects around us and an environment recognising and responding to the presence of individuals in an invisible way by year 2010" [ISTAG, 2001] supported and enforced by the European Commission, it is obvious that an exponentially increased number of objects hosting SOA components cannot be managed manually and hard-coded.

The goal of semantically enabled resource management is to propose a semantic-based resource management that goes beyond the current state-of-the-art of discovery, selection and optimization. Resource management as a part of an SOA will provide mechanisms to manage virtualized resources annotated with semantic descriptions by evolving current groundings in Grid services (WSRF, WS-I*) and Semantic Web Services (WSMO, WSDL-S) in order to allow their mechanized execution. It will also provide support for the definition and execution of policies specified in the form of constraints. Resources will be accessed as services to follow the idea that everything is a service within an SOA. Thus resources and resource management naturally become an integral part of an SOA.

On the other hand, resources will be instrumented and monitored for non-functional parameters [Truong and Fahringer, 2004] such as usage, reliability, costs, and performance. These parameters are of paramount importance to drive optimization, negotiation, reservation, and scheduling services. We distinguish between static and dynamic non-functional parameters. Static parameters refer to a resource behaviour or attribute that does not change at all or with long time intervals. Examples of such static

non-functional parameters are operating system, computer architecture, communication protocol, data centre name, etc. Dynamic non-functional parameters may frequently change during the lifetime of a resource comprising of degree of usage, costs for usage, performance, number of processes running, number of failures, number of security attacks, etc. Clients (e.g. WSMX services) that request for resources will commonly specify or negotiate for SLAs defined over these (QoS) parameters. Research [Strang, 2003] on SLAs in Web service environments such as WSLA[29] showed that syntactical descriptions of non-functional properties and their metrics, as well as their limited validity between selected named parties, are not sufficient in large scale SOAs. Thus it is crucial to develop ontologies for such parameters in order to guarantee a unique meaning of these parameters ideally based on widely accepted standards. Non-functional parameters must be monitored and stored in well-defined registries that keep resource information up-to-date. Based on these resource registries clients can negotiate, reserve and account for resources with a resource management service (integrated part of the resource layer).

Configuration and lifecycle management of the physical resources including computing elements and logical resources is still a tedious task that requires much manual effort. Incorporating semantic descriptions based on state-of-the-art ontology languages will substantially facilitate [Siddiqui et al., 2005] configuration and lifecycle management as part of a lifecycle environment with the goal to minimize human interaction required as much as possible.

Moreover, distributed applications and their execution times are somehow unpredictable in the scenarios of workflow applications. This emphasizes the need of rescheduling or dynamic scaling [Prodan and Fahringer, 2005] [Wieczorek et al., 2005] of the resources allocated to the running applications. Furthermore, advanced reservation of resources is a crucial aspect to simplify the execution of applications on a set of distributed resources and requires a negotiation mechanism in order to support service-level agreement (SLA) between resource providers and clients.

The Resource Management Layer provides the architectural support to enrich space-based computing [Fensel, 2004] with aspects of pervasiveness such as asynchronous communication [Krummenacher et al., 2005] as well as aspects of

[29] Web Service Level Agreements. *http://www.research.ibm.com/wsla.*

Semantic Grid technology such as virtual organizations. Our layering goes also in line with [De Roure et al., 2005], where it is argued that "both need ease of dynamic assembly of components, both rely on interoperability to achieve their goals", and, in particular, to achieve semantic interoperability, "the semantic approach sits above the large scale distributed systems of Pervasive and Grid computing".[30]

Capacity planning is of paramount importance for resource infrastructures. Proper capacity planning ensures a stable and powerful resource infrastructure that can grow to meet future needs. QoS is an increasingly important aspect for the management of resource infrastructure capacity as more sophisticated and heterogeneous distributed applications and services get deployed.

In summary, problems are implemented by a well-defined and ordered set of services (application). Development of complex services involves the selection of services appropriate for the given context and the orchestration of those services into a composite or complex service to meet the requirements of that context. Orchestration and invocation of services will rely on registries for resources on which these services are deployed. Besides the functional description of services that is necessary to create the desired application, a variety of non-functional parameters will be used to achieve additional important goals such as cost effectiveness, reliability and security. The management and control of resources, common services deployed on them, connection services, scalability -- as well as the monitoring of resource behaviour -- are the main goals of an SOA resource management.

[30] The fact that both areas are starting to grow together is also illustrated by the 1st International Conference on Grid and Pervasive Computing.

6 Progress towards the Vision

The SESA Manifesto may take a decade to realize. However, the objectives are well understood and widely shared. This paper attempts to characterize the objectives in terms of a single vision and framework, as well as provide a technical outline for the major building blocks. Progress has already been made in both research and industry on many aspects of the vision and commitments that have been made in the realization of various aspects of the vision. This section reviews progress towards the realization of the SESA vision in both research and industrial communities.

6.1 Research Efforts

Considerable research has been completed and is under way to realize the potential of semantically enabled SOA. This section reviews four relevant research initiatives, OWL-S, SWSF, IRS-III, and WSDL-S, each of which has gained some momentum and addresses some pragmatic aspects. Each initiative can be characterized in terms of (1) a conceptual model describing the underlying principles and assumptions; and (2) a language or a set of languages that provide the means to realize the model[31]. At the end of this section, we summarize the relations of the four approaches to the framework provided by the W<Triple> approach.

[31] This scheme has been applied to discuss all frameworks with a small derivation when discussing the IRS-III, since as an architecture and concrete platform we need a more implementation oriented reflection to appropriately reflect it.

6.1.1 The OWL-S Approach

OWL-S [OWL-S, 2004] specifies a set of ontologies based on OWL which are used to describe the different aspects of a Semantic Web Service. OWL-S defines its meta-model using the Web Ontology Language, the same language that it uses for a concrete description. The model is described by three sub-ontologies, known as service profile, service model, and grounding, as illustrated in Figure 13.

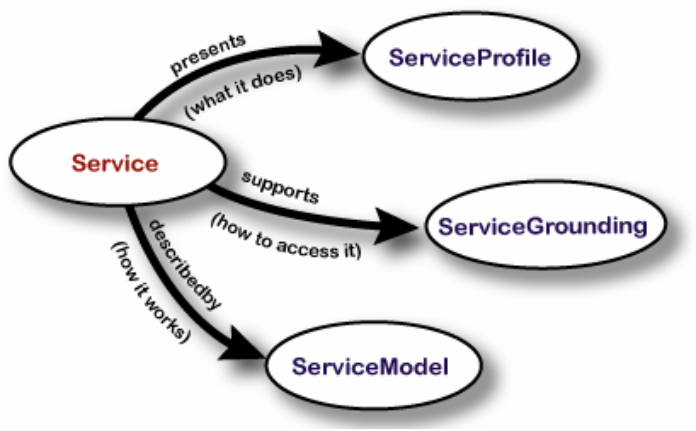


Figure 13. Top level elements of OWL-S [OWL-S, 2004].

Conceptual model

The *service profile* is used to express "what a service does" *(presents)* for purposes of advertising, constructing service requests, and matchmaking. It can be used to reference both non-functional descriptions and existing categorization schemes or ontologies. The most essential information presented in the profile is the specification of what functionality the service provides, the information transformation -- represented by inputs and outputs of the service -- and the state change produced by the execution of the service (which is represented by the preconditions and effects of the service).

The *service model* is used to describe "how a service works" *(describedby)* to enable invocation, enactment, composition, monitoring and recovery. The interaction is viewed as a process. A process is not necessarily a program to be executed, but rather a specification of ways a client may interact with a service. Standard work flow constructs such as sequence, split or join can be used to describe the service model.

The *grounding (supports)* maps the constructs of the process model to detailed specifications of message formats, protocols, and so forth (normally expressed in WSDL).

Language

By design the primary language used for description of services is the web ontology language (OWL), however it soon became clear that it is not sufficiently expressive for all aspects of a service, hence other more expressive languages have been syntactically integrated: SWRL ([SWRL, 2003]), KIF ([KIF, 1998]), DRS, and PDDL ([PDDL, 1998]). By reusing OWL as a recommended standard OWL-S gained considerable momentum. However, it has the disadvantage of the need to retrofit more expressive languages into the framework which then opens new research questions on how they should interact.

6.1.2 The SWSF Approach

Semantic Web Services Framework (SWSF) ([SWSF, 2005]) is one of the newest approaches for Semantic Web Services. It is being proposed and promoted by Semantic Web Services Language Committee (SWSLC)[32] of the Semantic Web Services Initiative (SWSI)[33]. It is based on two major components: an ontology (or conceptual model) and a language used to axiomatize it.

Conceptual model

The *Semantic Web Services Ontology* (SWSO) has been influenced by OWL-S and shares its three concepts: profile, model and grounding as described in the previous section. Thus SWSO can be seen as an extension or refinement of OWL-S. Although there are many similarities with the OWL-S ontologies, one important difference is the expressiveness of the underlying language. Another fundamental aspect is a rich behavioural process model based on the Process Specification Language (PSL) [Gruninger, 2003].

In the SWSF approach there are two independent languages. In the following we briefly review the ontology as it is described in the FLOWS variant – First Order Logic Ontology for Web services. The second axiomatization, ROWS - Rule Ontology for Web services, shares the conceptual model but provides a different set of concrete semantics which relies on Logic Programming semantics.

The Process Model is that part of the FLOWS ontology, which offers constructs to describe the behaviour of the service based on the Process Specification Language (PSL) approach, by adding two fundamental elements: (1) the structured notion of atomic process as found in OWL-S and (2) the infrastructure for specifying various forms of data flow. The core part of the PSL extended by FLOWS is called PSL Outer Core and the resulting FLOWS sub-ontology is called FLOWS-Core. The process ontology is made up of six parts that are divided according to their expressivity such as ordering constraints and occurrence constraints.

Language

The Semantic Web Services Language (SWSL) is the language for describing Web services concepts and descriptions of individual services. SWSL comes as previously

[32] http://www.daml.org/services/swsl/
[33] http://www.swsi.org/

mentioned in two variants: SWSL-FOL and SWSL-Rules. The design of both languages was driven by compliance with Web principles such as usage of URIs, integration with XML built-in types, and XML-compatible namespaces and import mechanisms. Both languages are layered languages where every layer includes a number of new concepts that enhance the modeling power of the language. This means it is richer and more expressive than OWL-S which is based on OWL-DL, a Description Logics formalism. Being based on First Order Logic, FLOWS makes use of logic predicates and terms to model the state of the world. Features from situation calculus, like the use of fluents, predicates, and terms which vary over time, were introduced to model the change of the world.

SWSL-Rules is a logic programming language, which includes features from Courteous logic programs [Grosof, 1999], HiLog [Chen and Kifer, 1993] and F-Logic [Kifer et al., 1995] and can be seen as both a specification and implementation language. SWSL-Rules language provides support for service related tasks such as discovery, contacting, and policy specification. It is a layered language as illustrated in Figure 14. The core of the SWSL-Rules language is represented by a pure Horn sub-set. This subset is extended by adding features such as disjunction in the body and conjunction and implication in the head (Mon LT) [Lloyd, 1987], or negation in the rule body interpreted as negation as failure (called NAF). Other extensions are (1) Courteous rules (Courteous), (2) HiLog, and (3) Frames.

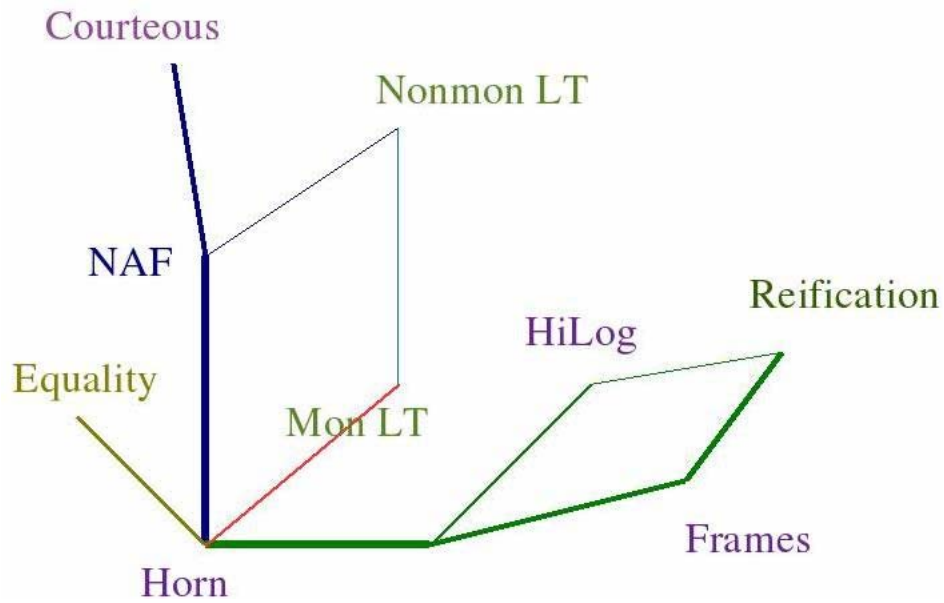


Figure 14. The Layered Structure of SWSL-Rules [SWSF, 2005].

The second language SWSL-FOL is based on First Order Logic and includes features from HiLog and F-Logic. It has a layered structure similar to SWSL Rules with various extensions. Some of the extensions provided for SWSL-Rules apply for SWSL-FOL as well.

6.1.3 The WSDL-S Approach

WSDL-S [Akkiraju et al., 2005] proposes a mechanism to augment Web service functional descriptions with semantics, as represented by WSDL [WSDL, 2005]. This work is a refinement of a proposal developed by the Meteor-S group, at the LSDIS Lab[34], Athens, Georgia.

Conceptual model

Starting with the assumption that a semantic model of the Web service exists, WSDL-S describes a mechanism to link this semantic model with the syntactic functional description captured by WSDL. Using the extensibility elements of WSDL, a set of annotations can be created to semantically describe the inputs, outputs and operations of a Web service. This method keeps the semantic model outside WSDL, making the approach agnostic to any ontology representation language as illustrated in Figure 15.

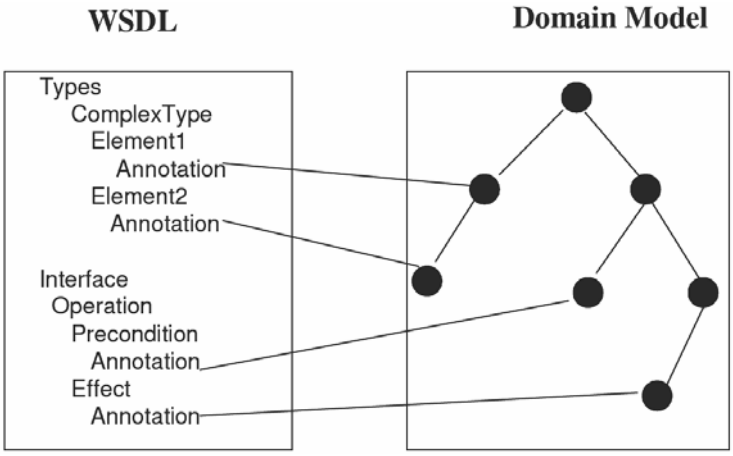


Figure 15. Associating semantics to WSDL elements [Akkiraju et al., 2005].

The advantage of this approach is that it builds on an existing standard. The underlying design principles of WSDL-S can be summarized as follows: (1) WSDL-S aims to *build on existing Web services' standards* and promotes an upwardly compatible mechanism for adding semantics to Web services; (2) annotations should be *agnostic to*

[34] See http://lsdis.cs.uga.edu/.

the semantics representation language -- consequently, WSDL-S does not prescribe what semantic representation language should be used; and (3) support *annotation of XML Schema data type* needs to be added to XML Schema, as it is an important data definition format. Annotations are used for adding semantics to input and output descriptions. In addition, the creation of mappings between the XML Schema complex types and the corresponding ontological concepts is important and corresponding attributes are included in WSDL-S.

Language

WSDL-S does not fix a specific formalism for semantic description. However, unlike the other three frameworks, it fixes the underlying language for the grounding to WSDL. WSDL-S proposes concrete extension points to WSDL. Following the principle of reusing the typing information given in the XML, the schemaMapping and modelReference allow the mapping to the ontology language to be used as semantic annotation. A category attribute can be used for classification and precondition and effects can be used for the annotation of operation functionality. Each of these elements can be used to create annotations, whereas WSDL-S does not dictate a specific language but just includes some recommendation of usage of the extensions.

6.1.4 The IRS-III Approach

IRS-III [Domingue et al., 2004] is a framework and an implemented platform that acts as a broker mediating between the goals of a user or client and available deployed Web services. Thus the IRS is not a framework on its own but uses WSMO as its ontology and follows the WSMO design principles.

Conceptual model

IRS-III is based on five design principles: (1) Supporting *Capability Based Invocation* - IRS-III enables clients (human users or application programs) to invoke a Web service simply by specifying a concrete desired capability. The IRS acts as a broker finding, composing and invoking appropriate Web services in order to fulfil the request. (2) *Ease of Use* - IRS interfaces are designed such that much of the complexity surrounding the creation of Semantic Web Service based applications are hidden. One Click Publishing is a corollary of the above design principle. (3) *Agnostic to Service Implementation Platform* - Within the design of the IRS there is no strong assumption about the underlying service implementation platform. (4) *Connected to the External Environment* - To support this, functions and relations can be defined in order to make calls to external systems – for example, invoking a Web service. (5) *Inspectibility* - In many parts of the life cycle of any software system, it is important that the developers are able to understand the design and behaviour of the software being constructed. This principle is concerned with making the semantic descriptions accessible in a human readable form. The key is that the content and form are easily understandable by SEMANTIC WEB SERVICE application builders.

Many of the direct principles of IRIs are application focused, but from the first design principles and the reuse of WSMO we can say that it largely follows the problem solving approach described in this paper. The IRS-III ontology is based on the WSMO conceptual model; however it has a number of differences. To achieve the goal of capability based invocation, Web services are required to have input and output roles and goals are linked to Web services via mediators. Web services are linked to goals ‘inherit’ to the goal’s input and output roles. In WSMO, the mediation service slot of a mediator may point to a goal that declaratively describes the mapping. Goals in a mediation service context play a slightly different role in IRS-III. Rather than describing a mapping, goals are considered to have associated Web services and are therefore simply invoked.

IRS clients are assumed to be able to formulate their request as a goal instance. This means that it is only required choreographies between the IRS and the deployed Web services. IRS-III choreography execution thus occurs from a client perspective [Domingue et al., 2005]. That is to say, to carry out a Web service invocation, the IRS executes a Web service client choreography which sends the appropriate messages to the deployed Web service. In contrast, current WSMO choreography describes all of the possible interactions that a Web service can have.

Language / Concrete architecture

Since IRS is more of an implementation platform than an abstract framework, in this section we focus on how it has been concretely realized. At the heart of the server is the WSMO library where the WSMO definitions are stored using the representation language OCML [Motta, 1998]. The library is structured into knowledge models for WSMO goals, Web services and mediators. The structure of each knowledge model is similar but typically the applications consist of mediator models importing from relevant goal and Web service models. Following our design principle of inspectability, all information relevant to a Web service is stored explicitly within the library.

Within WSMO a Web service is associated with an interface that contains a separate orchestration and choreography. Orchestration specifies the control and dataflow of a Web service, which invokes other Web services (a composite Web service). Choreography specifies how to communicate with a Web service. The choreography component communicates with an invocation module able to generate the required messages in SOAP format.

A mediation handler provides functionality to interpret WSMO mediator descriptions including running data mediation rules, invoking mediation services and connecting multiple mediators together. Following from the openness principle above orchestration, choreography and mediation components are themselves Semantic Web Services. At the lowest level the IRS-III Server uses an HTTP server written in lisp [Riva and Ramoni, 1996] and extended with a SOAP [SOAP, 2003] handler.

6.1.5 Relations with W<Triple>

The OWL-S approach is based on OWL. OWL was not designed to define the semantics of processes that require rich definitions of their functionality, thus OWL-S also has significantly limited expressivity. WSMO/WSML was designed to overcome this limitation by providing different layers of expressivity, thus allowing rich definitions of Web services. Moreover, OWL-S inherits some of the drawbacks of OWL [de Brujin et al., 2005a] including lack of proper layering between RDFS, less expressive species of OWL, and lack of proper layering between OWL DL and OWL Lite, on the one side, and OWL Full on the other. OWL-S provides the choice between several other languages, e.g. SWRL, KIF, etc. By leaving the choice of the language to the user, OWL-S contributes to the interoperability problem, rather than solving it. In OWL-S, the interaction between the inputs and outputs -- which have been specified as OWL classes -- and the logical expressions in the respective languages, is not clear. OWL-S does not make any explicit distinction between Web service communication and cooperation. WSMO makes this distinction in terms of Web service choreography and orchestration, thus applying the principle of separation of concerns between communication and cooperation and making the conceptual modelling all the more clear. OWL-S does not explicitly consider the heterogeneity problem in the language itself, treating it as an architectural issue, i.e. mediators are not an element of the ontology but rather a part of the underlying Web service infrastructure. WSML provides an integrated language framework for the description of both the ontologies and the services. Furthermore, the logical language used for the specification of Web service preconditions and postconditions is an integral part of the language. Thus the overall Web service description and the logical expressions that specify the pre- and postconditions are connected automatically.

The SWSF approach can be seen as an attempt to extend on the work of OWL-S and simultaneously incorporate a variety of capabilities not included in the OWL-S goals. A difference between FLOWS – the ontology part of SWSF, and OWL-S is the expressive power of the underlying language. FLOWS is based on First Order Logic, which means that it can express considerably more than can be expressed using, for instance, OWL-DL. In comparison to OWL-S, the use of First Order Logic enables a more refined approach to representing different forms of data flow that can arise in Web services. Another difference is that FLOWS tries to explicitly model more aspects of

Web services than OWL-S; this includes the fact that FLOWS can readily model process models using a variety of different paradigms and data flow between services, which is achieved either through message passing or access to shared *fluents*. Although the SWSF approach seems to tackle both conceptual modelling, as well as language issues, it is very unclear how the various differing paradigms interact. Moreover, the purpose of FLOWS was not to develop a Web language, but rather to develop a First Order Logic ontology for Web services; FLOWS does not even use URIs to specify their concepts.

Of the approaches presented in this section, only the IRS-III approach is integrated with the WSMO approach in the sense that IRS-III uses WSMO as its underlying epistemological framework. IRS-III places great emphasis on creating a capability based *broker* (facilitating the invocation of Web services through WSMO goals), ease-of-publication (being able to turn standalone code into a SWS through a single simple dialog), and tightly coupling the semantic descriptions with deployed Web services (e.g. semantic concepts and relations can be implemented as Web services). Ongoing work continues to align the two approaches.

The WSDL-S approach is a more technology centered strategy, where rather than providing a conceptual model for the description of Web services and their related aspects, it takes a bottom-up approach by annotating existing standards with metadata. WSDL-S can actually be used to represent a grounding mechanism for WSMO. WSDL-S is impartial to a specific ontology language and thereby allows Web service providers to directly annotate their services using WSML. That is, *modelReference* attributes in WSDL-S can point to concepts from WSML ontologies and the expressions in the preconditions or effects can be directly described in WSML.

6.2 Standardization Efforts

Standardization organizations such as OASIS, OMG, and W3C have established several groups or technical committees (TC) to develop and standardize SOA and the SESA vision presented in this paper. While some of these groups, such as SEE TC, directly focus on the development of the CSL layer of the SESA architecture, other groups are working on other important related aspects.

The following Technical Committees in OASIS contribute to the SESA vision:

- *OASIS Semantic Execution Environment (SEE) TC*[35] - The OASIS SEE TC aims to continue work initiated by the WSMX project and several other European Union projects from the area of Semantic Web Services. The aim of the SEE TC is to provide guidelines, justifications and implementation directions for an execution environment for Semantic Web Services. The resulting architecture will incorporate the application of semantics to service-oriented systems and will provide intelligent mechanisms for consuming Semantic Web Services.
- *OASIS SOA Reference Model (RM) TC*[36] - This OASIS TC is chartered to develop a Reference Model for Service-Oriented Architecture. This is primarily to address SOA being used as a term in an increasing number of contexts and specific technology implementations. The Reference Model is being developed to encourage the continued growth of different and specialized SOA implementations whilst preserving a common layer of understanding about what SOA is.
- *OASIS Electronic Business Service-Oriented Architecture (ebSOA) TC*[37] - This committee focuses on continuing work on the ebXML Technical Architecture to bring it to a more current architecture that takes into account both subsequent releases of the ebXML specifications and other Web services and Service-Oriented Architecture works, including the work of the W3C Web services Architecture WG.
- *OASIS SOA Adoption Blueprints TC*[38] - This committee has recognized there is a shortage of clear, demonstrable examples of working implementations of SOAs

[35] http://www.oasis-open.org/committees/semantic-ex/charter.php
[36] http://www.oasis-open.org/committees/soa-rm/charter.php
[37] http://www.oasis-open.org/committees/ebsoa/charter.php
[38] http://www.oasis-open.org/committees/soa-blueprints/charter.php

based on real needs and requirements that can be used as best practices reference. To encourage these examples SOA blueprints aim to supply an archetypal "blueprint" set of business requirements and functions that can be fulfilled by SOA methods.

- *OASIS Web Services Resource Framework (WSRF) TC*[39] - The purpose of the Web Services Resource Framework (WSRF) TC is to define a generic and open framework for modelling and accessing stateful resources using Web services. This includes mechanisms to describe views on the state, to support management of the state through properties associated with the Web service, and to describe how these mechanisms are extensible to groups of Web services.

OASIS hosts several TCs whose work relates to SESA vision, for example, ebXML Registry TC, UDDI TC, FWSI TC, SOA Adoption Blueprints TC and ebXML BP TC.

While W3C does not address issues specifically related to Service-Oriented Architectures, the results of several W3C groups are crucial to the realization of the SESA vision. These are:

- *Semantics for Web Services Characterization Group*[40] - As described in the mission statement, the Semantics for Web Services Characterization Group will to continue in the footprints of solutions like WSDL-S and study the field of applications and identify key points that are not immediately solved using Web services technologies. This characterization effort will demonstrate the existence of requirements, hence the need for one or more pieces of a framework for the use of Semantics in Web services. If it succeeds in this characterization work, the Group is expected to propose future directions of work in the domain of Semantics for Web services.
- *Semantic Annotations for WSDL Working Group*[41] - The WSDL specifies a way to describe the abstract functionalities of a service and concretely how and where to invoke it. The WSDL 2.0 specification does not include semantics in the description, thus two services can have similar descriptions while totally different meanings. The objective of the Semantic Annotations for WSDL Working Group

[39] http://www.oasis-open.org/committees/wsrf/charter.php
[40] http://www.w3.org/2005/10/sws-charac-charter.html
[41] http://www.w3.org/2005/10/sa-ws-charter.html

is to develop a mechanism to enable annotation of Web services descriptions. This mechanism will take advantage of the WSDL 2.0 extension mechanisms to build a simple and generic support for semantics in Web services.

- *Rule Interchange Format Working Group*[42] - As stated in the charter the WG will specify a format for rules, so they can be used across diverse systems. This format (or language) will function as an interlingua into which established and new rule languages can be mapped, allowing rules written for one application to be published, shared, and re-used in other applications and other rule engines.

Finally OMG has recognized importance of Service-Oriented Architectures[43] and has established an SOA Working Group the will commence in early 2006.

While SOA is widely accepted as the next generation of computing to which most software vendors are committed, standards are still evolving. From 2000 to 2005 SOA standards grew enormously in number and complexity with few reference technologies. Some standards already exist, while others are scheduled for development and release through 2012. A review of the hundreds of Web service and SOA related standards is beyond the scope of this paper. However, it is worth noting that at the end of 2005, Web service and SOA standards were being rationalized as vendors recognize that without reasonable, effective standards, SOA technology will not progress and customers will not purchase or deploy SOA solutions.

[42] http://www.w3.org/2005/rules/wg.html
[43] http://www.omg.org

6.3 Industrial Efforts

In industry, SOA, or an SOA refinement, is recognized not only as the next generation of computing, but also as the technology that will largely replace or encapsulate current technologies. This subsection summarizes the state of SOA adoption and the growth in supply and demand for semantics and knowledge technologies to enable SOA to achieve its full potential.

6.3.1 The State of SOA Adoption

A Sea-Change in IT. While service-orientation has dominated research and industry discussions of next generation computing since the 1999 IBM-Microsoft-Sun declaration on Web services, and while most software vendors announced their commitment to service-orientation starting in 2003, two "leaked" Microsoft memos dated October 2005 confirmed the likely ascension of SOA to become the dominant computing paradigm. In 1995 Microsoft launched the Longhorn project, now called Vista, to completely replace its current operating systems with a service-oriented technology that would become the platform for all Microsoft products. Even though this bet-the-company commitment to service-orientation was made in 1995, it took a decade, and presumably billions of dollars in development, for Microsoft to acknowledge the significance of service-orientation. Bill Gates' widely leaked internal memo "Internet Software Services" [Gates, 2005] acknowledged that service-orientation was causing a "Sea Change" in Information Technology. In a companion leaked memo, Ray Ozzie, Microsoft CTO, described service-oriented software development and delivery as "The Internet Service Disruption" [Ozzie, 2005]. Gates' and Ozzie's Sea Change is, at the infrastructure level, Service-Oriented Computing and at the development level, "programming" by compositional methods rather than coding. The memos also refer to higher problem solving levels and lower hardware levels. Bill Gates latent acknowledgement of service-orientation is similar to his 1995 latent acknowledgement of the Internet. Both cases are widely seen as the confirming endorsement of major moves in computing and information technology.

SOA Deployments. Most enterprises have deployed or are in the process of deploying SOA infrastructures and applications. Forrester [Gilpin et al., 2005] predicted that by the end of 2005, "77% of large enterprises, 51% of medium enterprises, and 46% of small enterprises will be actively implementing SOA." While most enterprises are deploying SOA technologies, SOA deployments are small, are at an early stage, and few have full production systems. This is due to immature SOA products (see below) and because limited SOA knowledge and experience. Gartner's fall 2005 survey of enterprises deploying Web services [Wurster et al., 2005] describes the extent of current deployments as being across all industry sectors and all sizes of companies, although deployments are more likely in large enterprises than in small to medium businesses. The use of Web services is a first step towards SOA, indicating the growing trends to SOA.

While early Web service deployments were for wrapping legacy data and for point-to-point integration, the dominant use of Web services is now primarily in information intensive applications. According to the survey, on average over 11% of IT development budgets is devoted to new Web services. Forrester estimates [Gilpin et al., 2005] that "69% [of large enterprises] are using SOA for internal integration, while 50% use SOA for external integration with business partners and customers."

While most SOA deployments are proofs of concept or early stage developments, there are a significant number of large-scale SOAs that have been in production for some time. Most published SOA case studies address the methods and technologies used but do not mention specific measures of scale. IBM has published a case study of the Standard Life Assurance Company SOA that consists of over 300 reusable business services and initiates over 40% of its mainframe-based transactions.

A large-scale production SOA would include more than 500 reusable business services and at least 5 million transactions per day. A small number of production SOAs exceed that level, including those at Wells Fargo, Google, AXA Financial, and Verizon Communications. Verizon's SOA, ITW [Havenstein, 2005], has been in production for more than three years with over 575 services in production published enterprise-wide, and over 1,000 in development. While ITW's average transaction rate is approximately 7.5 million transactions per day, ITW has topped 9 million transactions per day.

Gartner predicts [Wurster et al., 2005] a "tipping point" in the 2008-2010 period when "composite [SOA] applications" will be widely commercially available accompanied by "a shift from coding to assembly."

SOA Products. In an SOA world, most software will be service-oriented or will be wrapped to operate within an SOA. At this stage in the development of service-orientation there are two dominant SOA platforms or technologies: the SOA runtime platforms and the SOA integration platforms. SOA runtime platforms provide the infrastructure for supporting reusable Web services across an enterprise. This expansion of application servers is called an Enterprise Service Bus (ESB) defined as a "software infrastructure that enables Service-Oriented Architecture (SOA) by acting as an intermediary layer of middleware through which a set of reusable business services are made widely available." [Gilpin et al., 2005]. SOA integration platforms support process and data integration and are an extension of various integration product classes:

Enterprise Application Integration (EAI), Enterprise Information Integration (EII), Business Process Management (BPM), and many more, all of which are to converge into the SOA world. These platforms correspond roughly to Forrester's ESB suites and comprehensive ESB suites. ESB suites are "ESBs with optional components for service orchestration, service management, and partner collaboration, delivered as suites." Comprehensive ESB suites are "full-service integration suites that encompass all of the ESB suite features plus support for human workflow, vertical industry solutions, portals, rules engines, and more."

By the end of 2005 SOA technology and Web service standards matured to the point that the major software vendors had released significant SOA products. While most SOA vendors, e.g., IBM, BEA, and Sun, have separate product suites for SOA runtime and integration platforms, other SOA vendors do not make such an architectural or product distinction and offer only one of the two platforms or aspects of both categories in a single suite. Forrester [Gilpin et al., 2005] identifies the leading ESB suites to be (in alphabetical order): BEA Systems' AquaLogic Service Bus™, Cape Clear Software's Cape Clear™ 6.1, Fiorano Software's Enterprise Service Bus, IONA Technologies' Artix™, PolarLake's Integration Suite™, and Sonic Software's SOA suite. The leading comprehensive ESB suites are: BEA Systems' AquaLogic Service Bus™ plus WebLogic Integration™ plus AquaLogic Data Service Platform™ [not referenced by Forrester], IBM's WebSphere ESB [not evaluated by Forrester], Oracle's Fusion Middleware™, Sun Microsystems' Java Integration Suite™, TIBCO Software Business Works™, and webMethods' Fabric™. Microsoft's Vista™ is not listed above as Microsoft Vista™ does not offer an ESB suite but rather a core SOA infrastructure for ESBs. Microsoft's Indigo™, however, does contain some ESB features.

Forrester evaluates the above SOA platforms relative to their change, connection, and control functions. "Connection consists of protocols (including Web services support), adapters, and architecture — including support for SOA. Mediation includes transformation and mapping, repository and registry, trading partner management, and process management. And change and control includes policy management, service life-cycle support, security, and monitoring and management." [Gilpin et al., 2005]. These core SOA functions are the basis for semantic enablement of SOAs in the SESA vision.

In addition to the core SOA software products listed above, a plethora of SOA-based software products are being released and announced; and this is only the beginning

of the SOA tidal wave. The leading SOA infrastructure and integration products are listed above for two reasons; first, to show the commitment of some of the most influential software vendors and second -- for the purposes of this paper -- to identify the leading products as candidates for semantic enablement discussed in the next section.

We conclude this section in industry efforts with a reference to a significant industry research initiative on SOA and the core elements of SESA. “On the 7th of September 2005 under the auspices of the European Commission the ETP (European Technology Platform) NESSI (Networked European Software & Services Initiative) was launched. NESSI has been promoted by 13 major software and services industrial players with the aim to develop a visionary strategy for software and services by a common European Strategic Research Agenda” [NESSI, 2005]. “Promoted by thirteen major European ICT corporations, totaling almost a million jobs and about 300B € revenues, the NESSI Technology Platform aims to provide a unified view for European research in Services Architectures and Software Infrastructures that will define technologies, strategies and deployment policies fostering new, open, industrial solutions and societal applications that enhance the safety, security and well-being of citizens” [NESSI, 2005a].

NESSI represents a commitment not only to the realization of SOA but also to significant aspects of SESA. NESSI’s Research Agenda states “In order to achieve this level of automation, problems like on-the-fly composition of services, dynamic ontology-driven discovery of services and contents, contextualized enactment of services and management facilities for the ontology life-cycle need to be addressed and solved” [NESSI, 2005]. Like the SESA vision, NESSI is also committed to the support of problem solving for improved productivity and enhanced working environments.

6.3.2 Semantic Solutions: Demand and Supply

There are two sides to the semantics story – demand and supply. In addition to the demand for SOA scalability and precision are the demands for semantic solutions in several industry segments. We illustrate the demand in one industry, Pharmaceuticals, as one industrial need of SESA. We conclude our summary of demand with the greatest, long-term demand for semantic solutions, that of integration. The demand for semantic solutions of integration has a recent interesting chapter. Frustrated with ad hoc and inadequate solutions from the integration software vendors, industry groups have formed together to express commonly shared requirements for solutions to what they call semantic assurance and reconciliation. Following the demand story, we review the supply story. Software vendors are adopting semantics, ontologies, and other knowledge technologies primarily for increased automation and precision, not only in the SOA domain, but also in other domains.

6.3.3 Demand: Dynamic, Scalable SOA

Currently there is little or no demand for dynamic or scalable SOAs. Not only are the few SOAs in production relatively small, and thus not yet posing scalability problems, but also current technologies are inadequate for service discovery, negotiation, adaptation, and composition. In the real (non-automated) world, business transactions are governed by legal, regulatory, financial, tax, and other agreements or obligations. Partners who wish to automate business transactions do so after defining the terms by which automated actions must correspond to the relevant governance. At the same time, these partners establish terms of recourse or remediation in the case of failures. Such agreements, referenced above as non-functional requirements, are a long way from being dynamically discovered, selected, and enacted.

As SOA technology and deployments mature over the next five years, scale will become a significant issue. Verizon's SOA currently accounts for less than 1% of their transactions. Implementing only 25% of a large enterprise in an SOA would lead to many millions of simple and composite services and billions of transactions per day. Midwest Independent Transmission System Operator Inc., one of the world's largest energy trading markets, has a large-scale SOA [Hedin, 2005]. Midwest has 15 million customers but participates in an energy grid that supports over 30 million customers. Verizon has more than 50 million customers in its wireless business alone. Customer-facing services represent less than half of the systems of such enterprises. Supply chain, human resources, and internal operations often dwarf employee-facing systems in size and complexity. Imagine the number of services required to operate even 25% of such enterprises, the number of potential transactions, and then how these enterprises might work with partners via SOA-based information systems. A few large enterprises would alone scale beyond billions of services and transactions. Now consider how such business would operate using an SOA with no dynamic service discovery, selection, negotiation, adaptation, and composition. Manual intervention would be required to complete or approve the results of those actions. This simply will not scale.

6.3.3.1 Domain-Specific Demand: Pharmaceuticals

This section provides an industry-specific example illustrating the requirement for automated semantic solutions in domains other than integration. Similar examples could be provided for almost every industrial sector.

To accelerate new product discovery and development using information technology, the pharmaceutical industry must overcome current limitations of database technology including information overload, information extraction, information integration, information sharing, and information understanding and reasoning [Lundstroms, 2005]. All of this requires increased automation of "understanding" of information and services. Hence, Pharma IT is turning to Semantic Web technologies such as XML, RDF, and OWL.

IDC proposes [Lundstroms, 2005] that Pharma IT adopt what IDC calls a Semantic Enterprise Architecture (SEA) for the development of "semantics-enhanced applications". IDC proposes a SEA as being able to support: dynamic or adaptive discovery over enterprise information and service resources that are described semantically using standard ontologies; semantic content extraction, classification, and representation using semantic views of enterprise information and services; interoperability and composability of information; inference, reasoning, and decision support based on common semantic representations (i.e., ontologies), and ontology-based knowledge interaction.

The semantic requirements of the pharmaceutical industry over pharmaceutical information and services are a subset of the more general requirements posed above for scaling SOAs. The capabilities proposed for the SEA are generic capabilities proposed for the SESA described in this paper. Hence, Pharma IT poses a very real and critical demand for the semantically enhanced problem solving described in this paper.

Pharma IT is considering solutions in terms of the Semantic Web and in terms of services, however they do not consider SOA. The Semantic Web and Semantic Web Services are highly related but different in approaches to semantic enablement. In the Semantic Web, every resource accessible over the web has an accessible semantic

description. In an SESA, only services have semantic descriptions since service providers may wish to encapsulate or hide some information and services, as would be the case in most business applications. This paper addresses Semantic Web Services and not the Semantic Web, both of which use semantic technologies but in different architectural contexts.

6.3.3.2 Demand: Integration with Semantic Assurance

For over two decades the cost of data and process integration for an average IT project has risen from an estimated 35% to well over 50% of the total development effort. Integration represents an increasingly larger factor in information technology with increases in automation, Internet-enabled universal access, process-orientation, and automating processes between business partners. The breakthrough leading to SOA was to treat all computing as interactions between services; hence what was called integration became the core of the computational model. To understand how services must agree on the meaning of their interactions let's look at how integration evolved prior to SOA.

Unfortunately, a large number of software categories developed for integration. Process-based integration categories included Business Process Management (BPM) and workflow integration; application integration categories included Enterprise Application Integration (EAI) and many others; data integration included the largest number of categories, e.g., Enterprise Information Management (EIM), Enterprise Information Integration (EII), Extract, Transform and Load (ETL), Electronic Data Interchange (EDI), plus a large number of database integration software categories.

While each software category developed specialized solutions for the relevant integration domain, they all followed a similar pattern of recognizing the source and target to be mapped and then selecting the appropriate adapter that mapped the source format to that of the target. The general solution came to be known as semantic adaptation or synchronization since the mapping would attempt to preserve the semantics of the source process or data when mapping it to the target. More precisely, **semantic adaptation** refers to adapting or mapping a service request or data from the format of the requester (source) to that required by the receiver (target) so that the receiver can process or manipulate it in a required format while preserving its "meaning". Semantic adaptations were coded by people who understood how to map and preserve the meaning. In simple cases, when patterns can be clearly recognized, simple code generators could be used to map well-defined source formats to equally well-defined target formats. With rare exceptions, like Vitria™ and Metallect™, semantic adaptations were not automated nor did they use semantic technologies such as ontologies, taxonomies, or categories. Successful EAI, EII, and EIM products developed proprietary mapping frameworks and

large libraries of adapters. One simplification is that SOA provides a generic framework for all such mappings. Hence, these separate integration software categories will consolidate and the focus of integration will turn to the adapters, most of which are proprietary.

The demand for integration is enormous. In conventional computing, integration accounts for over 50% of application development budgets that are spent on a wide range of proprietary integration products and solutions. While the software product space may consolidate in the SOA world, the demand for semantic adaptation will increase since all Service-Oriented Computing requires “integration” of service requests and data between services.

The story gets even more interesting from a semantic point of view. In the late 1990’s, industry groups formed alliances to define and demand semantic assurance from software vendors. While it seems odd that semantic adaptation does not involve computational semantics, neither does the demand for semantic assurance and semantic reconciliation. Let’s first consider the industry demand and then consider the semantic requirement.

The concept of semantic assurance was developed by industry groups to dramatically reduce the cost of the alternative, ad hoc integration. The first industry group to do this was the Global Data Synchronization Network. They formed in 1999 in response to growing requests from the retail and consumer package goods industries in order to resolve the problem of inaccurate supply chain data. It was extended to include hardlines, private label, Direct Store Delivery, pharmaceutical, office supply, apparel, automotive aftermarket, home entertainment, chemicals, and others.

Global Data Synchronization is an industry-spanning, worldwide initiative involving Wal-Mart, Carrefour, Home Depot, Nestle, P&G and Unilever, as well as thousands of other retailers and suppliers. Global Data Synchronization — along with its attendant components, such as Global Trading Identification Numbers, Global Location Numbers and the GS1 Global Registry — is the foundation for worldwide commerce. Gartner predicts [White, 2004] that “Through 2007, businesses that use a formal, enterprise-wide strategy for GDS will realize 30 percent lower IT costs in integration and data reconciliation at the departmental level through the rationalization of traditionally separate and distinct IT projects (0.7 probability).”

Now let's consider the requirement: semantic assurance. Enterprises with large numbers of applications and databases and enterprises that transact business electronically via applications and databases must ensure that the services, requests, or information exchanged between applications and databases are treated "meaningfully," e.g., understood by the sender and receiver. This leads to the concepts of semantic assurance and semantic reconciliation. A semantic assurance service is [White and Abrams, 2005] a "service that assures semantic persistence and consistency between multiple data repositories, independent of any application, service or user request", so that each application, service, or user request need not verify semantics independently but have semantics assured by an independent, reusable service. For example, all retailers, suppliers, and other members of the same supply chain want to refer to a unique product in a common way, e.g., a unique product identifier, while allowing each member to use their own identifiers and descriptions. The architectural argument for an application-independent service is that in a service-oriented business application, individual services should not each have to address semantic assurance with each service they interact with. Enterprise level semantics should be dealt with by a semantic assurance service. This facilitates choreography for flexible, dynamic assembly and re-assembly of processes across business while maintaining relatively limited semantics concerning products.

In summary, the demand for integration solutions is higher than it has ever been, and it has until recently represented 50% of all application development. Industry groups have codified their demand that the software industry provide integration solutions that preserve relatively limited semantics. Combined with the scale of the requirement, current solutions that do not use semantic technologies will simply not suffice.

6.3.3.3 Supply: Semantically Enabled SOA and Integration

This paper proposes an SESA that will semantically enable an SOA to support automated core SOA functions: service discovery, selection, negotiation, adaptation, composition, invocation, and monitoring, as well as service interaction requiring data, protocol, and process mediation. Such an SESA should meet the demands for semantic solutions described above. This section summarizes potential and actual commitments by software vendors to use semantics and knowledge technologies in SOA and related software products to address the above needs.

Without exception, all SOA software vendors are committed to working on and achieving the relevant standards summarized in this paper. Most are developing solutions to enhance metadata management and to automate the above listed core SOA functions. Many are working on solutions that will support policies as a means of higher-level programming to govern key SOA infrastructure domains such as security, privacy, and network management, as well as key business domains such as pricing and manufacturing. Early descriptions of Microsoft's Vista™ presents the possibility of policy-defined computation models, with which user organizations could uniformly define and redefine all of the relevant parameters in their computational environment (at a high level and in a single place). Hence, the scope for semantic enablement in SOA products is even greater than that described so far in this paper. Both the demand and the potential for semantically enabled technologies are greater than we imagined.

Almost all of the solutions being developed by software vendors do not currently involve semantics, in part due to the lack of relevant standards that would provide a basis for developing interoperable semantic solutions. However, the path taken by all SOA vendors, the path defined by standards, and the paths of individual software vendors is headed down the semantic enablement path. Some relevant standards are already being augmented with semantics and many more will follow suit. Hence, all SOA vendors will start to realize SESA capabilities, feature by feature. What is more striking is both the recognition of the value of semantic enablement and the actual progress being made towards semantic enablement by some SOA vendors.

For the purposes of this paper, we asked some SOA software vendors if they were committed to investigating or adopting semantics, ontologies, or other knowledge technologies to enable SOA. Some were pleased to state their commitment. Some were pleased to state the recognition of the potential and their intent to pursue semantic solutions. Others were reluctant to respond. We conclude that the best-informed SOA vendors perceive semantic enablement as a strategic differentiator in the SOA and other software segments, such as integration, and hence they will not disclose the nature and depth of their commitment. To honour these assumed positions, we summarize our perception of vendors' commitment to the semantic enablement path.

Of the SOA vendors, IBM has the longest and deepest commitment to research and development in semantics, ontologies, and knowledge technologies. IBM outlined its commitment to semantics in a keynote speech [Spector, 2005] given at the 4th International Semantic Web Conference in November 2005 that reviewed the semantic potential of an integration framework, Unstructured Information Management Architecture (UIMA) as well as a wide range of semantic capabilities built by IBM and partners for UIMA. IBM, its partners, and the open source community, to which IBM donated UIMA, are working on a wide range of semantically oriented projects (alphaworks.ibm.com). IBM actively supports semantics-related standards and has developed semantically enabled tools including semantic matching and composition of Web services and ontology toolkits.

Many analysts perceive IBM as the leading influence on and vendor of SOA software. IBM has one of the most extensive SOA product lines. It is widely believed that WebSphere ESB, and other SOA products are semantically enabled using the results of IBM's research into ontologies and semantics. IBM declined to confirm this.

For more than a decade IBM has been one of the leaders in all forms of integration and has market-leading product offerings in many integration domains. While these products were not based on semantic solutions, they are all being integrated into IBM's SOA suites and interoperate fully with UIMA. In November 2005 IBM announced its Master Data Management Strategy directed at the semantic assurance and reconciliation. Hence, if benefits arise from semantic solutions in the SOA runtime platform, semantic solutions will likely migrate quickly to the SOA integration platforms, and *vice versa*, if they are not already there.

BEA, one of the three most influential SOA vendors, offers a comprehensive ESB suite, BEA Systems' AquaLogic Service Bus™ and WebLogic Integration™ plus a powerful integration platform AquaLogic Data Service Platform™. While the AquaLogic product set does not currently use semantics, BEA is investigating the use of semantics, including ontologies, in various areas including data integration, messaging, routing, service brokering, and security. BEA is committed to supporting the relevant Web service and SOA standards, hence will support standards that support semantics. BEA describes its SOA products as truly service-oriented in that you can choose to use one or more of the BEA SOA components in combination with other commercial of components as long as they are standards compliant. Hence, semantically enabled components, such as a registry, could be used within the BEA SOA suites.

BEA's vision of the SOA world is similar to the SESA vision described in this paper. In BEA's vision, coding is replaced by composition at three distinct levels. At the lowest level IT Composers deal with infrastructure issues such as servers. Development is done by Developer Composers who use composition tools to create composite, executable services for the use of Analyst Composers. Analyst Composers are domain experts who use services specific to their domain to solve problems. As with the DERI vision, the objective is to support domain-specific problem solving. Semantic enablement is critical to this vision.

Microsoft is the third most influential SOA vendor. Microsoft's Vista provides a complete infrastructure for Service-Oriented Computing. While Vista does not include an ESB suite, Vista provides the required infrastructure including the core SOA functions. Vista also supports models and policies at all levels of an application, as described above. Hence, Vista is a candidate for semantic enablement. Microsoft recognizes the potential and is conducting research in the area of semantic enablement.

Sun Microsystems' Java Integration Suite™ supports runtime and integration capabilities. Sun is investing heavily in developing core SOA functions to support improved automation for service registration, representation, governance, and policies. Sun's integration product, Java Integration Suite™ (previously SeeBeyond™) supports process mediation and composition and is a likely candidate for semantic enablement. Sun actively supports open standards that are leading to semantic enablement.

IONA Technologies' ESB suite, Artix™ does not currently support semantic mediation or other semantic solutions but, like BEA, provides a framework within which semantic mediation can be implemented.

Due to lacking well-defined standards for semantics (e.g. ontological standards), current semantic technologies cannot provide discrete solutions for dynamic SOA functions. This resultantly leaves semantic enablement of SOA runtime platforms further behind than semantic enablement of SOA integration platforms. The multi-billion dollar integration and adapter market discussed above will have semantically enabled solutions before we have fully fledged dynamic SOA core functions. Semantic enablement of integration has been a research and development focus for decades. A survey of semantically enabled integration products is beyond the scope of this paper. However, SOA is causing the integration market to be rationalized. Hence, semantic enablement of integration solutions placed in an SOA context will likely influence the semantic enablement of SOA core functions.

In summary, all SOA software vendors recognize the scaling and precision requirement for core SOA functions and for integration. All are either actively pursuing semantic enablement of SOA and integration or at the very least providing a framework for semantic enablement. Minimally, one SOA software vendor has semantic enablement based on ontologies within the existing product set.

6.3.3.4 Supply: Semantically Enabled Solutions

Semantically enabled solutions for core SOA functions are applicable to many problem domains. Some products already offer such solutions.

Fujitsu Laboratories of America's "Task Computing allows users to focus on the tasks they want to accomplish with computing devices rather than how to accomplish them." (taskcomputing.org) Task Computing uses semantics (OWL and OWL-S) to automate service discovery, adaptation, composition, invocation, monitoring, and dynamic creation to enable end-users to combine services from diverse sources to accomplish complex tasks. Task Computing is one of the first operational prototypes of semantically enabled SOA technology.

Since Service-Oriented Computing is message-based, efficient message delivery is critical. As a result many new solutions and products are being developed for areas such as intelligent or smart message delivery, content-based messaging routing and delivery, and intelligent networks. Cisco Systems, a leading provider of networking technologies, has developed an SOA-based product in this space. "Cisco Application-Oriented Networking (AON) technology and products operate as a set of distributed application and network services that span business, security, administrative, and network domains. The management software provides tools to effectively and uniformly address different aspects of configurability, manageability, and visibility of the system." (cisco.com). Cisco is aware of the potential of semantic enablement in this domain.

Finally, mature and critical infrastructure management technologies such as configuration management, resource management, and inventory management must all "discover" computing resources relevant to their domain and examine their state. For example, a capacity planning tool may need to first dynamically find a computing resource that has available capacity that matches a specific set of functional and non-functional requirements. The tool must then select that resource, change its state (e.g., to busy), utilize the resource, monitor its operation, free up the resource when the task is complete, and then finally record information about the task execution (e.g., for charge-back and management). These infrastructure technologies depend on capabilities identical to the core SOA functions. In late 2005, the eight major configuration

management competitors, BMC, CA, Mercury, IBM-Tivoli, Cendura, nLayers, Relicore, and Tideway acquired resource and application discovery and related core SOA-like functions. nLayers uses Managed Objects, Opsware and Bladelogic. Cesura (former Vieo) has a similar capability embedded in its monitoring system. Entuity and Smarts have the capability to do port mapping. IBM enhanced its IT services management strategy with functionality for application discovery and relationship mapping in its Tivoli product through the acquisition of Collation that uses Micromuse and Compuware. Collation's Configia product provides the ability to discover and view dependencies between applications and servers across data center applications. It shows how IT resources, such as servers and databases, are configured and the relationships between them, whether peer-to-peer or hierarchical.

7 Conclusion

This document declares and elaborates DERI's SESA Manifesto. SESA (*Semantically Enabled Service-Oriented Architecture*) is a comprehensive framework that integrates two complimentary and revolutionary technical advances, Service-Oriented Architectures (SOA) and Semantic Web, into a single computing architecture. While SOA is widely acknowledged for its potential to revolutionize the world of computing, that success depends on resolving two fundamental challenges that SOA does not address; namely, search and integration or mediation. SESA addresses that SOA semantic gap by providing following: a framework for the basis of semantic specification of SOA, means for grounding semantic specifications in each component of the SOA framework, and semantically enabled solutions for the components of the SOA framework.

In Section 2, we outlined the SESA vision of a semantically enabled SOA in terms of evolving SOA concepts and standards. We outlined some implications of a semantically enabled SOA, foreshadowing the future of resource management, architecture, and economics. And we provided a SESA-based problem-solving scenario.

Sections 3, 4, and 5 described the three layers in our SESA vision. The Problem Solving Layer, in which domain-specific problem solvers will conduct their work solely in domain-specific terms. The Common Service Layer, the technical core of SESA, is described in terms of the four building blocks of our proposed initiative. Finally, the Resource Layer, in which resources are managed to meet the needs of the Common Service Layer, was characterized in terms of objectives, challenges, and opportunities that SESA presents. The SESA framework places development aspects at the Common Services Layer and computational aspects at the Resource Layer. The real goal of computing and problem solving lies in the hands of end users at the Problem Solving Layer.

The paper concludes with a description of concrete progress already made and being made in research, standardization, and software development communities as they strive towards the realization of SOA and beyond to SESA. We envisage that a fully functioning SESA will take a decade to realize in terms of standards, products, and (what we currently call) production systems.

The SESA Manifesto defines the research challenges to which DERI is dedicated. Such significant steps and paradigm shifts are seldom achieved in isolation. The purpose of the SESA Manifesto is to provide a challenge to the industrial and research communities to collaborate in addressing the SOA semantic gap in hopes that the SOA

benefits can be fully realized. We conclude the paper with challenge to stimulate this collaboration and progression.

While scalability and precision in SOA is a challenging goal, the more basic goal of Semantically Enabled Service-Oriented Architectures (SESA) is to place semantics at the core of computer science.

7.1 The SESA Challenge

We challenge the research and industrial communities to collaborate to realize the *Semantically Enabled Service-Oriented Architecture (SESA)* vision. This challenge involves two on-going and complementary paradigm shifts in computing: the movement to service-orientation and the semantic enablement of industrial scale infrastructures and applications. Achieving such a goal will require collaboration not only within the research community, but also within the industrial community. And, in accordance to what we consider true realization – thereby defined not in terms of research prototypes but in terms of industrial scale production applications – collaboration between the research and industrial communities is of course compulsory. This will require research to understand the state and nature of the relevant industrial problems, products, and solutions. It will require industry to understand the relevant challenges and opportunities to which research can contribute. It will require research to collaborate with industry so that research results can be achieved and integrated into industrial solutions.

We define the realization of SESA by the following requirements:

- An industrial-scale application that has run successfully in a production environment for at least six months.
- The application must:
    - Support some form of collaboration substantially involving at least five internal or external, but separate, organizations between which measures of trust are partially automated.
    - Consist of at least 1,000 entity types and 1,000 service types with at least 2 subscribers per service type.
    - Have an average daily service (read and write) transaction rate of 10 million service executions.
    - Comply with at least three industrial standard ontologies and the majority of relevant SOA and Web service standards.
    - Support at least 20 concurrent domain-specific problem solvers.
    - Support problem solving, as described in this paper, in at least two distinct aspects (e.g., ordering and billing) of standard problem domain, e.g., manufacturing, financial services, health care, inventory, and tourism. This requires at least:
        - 10 industry-specific tools of which 5 must be standard industry

practice
        - 10 industry-specific problem solving capabilities supported by the tools
        - Automatic workspace configuration: The application must automatically configure the workplace for each user with the tools and capabilities required by each user as defined in their profile which defines their roles, responsibilities, and user-specified or system-deduced configuration preferences.
- At least 50% of service discovery, selection, negotiation, adaptation, composition, invocation, and monitoring -- as well as service interaction requiring data, protocol, and process mediation -- are fully automated, with no human intervention. At a minimum, this must dynamically and automatically address and resolve the conflicting non-functional business aspects that arise when a consumer discovers a service offered by a producer with whom there is no business agreement for the discovered service.
- Normal business: All normal business conventions must apply. There must be significant, e.g., legal and financial, consequences should there be a failure in any of the above automated service operations.
- All service offerings and requests are expressed in terms of service descriptions that contain: 1) a functional specification expressed in semantic terms consistent with one or more industrial standard ontologies, and 2) a non-functional specification consisting of at least 5 non-functional terms such as price, promised service levels (SLAs), and performance characteristics.

We propose this as an extension of the currently commencing Semantic Web Service Challenge (see http://www.sws-challenge.org/).

8 References

[Ahuja et al., 1986] B S. Ahuja, N. Carriero, and D. Gelernter: Linda and friends. IEEE Computer, 19:26--34, August 1986.

[Akkermans et al., 2004] Akkermans, H., Baida, Z., Gordijn, J., Pena, N., Altuna, A., Laresgoiti, I. Value Webs: Using Ontologies to Bundle Real-World Service. IEEE Intelligent Systems 19/4, 2004.

[Akkiraju et al., 2005] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M. Schmidt, A. Sheth, and K. Verma. Web Service Semantics – WSDL-S. Technical note, April 2005. Available at http://lsdis.cs.uga.edu/library/download/WSDL-S-V1.html.

[Baader et al., 2003] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.

[Baida et al., 2005] Z. Baida, J. Gordijn, B. Omelayenko, and H. Akkermans. A shared service terminology for online service provisioning. In *ICEC '04: Proceedings of the 6th international conference on Electronic commerce*.

[Brickley and Guha, 2004] D. Brickley and R.V. Guha (eds.): RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation, February 2004, http://www.w3c.org/TR/rdf-schema/

[Brodie et al., 1995] Brodie, Michael, Michael R. Stonebraker, Legacy Information Systems Migration: The Incremental Strategy, Morgan Kaufmann Publishers, San Francisco, CA (1995) ISBN 1-55860-330-1.

[Bussler, 2005] C. Bussler: A Minimal Triple Space Computing Architecture. In *Proceedings of the 2nd WSMO Implementation Workshop*, Innsbruck, Austria, June 2005.

[Carriero and Gelernter, 1989] A N. Carriero and D. Gelernter. Linda in context. Communications of the ACM, 32(4):444--458, 1989.

[Chen and Kifer, 1993] *HiLog: A Foundation for Higher-Order Logic Programming*. W. Chen, M. Kifer, D.S. Warren. Journal of Logic Programming, 15:3, February 1993, 187-230.

[Cimpian et al., 2005] Emilia Cimpian, Tomas Vitvar, Michal Zaremba (editors): Overview and Scope of WSMX. WSMX Deliverable D13.0, WSMX Final Draft v0.2, 2005, http://www.wsmo.org/TR/d13/d13.0/v0.2/.

[de Bruijn, 2005] Jos de Bruijn, editor. The Web Service Modeling Language WSML. 2005. WSMO Deliverable D16, WSMO Final Draft v0.2, 2005, http://www.wsmo.org/TR/d16/d16.1/v0.2/.

[de Brujin et al., 2005a] Jos de Bruijn, Axel Polleres, Rubn Lara, and Dieter Fensel. Owl DL vs. OWL FLight: Conceptual modeling and reasoning for the semantic web. In *Proceedings of the 14th International World Wide Web Conference*, 2005.

[De Roure et al., 2005] D. De Roure, N.R. Jennings, and N.R. Shadbolt: The Semantic Grid: Past, Present and Future. 2005.

[Dean and Schreiber, 2004] Mike Dean and Guus Schreiber, editors. OWL Web Ontology Language Reference. 2004. W3C Recommendation 10 February 2004.

[Decker and Frank, 2004] Decker, S., Frank, M.; The Social Semantic Desktop DERI Technical Report 2004-05-02, May 2004.

[Domingue et al., 2004] J. Domingue, L. Cabral, F. Hakimpour, D. Sell, and E. Motta. Irs-iii: A platform and infrastructure for creating WSMO-based Semantic Web Services. In *Proceedings of the Workshop on WSMO Implementations (WIW 2004)*, Frankfurt, Germany, September 2004. CEUR.

[Domingue et al., 2005] Domingue, J., Galizia, S. and Cabral, L. Choreography in IRS-III- Coping with Heterogeneous Interaction Patterns in Web Services, in Proceedings of 4th International Semantic Web Conference (ISWC 2005), November 6-10, 2005, Galway, Ireland.

[Fahringer et al., 2005] T. Fahringer, R. Prodan, R. Duan, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, and M. Wieczorek: ASKALON A Grid Application Development and Computing Environment. In *Proceedings of the 6th ACM/IEEE International Workshop on Grid Computing (Grid 2005)*, Seattle, November. 2005.

[Fensel, 2004] D. Fensel: Triple-space computing: Semantic Web Services based on persistent publication of information, In *Proceedings of the IFIP International Conference. on Intelligence in Communication Systems November 2004*, Bangkok, Thailand, Lecture Notes in Computer Science (LNCS), Springer-Verlag, Berlin, 2004.

[Gates, 2005]

From: Bill Gates

Sent: Sunday, October 30, 2005 9:56 PM
To: Executive Staff and Direct Reports; Distinguished Engineers
Subject: Internet Software Services

Microsoft has always had to anticipate changes in the software business and seize the opportunity to lead. …

Ten years ago this December, I wrote a memo entitled The Internet Tidal Wave which described how the internet was going to forever change the landscape of computing. Our products could either prepare for the magnitude of what was to come or risk being swept away. We dedicated ourselves to innovating rapidly and lead the way much to the surprise of many industry pundits who questioned our ability to reinvent our approach of delivering software breakthroughs. …

[Gerlernter, 1992] D. Gerlernter: *Mirrorworlds*, Oxford University Press, 1992.

[Gilpin et al., 2005] Mike Gilpin, Ken Vollmer, John R. Rymer, Lindsey Hogan: The Forrester Wave™: Enterprise Service Bus, Q4 2005: Evaluation Of Top Enterprise Service Bus Vendors Across 100 Criteria, Forrester, November 15, 2005.

[Gordijn et al., 2001] Gordijn, J., J. Akkermans, J. van Vliet. Designing and Evaluating E-Business Models, IEEE Intelligent Systems, 16/ 4, 2001.

[Grosof et al., 2003] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: Combining logic programs with description logic. In Proc. Intl. Conf. on the World Wide Web (WWW-2003), Budapest, Hungary, 2003.

[Grosof, 1999] *A Courteous Compiler From Generalized Courteous Logic Programs To Ordinary Logic Programs*. B.N. Grosof. IBM Report included as part of documentation in the IBM CommonRules 1.0 software toolkit and documentation, released on http://alphaworks.ibm.com. July 1999. Also available at: http://ebusiness.mit.edu/bgrosof/#gclp-rr-99k.

[Gruninger, 2003] *A Guide to the Ontology of the Process Specification Language.* M. Gruninger. *Handbook on Ontologies in Information Systems.* R. Studer and S. Staab (eds.). Springer Verlag, 2003.

[Hakimpour et al., 2004] Hakimpour, F., Domingue, J., Motta, E., Cabral, L. and Lei, Y. (2004) Integration of OWL-S into IRS-III, Proceedings of the first AKT Workshop on Semantic Web Services.

[Haller et al., 2005] A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler: WSMX - A Semantic Service-Oriented Architecture. In *Proceedings of the International Conference on Web Service (ICWS 2005)*, 2005.

[Havenstein, 2005] Heather Havenstein: Hammering Out Web Services Links: Verizon's IT Workbench SOA lets the company use Web services to integrate disparate systems, Computerworld, April 18, 2005.

[Haystack, 2003] Haystack: A Platform for Authoring End User Semantic, Web Applications. International Semantic Web Conference 2003: 738-753.

[Heffner, 2005] Randy Heffner: Your Strategic SOA Platform Vision: Crafting Your Architectural Evolution To Service-Oriented Architecture, IT View Research Documents, March 29, 2005, by Randy Heffner, Forrester.

[Hedin, 2005] Marianne Hedin: SOA and Web Services Keep the Lights on for Millions of U.S. and Canadian Households: A Case Study of Midwest ISO, IDC #34374, IDC, December 2005.

[Hepp, 2005] M. Hepp, "Product Representation in the Semantic Web", 2005, Working Paper available at http://www.heppnetz.de.

[Hey, 2005] T. Hey: On the Need for Web Services Standards, *Grid Today, Daily News and Information for the Global Grid Community*, vol 4(1), 2005, http://www.gridtoday.com/05/0110/104428.html.

[ISTAG, 2001] ISTAG: Scenarios for Ambient Intelligence in 2010. European Commission, Feb. 2001. Available at http://www.cordis.lu/ist/istag.htm

[Johanson and Fox, 2004] B. Johanson and A. Fox: Extending Tuplespaces for Coordination in Interactive Workspaces, *Journal of Systems and Software*, 69(3), January 2004:243-266.

[KIF, 1998] Knowledge Interchange Format: Draft proposed American National Standard (dpans). Technical Report 2/98-004, ANS, 1998. Also at http://logic.stanford.edu/kif/dpans.html.

[Kifer et al., 1995] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. JACM, 42(4):741-843, 1995.

[Klyne and Carroll, 2004] G. Klyne and J. J. Carroll (eds.): Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recommendation, February 2004, http://www.w3.org/TR/rdf-concepts/.

[Kotler, 1988] Kotler, P. Marketing Management: Analysis, Planning, Implementation and Control, 6th edition. Prentice Hall, Englewood Clis, NJ, 1988.

[Krummenacher et al., 2005] R. Krummenacher, J. Kopeck˝, and T. Strang: Sharing Context Information with Semantic Spaces. In Proceedings of the Workshop on Context-Aware Mobile Systems (CAMS 2005) as part of OnTheMove Federated Conferences (OTM 2005), Agia Napa, Cyprus, 30 October - 4 November, 2005.

[Krummenacher and Strang, 2005] R. Krummenacher and T. Strang: Ubiquitous Semantic Spaces. In *Proceedings of the 7th International Conference on Ubiquitous Computing (UbiComp 2005)*, Tokyo, Japan, September 11-14, 2005.

[Lloyd, 1987] *Foundations of logic programming (second, extended edition)*. J. W. Lloyd. Springer series in symbolic computation. Springer-Verlag, New York, 1987.

[Lundstroms, 2005] Scott Lundstroms: Semantic Web and Pharma IT: Creating a New Infrastructure for Pharmaceutical Knowledge Management, LifeScience Insights, IDC, August 2005.

[Mendling and Nüttgens, 2003] Mendling, J., Nüttgens, M. XML basierte Geschäftsprozessmodllierung. In: Uhr et al. (eds) Proc. of Wirtschaftsinformatik, 2003, Band 2. Dresden, Germany.

[MOF, 2002] Object Management Group Inc. (OMG). Meta object facility (MOF) specification v1.4, 2002.

[Motta, 1998] Motta, E. (1998). An Overview of the OCML Modelling Language. In proceedings of the 8th Workshop on Knowledge Engineering Methods and Languages (KEML '98).

[Nau et al., 2004] D. Nau, M. Ghallab, and P. Traverso. Automated Planning: Theory & Practice. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[NESSI, 2005] NESSI Strategic Research Agenda: Invitation for Contribution, Networked European Software & Services Initiative (NESSI) www.nessi-europe.com, November 4, 2005.

[NESSI, 2005a] Vision Document, Networked European Software & Services Initiative (NESSI) www.nessi-europe.com, May 23, 2005.

[Noia et al., 2003] T. Di Noia, E. Di Sciascio, F. M. Donini, and M. Mongiello, "A System for Principled Matchmaking in an Electronic Marketplace", presented at Twelfth International World Wide Web Conference (WWW2003), Budapest, Hungary, 2003.

[O'Sullivan et al., 2002] O'Sullivan, J., Edmond, D., ter Hofstede, A. H. M. Service description: A survey of the general nature of services, report FIT-TR-2003-02. 2002.

[Osterwalder and Pigneur, 2002] Osterwalder, A., Pigneur, Y. (2002) An e-Business Model Alexander Osterwalder, A., Pigneur, Y. An e-Business Model Ontology for Modeling e-Business. Proc. of the 15th Bled Electronic Commerce Conference, 2002.

[OWL-S, 2004] The OWL Services Coalition. OWL-S 1.1. Available from http://www.daml.org/services/owl-s/1.1/, November 2004.

[Ozzie, 2005]

From: Ray Ozzie

Sent: Friday, October 28, 2005
To: Executive Staff and direct reports
Subject: The Internet Services Disruption

… But we bring these innovations to market at a time of great turbulence and potential change in the industry … This isn't the first time of such great change: we've needed to reflect upon our core strategy and direction just about every five years. Such changes are inevitable because of the progressive and dramatic evolution of computing and communications technology, because of resultant changes in how our customers use and apply that technology, and because of the continuous emergence of competitors with new approaches and perspectives. …

[PDDL, 1998] PDDL-The Planning Domain Definition Language V. 2. Technical Report, report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[Preist, 2004] Chris Preist. A conceptual architecture for semantic Web services. In *3rd International Semantic Web Conference (ISWC2004)*. Springer Verlag, November 2004.

[Prodan and Fahringer, 2005] Radu Prodan and Thomas Fahringer. Dynamic Scheduling of Scientific Workflow Applications on the Grid using a Modular Optimisiation Tool: A Cae Study. In 20th Symoisum of Applied Computing (SAC 2005), ACM Press, Mar. 2005. Santa Fee, 2005.

[Riva and Ramoni, 1996] Riva, A. and Ramoni, M. (1996) LispWeb: a Specialised HTTP Server for Distributed AI Applications. Computer Networks and ISDN Systems, 28, 7-11, 953-961.

[Roman et al., 2005] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Christoph Bussler, and Dieter Fensel: Web Service Modeling Ontology, Applied Ontology, 1(1): 77 - 106, 2005.

[Schmid, 1993] Schmid, B. Elektronische Märkte. Wirtschaftsinformatik 35/5, 465-480, 1993.

[Siddiqui et al., 2005] Mumtaz Siddiqui, Alex Villazon, Juergen Hofer, and Thomas Fahringer. GLARE: A Grid Activity Registration, deployment and provisioning framework. In Proceedings of the International Conference for High Performance Computing, Networking and Storage (Supercomputing 2005), Washington, USA, November 12-18 2005.

[Sirin et al., 2004] Sirin, E., Parsia, B., Hendler, J. Filtering and Selecting Semantic Web Services with Interactive Composition Techniques. IEEE Intelligent Systems 19/4, 2004.

[SOAP, 2003] XML Protocol Working Group. Soap version 1.2. Technical report, June 2003. W3C Recommendation. Available from http://www.w3.org/TR/2003/REC-soap12-part0-20030624/.

[Spector, 2005] "The Practical Web" - Semantic Acceleration Helping Realize the Semantic Web Vision stated some public position on semantics. Keynote ISWC 2005 (http://iswc2005.semanticweb.org/) Dr Alfred Spector, Vice President of Strategy and Technology IBM Software Group.

[Strang, 2003] Strang, T.: Service Interoperability in Ubiquitous Computing Environments. PhD thesis, University of Munich, Germany, 2003.

[Strang et al., 2003] Strang, T., Linnhoff-Popien, C., Roeckl, M.: *Highlevel Service Handover through a Contextual Framework*. 8th International Workshop on Mobile Multimedia Communications, Munich, Germany, October 5-8 2003, Center for Digital Technology and Management, Proceedings of MoMuC 2003, (2003).

[SWRL, 2003] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. Swrl: A semantic web rule language combining owl and ruleml, 2003. Available at http://www.daml.org/2003/11/swrl/.

[SWSF, 2005] Semantic Web Services Framework. SWSF Version 1.0. Available from http://www.daml.org/services/swsf/1.0/, 2005.

[Timmers, 2001] Timmers, P. Electronic Commerce: Strategies and Models for Business-to-Business Trading. John Wiley & Sons, NJ., 2001.

[Truong and Fahringer, 2004] H.L. Truong and Thomas Fahringer. SCALEA-G. a unified monitoring and performance analysis system for the Grid. Scientific Programming, vol. 12, no. 4, pp. 225-237, 2004, IOS Press.

[Vervest et al., 2005] Vervest, P., van Heck, E., Preiss, K., Pau, L. (eds) Smart Business Networking. Springer Verlag, 2005.

[Weibel et al., 1998] S. Weibel, J. Kunze, C. Lagoze, and M. Wolf. Dublin core metadata for resource discovery. RFC 2413, IETF, September 1998.

[Werthner, 2003] H. Werthner: Intelligent Systems in Travel and Tourism. In Proceedings of International Joint Conference on Artificial Intelligence (IJCAI 2003), Acapulco, Mexico, 2003.

[Werthner and Ricci, 2004] Werthner, H. and Ricci, F. (2004) Electronic Commerce and Tourism. *Comm. of the ACM, 47/12.*

[White, 2004] Andrew White: The Pressure to Participate in GDS Is Building, Document TU-21-4076, Gartner, August 20, 2004.

[White and Abrams, 2005] Andrew White and Charles Abrams: Service-Oriented Business Applications Require EIM Strategy, Gartner, March 30, 2005.

[Wieczorek et al., 2005] Marek Wieczorek, Radu Prodan, Thomas Fahringer: Scheduling for scientific workflows in the ASKALON Grid environment. ACM SIGMOD 34(3):56-62, 2005.

[Williamson, 1985] Oliver Williamson: The Economic Institutions of Capitalism: Firms, Markets, Relational Contracting. New York: The Free Press, 1985.

[WSDL, 2005] Web Services Description Language (WSDL) Version 2.0. Last Call Working Draft, W3C WS Description Working Group, August 2005. Available at http://www.w3.org/TR/2005/WD-wsdl20-20050803.

[Wurster et al., 2005] Laurie F. Wurster, Fabrizio Biscotti, Michele Cantara: Web Services User Survey for the U.S. and Europe, 2005, Gartner/DataQuest, December 7, 2005.

[Zeithaml and Bitner, 1996] Zeithaml, V., Bitner, M. J. Services Marketing. McGraw-Hill, New York, NY, 1996.