Semantic Web Services

Rudi Studer · Stephan Grimm ·
Andreas Abecker (Eds.)

Semantic Web Services

Concepts, Technologies, and Applications

With 102 Figures

Springer

Editors

Rudi Studer
Universität Karlsruhe
Inst. Angewandte Informatik und
Formale Beschreibungsverfahren
76128 Karlsruhe
Germany
studer@aifb.uni-karlsruhe.de

Andreas Abecker
Forschungszentrum Informatik (FZI)
Haid-und-Neu-Str. 10-14
76131 Karlsruhe
Germany
abecker@fzi.de

Stephan Grimm
Forschungszentrum Informatik (FZI)
Haid-und-Neu-Str. 10-14
76131 Karlsruhe
Germany
grimm@fzi.de

Library of Congress Control Number: 2007923415

ACM Classification: H.4, D.2, I.2, J.1

ISBN 978-3-540-70893-3 Springer Berlin Heidelberg New York

Typesetting by the editors
Production: Integra Software Services Pvt. Ltd., India
Cover design: KünkelLopka, Heidelberg

Printed on acid-free paper 45/3100/Integra 5 4 3 2 1 0

Contents

Introduction

Rudi Studer, Stephan Grimm and Andreas Abecker

FZI Research Center for Information Technologies, University of Karlsruhe, Germany
{studer,grimm,abecker}@fzi.de

Motivation for the Topic of this Book

Web Service (WS) technology and the idea of a Service-Oriented Architecture (SOA) for web-based, modular implementation of complex, distributed software systems seem to become a tremendous success [17, 9]. In just a few years, the service-oriented approach not only gained considerable interest in Computer Science research, but was also taken up with a unique unanimity by all big international players in the IT industry, such as IBM, Microsoft, Hewlett Packard, and SAP.

Distributed software systems conquer more and more fields of daily life,[1] and the software itself becomes more and more powerful. On the other hand, such software systems also become increasingly complex, and the software bridges more and more between formerly separated, *heterogeneous* areas.[2] Hence, the matter of how to structure modular systems and how to achieve interoperability between heterogeneous parts becomes a key to success. The effective and efficient realisation of such modular, interoperable, large-scale software systems is facilitated by Web Services and SOA because they provide a *standardised* architecture for modular systems, for creating new functionality from existing building blocks, and for enabling communication between heterogeneous elements.

In contrast to the former, technologically well-founded, approaches that addressed (at least partially) similar goals, such as CORBA or Multi-Agent Systems, the current approach seems to have some striking advantages:

- It is simple, based on simple, open protocols.
- These protocols does not require or require only a limited amount of additional software.

[1] See, for instance, the ever growing importance of embedded software systems in the automotive area, or the thrilling new opportunities opened up by Ubiquitous Computing and RFID technology

[2] Consider, e.g., cross-platform implementations, cross-department workflows, cross-organisational application integration, or even cross-national eGovernment processes

- It is a conservative extension of accepted Internet standards which proved to work, also platform independently.
- It allows easy encapsulation of existing code and applications.

Maybe even more important for the success of SOA than purely technological issues is the fact that times have changed to some extent:

- Standardisation of some levels of software communication is commonly accepted through the success of the Internet.
- Supporting business processes and understanding business logic becomes more important than mastering low-level computer functionality.
- Interoperability in a networked world is going to be considered a bigger competitive advantage than binding customers through proprietary software and protocols.

All such considerations led to an atmosphere which facilitates the widespread industrial take-up of ideas like SOA and Web Services. Nevertheless, it would be an illusion to think that we have already found the golden bullet for solving all problems of interoperability in heterogeneous systems, as required for Enterprise Application Integration or Business-to-Business solutions. Essentially, Web Service standards provide a communication medium for distributed systems, but they cannot yet ensure that the communicating parties “speak the same language”–which is necessary for smooth, fully automated system interoperation.

For illustrating the deficiencies of existing SOA solutions, let us use the following analogy. If two parties want to communicate, they might want to send a letter by mail. Hence they need paper and pen for writing, they need stamps, postal offices, etc. which provide a transport infrastructure. They also need some addressing schemes and coding standards, such as ZIP codes. All this can be considered as given in the SOA approach with its lower levels for message transport, etc. For really communicating, our two parties also need to know the grammar and the lexicon of the English language. Even this might be considered as given in SOA technology, e.g. through the Web Service Description language.

However, such standards for structure, syntax, and vocabulary of Web Service functionality do not yet offer the semantics and the pragmatics of the used vocabulary. Software systems cannot know that the words bank and credit institute may be used in many cases as synonyms; that a flower shop in particular sells roses, tulips, and cloves because they are flowers; that flower shops *may* offer seedlings of salad or vegetables, but not always do – if a certain shop does not, a market-garden is the more appropriate address; that flower shops sometimes also sell greeting cards because they go often along with a bouquet of flowers; or that a market-garden may also produce the flowers for the flower shop, and thus might offer cheaper prices, but less floristic services. Coming back to our analogy, we can say that by writing a letter, *real* communication between the two parties is only achievable if they both share a common understanding of how language refers to concepts prevalent in the real world, and if both know which constraints and which background knowledge is typically associated with these concepts.

All such common-sense reasoning as well as general or business-specific background knowledge are typically not available in a computer system. Hence it is still often the case that also in today's most developed service-oriented software scenarios, there is much manual, human intervention required in order to interpret the semantics of informal descriptions of service functionality, or in order to harmonise incompatible data schemata or communication protocols.

Semantic Web Technology

This is the moment where Semantic Web (SW; see [1]) technology comes into play. Its aim is exactly to harmonise semantical discrepancies in software systems by providing machine-interpretable semantics, and to "understand" ambiguous descriptions – thus achieving a new quality of intelligent and automated information processing in the web [6, 28].

This is done on the basis of semantically rich meta data for webpages, for web-accessible data or multimedia resources, etc. This meta data is expressed in powerful, logic-based, representation languages (which are in part already standardised by the World Wide Web Consortium W3C) that refer to the controlled vocabulary of shared and quasi standardised domain knowledge models, so-called ontologies [10, 29]. The ultimate goal of such an approach–based upon formal, expressive languages and shared, controlled vocabularies–is to make semantics machine-processable to a much bigger extent than it is today.

Semantic Web Services

Semantic Web Services (SWS) employ such Semantic Web technology in the Web Services area: service functionality, Web Service inputs and outputs, their preconditions and effects, etc., all are expressed in knowledge representation languages, referring to shared ontological vocabularies [20, 30, 3, 16]. In this way, a higher degree of automation and more precise results can be achieved:

- When searching for a service providing a specific functionality, ontologies and associated thesauri can provide synonyms of words, the taxonomic structure of service capabilities, relationships between service capabilities, etc.
- When trying to harmonise different data formats for two services which have to exchange messages, ontologies can provide elaborated conceptual data models for message descriptions which facilitate automated translation.
- When mediating different communication protocols of services to work together, highly expressive Semantic Web languages can provide well-founded means to describe interaction patterns in communication protocols.
- When trying to compose complex business processes from given partial processes implemented by a number of Web Services, automated planning algorithms from Artificial Intelligence can be employed, provided the semantics of the input services is formally defined.

About this Book

This book aims to be a self-contained compendium of material for newcomers in the field, starting with the basics, and also coming to a level of technical depth which is sufficient to start one's own concrete technical work in the area. We aim at providing the necessary theoretical and practical knowledge for understanding the essential ideas and the current status of Semantic Web Services research. The reader should be familiar with Computer Science basics and fundamental terminology; prior knowledge in Semantic Web technology or Artificial Intelligence is useful, but not required. The book mainly addresses advanced Computer Science students or researchers, as well as practitioners with a good theoretical background, interested in the future of computing. It provides a snapshot of ongoing research in the SWS area and might be used as a supplementary textbook for Semantic Web, Artificial Intelligence, Web Services, or Middleware lectures. It shall also serve as an introductory and survey volume for IT professionals who prepare the step from Web Service programming to Semantic Web Services or who want to assess the potential of this new technology.

In order to achieve these goals, we followed some principles guiding the preparation of this book:

- The book aims at a *complete coverage of the topic and its background.* Therefore, we included in Parts I and II of the book introductory chapters on Web Services and SOA, as well as on the most important Semantic Web fundamentals, in order to provide all necessary prior knowledge for the SWS topic.
- The aim of a comprehensive discussion of Semantic Web Services also led to the goal of finding a *balance between theoretical foundations and practical or practice-oriented topics.* This led to the decision of discussing in Part III of the book the overall SWS life cycle and technology foundations in a principled survey manner, whereas Part IV contains concrete application examples and implementation issues.
- We tried to have all chapters reasonably *self-contained*, such that they could be taken (by a reader familiar with the required background) as stand-alone papers, also including their own list of references. Definitely, the several parts of the book can be read stand-alone.
- Although we had this aim of providing relatively self-contained chapters, we also tried to ensure a maximum level of *consistency between chapters*, meaning that we avoided redundancies and tried to ensure a consistent use of terminology and overall idea of SWS – which is mainly based on Chap. 6.
- The book is not committed to a specific knowledge representation or service description approach (such as OWL-S or WSMO), but tries to give a fair and comprehensive account of today's existing solutions.

It should be noted that the SWS topic is still pretty young; by far not all technical discussions are completed yet nor is any technical basis finally standardised. Consequently, this early stage of SWS research is also reflected in the content of the various chapters and their level of overall integration.

Structure of the Book

The structure of this book, divided into parts and chapters, is as follows.

Part I briefly presents the basics of current, non-semantic Web Services and SOA technology:

- Chapter 1 motivates the basic idea of Web Services and SOA.
- Chapter 2 introduces the Web Service technology stack and technical fundamentals of SOA, and thoroughly discusses the most important standards, protocols, and basic technologies underlying the approach (such as SOAP, WSDL, and UDDI).

Part II introduces major ideas and some basic technology of the Semantic Web:

- Chapter 3 introduces the basic ideas of knowledge representation and processing, in particular with respect to ontologies as a key feature of the Semantic Web.
- Chapter 4 gives a pragmatic introduction to ontology engineering.
- Chapter 5 explains the overall Semantic Web idea with ontology-based meta data, and meta data annotation of Web resources as its core concepts.

Part III presents the major principles and technological components of the SWS approach:

- Chapter 6 sketches the overall vision and idea of SWS and introduces the basic notions used in the subsequent chapters.
- Chapter 7 shows the principles and the major, widespread approaches for SWS description by ontology-based meta data.
- Chapter 8 illustrates the usage of such semantic description for precise discovery and selection of Web Services.
- Chapter 9 discusses several ways of how to compose complex Web Services from simpler ones.
- Chapter 10 identifies various kinds of heterogeneity prevalent in SWS scenarios and shows ways to overcome them with semantic mediation technologies.

Part IV illustrates implementation and application aspects of Semantic Web Services:

- Chapter 11 gives an impression of contemporary, implemented SWS technology by discussing basic tool categories for Semantic Web Services and showing many example implementations.
- Chapter 12 uses elements of the SWS technology as introduced in the former parts of the book for adding a new functionality to existing legacy systems in the area of Electronic Government. Concretely, an approach is shown which supports tracking of changes in an evolving world down to the affected service implementations.
- Chapter 13 describes some more application examples in the domain of Electronic Government. Here, the focus is on interoperability of different software systems.

- Finally, Chap. 14 shows two applications of SWS technology in the eHealth area. Again, interoperability is a major aim, and also the easier use of new mobile technologies is addressed.

Practical Relevance of the SWS Topic

At the time of editing this book, Semantic Web Services were a highly active research and development topic. Initiatives such as OWL-S, WSMO, IRS-III, or METEOR-S have gained a high level of visibility and produced valuable research results. Issues such as intelligent service discovery or fully automated service composition were subject to widespread ongoing research in many labs. Standardisation efforts such as OWL-S, WSMO, WSDL-S or SWSF (all submitted to W3C and partially discussed in OASIS and OMG) have found their way into relevant standardisation processes. Big IT companies like Hewlett Packard, SAP, and IBM have taken up the topic in their research agendas and belong to the major drivers in the field. Semantic Web Service approaches are investigated as a base technology for supporting other relevant Web Service issues such as policy modelling or quality of service [15, 32]. Other approaches to distributed computing, such as Peer-to-Peer or Grid computing, settle upon Web Services as an underlying technology and can thus also be "lifted" to Semantic Peer-to-Peer or Semantic Grid computing [14, 23].

However, regarding real-world practical applications, Semantic Web Services are still looking for their "killer applications". In this book, we included case studies from the healthcare and the government area. Both are perfect application domains for SWS[3]; however, in eBusiness, it is not yet clear which scenarios definitely need SWS functionality – although company-internal application integration (EAI) and cross-organisational business processes in Business-to-Business (B2B) relationships were a main driver for the development of SWS technology. There is a number of published applications, mostly in the prototype status:

- Logistics – In [25], logistics supply chains are generated on-the-fly, while changing availability of transportation alternatives may require real-time reconfiguration of service networks.

[3] eHealth and eGovernment seem to be fruitful for a number of reasons: both are characterised by a huge number of parties, the software of which should seamlessly interoperate (e.g. all doctor's surgeries with all hospitals and all health insurance companies); interoperability is a critical issue since both domains face a strong pressure for significant cost reductions; moreover, legal regulations enforce a better software process interoperability in some fields of eGovernment; traditionally, both domains are to some extent resistant against some market mechanisms which reduce interoperability problems in some eBusiness scenarios (e.g. when a big Original Equipment Manufacturer presses its suppliers or vendors to use a specific software that is compliant with its own systems, or when a certain de-facto standard arises for any economic reason which does not apply to public authorities); last but not the least, in spite of their huge heterogeneity, both areas have some tradition in standardisation and are thus prepared for the use of ontologies etc.

- Tourism – In [12, 7, 33], travel Web Services are composed for a virtual travel agency in an automated way, in order to loosen the currently centralised structures of the travel business.
- Collaborative work – Reference [11] presents a simple demonstrator for the ad hoc composition of virtual teams, exploiting semantic descriptions for matchmaking of appropriate collaborators and for facilitating interoperability of involved software applications.
- Finance – Reference [5] demonstrates automatic selection and composition of account monitoring and message delivery Web Services in an eBanking scenario where a user is automatically notified if he/she is financially overcommitted.
- Telecommunication – Reference [8] explains how British Telecom aims at an easier integration of new business partners into their BT Wholesale's B2B Gateway through SWS technology.
- Bioinformatics – Reference [24] employs SWS methods and models for a semantic workflow tool which configures and manages complex workflows for processing information about protein sequences in genes. Similar approaches are also under work in other bioinformatics labs. Reference [18] compares several architectural alternatives for semantics-based bioinformatics software, and draws some general conclusions about the potential for applying SWS technology in bioinformatics. We suspect that a similar application potential exists also in other eScience domains with complex information processing tasks.
- Business Intelligence (BI) – similarly to the above-mentioned bioinformatics example, [27] uses the IRS-III Semantic Web Service framework for integrating heterogeneous applications and for reusing code of existing BI software. Such a usage of SWS technology, namely easier web-based construction of software workbenches from existing code, seems to be possible and useful in many other domains too.
- Geographic Information Systems (GIS) – Recently, more and more spatial-related data becomes publicly available and opens up new opportunities for space-oriented information services which combine different information streams at runtime, within a given context. To this end, the integration of manifold heterogeneous data at different layers of abstraction is important. For instance, [31] describes an emergency management system based on a Semantic Web GIS, with SWS as the technological basis for achieving data interoperability and for interfacing different software services.

The examples above illustrate potential SWS use cases and show under which conditions the provided functionality can be used beneficially. It seems decent to think that large-scale SOA installations with thousands of available services and high expectations with respect to process automation cannot be realised at all *without* models of formalised semantics and powerful inferences acting upon them. However, the lack of widespread industrial take-up of SWS technology shows that practitioners are not yet fully convinced. Nevertheless, independent from possible future SWS usage scenarios in eBusiness or eScience, we suspect a remarkable success of SWSs in two further areas: Semantics-Based Software Engineering and Pervasive Computing.

Semantics-Based Software Engineering

Let us call our first vision *Semantics-Based Software Engineering* (SBSE, cp. [21, 22]). Imagine a software engineering scenario within a company that often builds large-scale software solutions from many components (modules, packages, classes, etc.) with different functionality, many of them being slight variations of others. In such a situation, the semantic description of components could facilitate the manual discovery of reusable components by employing well-known techniques from ontology-based information retrieval, thus increasing significantly the programmers' productivity. Components could be linked with supporting documentation, FAQs, example usages, etc. Usage constraints and interdependencies with other components would be modelled formally to enable automated consistency checks. Moreover, the semantic description of general, as well as domain-specific usage policies would facilitate automated policy enforcement for checking the consistency of system configurations, or for tracing the effects of policy changes. While the "general" SWS scenario strives for full automation, the SBSE vision keeps the human in the loop: software development tasks are supported, facilitated, and controlled by the system, thus leading to an approach which is much more realistic in the short term.

Pervasive Computing and Ubiquitous Intelligence

Another scenario, much more ambitious than the aforementioned, is the idea of *Ambient*, *Ubiquitous Intelligence*, *or Pervasive Computing*, where human–computer interaction is supported by networked physical devices which act as sensors or as actuators and are embedded in our clothes or in our everyday working and living environments, tools, or electrical appliances. For instance, the MyCampus project at Carnegie Mellon University [26] describes ubiquitous, context-aware, personalised information services in three sample domains: at a University campus, in a museum, and in a smart office environment. Reference [4] presents context-aware, policy-based, and personalised computing services in a smart meeting room. All such scenarios imply the seamless ad hoc interoperability of a variety of software components, which requires a high degree of automation in composition and mediation. Moreover, the implementation of intelligent system behaviour can benefit from the higher level of abstraction provided by declarative modelling of policies, context, behaviour, etc.

Acknowledgments

The work presented in this book is the result of cooperation in many inspired and committed teams, which could not have taken place without the generous financial support by many public and private institutions and organisations. At this place, we want to thank all these co-financing partners which made ambitious IT research possible and which helped to shape the future of our working environments.

Let us mention with special emphasis the European Commission (EC) which co-funded two ground-breaking research projects laying the foundations for a widespread adoption of SWS approaches in Europe:

1. SWWS (Semantic Web Enabled Web Services, funded by the EC under grant FP5-IST-2001-37134) was probably the first endeavour to join forces of several European research institutions and commercial players to come to a common, eBusiness-driven vision of Semantic Web Service technology as a basis for Enterprise Application Integration and Business-to-Business Interoperation.
2. DIP (Data, Information and Process Integration with Semantic Web Services, funded by the EC under grant FP6-IST-507483) continued the SWWS efforts and came up with the Web Service Modelling Ontology (WSMO), an ontology-based, comprehensive SWS framework.

Many of the chapters benefit from the work done in these two projects. Let us also mention especially the European OntoGov project (Ontology-enabled eGov Service Configuration, funded by the EC under grant FP6-IST-507237) which substantially supported the editors' work. OntoGov employed semantics-based service modelling methods for change management as part of service management in Electronic Government.

For the US authors, we have to mention DARPA which considerably supported the success of SWS research through its DAML programme.

Other projects which supported some of our chapter authors, include the following:

- ARTEMIS (A Semantic Web Service-Based P2P Infrastructure for the Interoperability of Medical Information Systems, funded by the EC under grant FP6-IST-002103)
- ASG (Adaptive Service Grid, funded by the EC under grant FP6-IST-004617)
- COCOON (Building Knowledge Driven and Dynamically Adaptive Networked Communities within Healthcare Systems, funded by the EC under grant FP6 IST-507126)
- CollaBaWü (Collaborative, Component-Based Business Application Software Development within the Financial Service Provider Domain in Baden-Wüerttemberg, funded by the German Federal State of Baden-Württemberg)
- DERI-Lion (funded by Science Foundation Ireland)
- Esperonto (Application Service Provision of Semantic Annotation, Aggregation, Indexing and Routing of Textual, Multimedia, and Multilingual Web Content, funded by the EC under grant FP5-IST-2001-34373)
- FIT (Fostering Self-Adaptive e-Government Service Improvement Using Semantic Technologies, funded by the EC under grant FP6-IST-027090)
- InfraWebs (Intelligent Framework for Generating Open (Adaptable) Development Platforms for Web-Service Enabled Applications Using Semantic Web Technologies, Distributed Decision Support Units and Multi-Agent Systems, funded by the EC under grant FP6-IST-511723)

- Knowledge Web (Network of Excellence, funded by the EC under grant FP6-507482)
- Monadic Media (funded under the ITEA scheme by the government of Italy)
- RW2 (Reasoning With Semantic Web Services, funded by the Austrian government in the FIT-IT programme)
- SEKT (Semantically Enabled Knowledge Technologies, funded by the EC under grant FP6-IST-506826)
- TSC (Triple-Space Computing, funded by the Austrian government in the FIT-IT programme).

References

1. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284, May 2001.
2. C. Bussler, J. Davies, D. Fensel, and R. Studer, editors. *The Semantic Web: Research and Applications, First European Semantic Web Symposium, ESWS 2004*, volume 3053 of *LNCS*. Springer-Verlag, 2004.
3. L. Cabral, J. Domingue, E. Motta, T.R. Payne, and F. Hakimpour. Approaches to Semantic Web Services: an Overview and Comparisons. In *[2]*, pages 225–239, 2004.
4. H. Chen, T. Finin, A. Joshi, L. Kagal, F. Perich, and D. Chakraborty. Intelligent Agents Meet the Semantic Web in Smart Spaces. *IEEE Internet Computing*, 8, November/December 2004.
5. J.M. López Cobo, S. Losada, Ó. Corcho, V.R. Benjamins, M. Niño, and J. Contreras. SWS for Financial Overdrawn Alerting. In McIlraith et al. [19], pages 782–796.
6. J. Davies, R. Studer, and P. Warren, editors. *Semantic Web Technologies – Trends and Research in Ontology-based Systems*. John Wiley & Sons, 2006.
7. J. Domingue, S. Galizia, and L. Cabral. The Choreography Model for IRS-III. In *Hawaii International Conference on System Sciences (HICSS 2006)*, 2006.
8. A. Duke, M. Richardson, S. Watkins, and M. Roberts. Towards B2B Integration in Telecommunications with Semantic Web Services. In Gómez-Pérez and Euzenat [13], pages 710–724.
9. T. Erl, editor. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, 2005.
10. D. Fensel. *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag, 2001.
11. M. Flügge and K.-U. Schmidt. Using Semantic Web Services for ad hoc Collaboration in Virtual Teams. In *Berliner XML Tage*, pages 187–198, 2004.
12. M. Flügge and D. Tourtchaninova. Ontology-derived Activity Components for Composing Travel Web Services. In *Berliner XML Tage*, pages 133–150, 2004.
13. A. Gómez-Pérez and J. Euzenat, editors. *The Semantic Web: Research and Applications, Second European Semantic Web Conference, ESWC 2005*, volume 3532 of *LNCS*. Springer-Verlag, 2005.
14. P. Haase, S. Agarwal, and Y. Sure. Service-Oriented Semantic Peer-to-Peer Systems. In C. Bussler et al., editor, *Workshop Web Information Systems Engineering*, volume 3307 of *LNCS*, pages 46–57. Springer-Verlag, 2004.
15. J. Miller, J. Arnold, J. Cardoso, A. Sheth, and K. Kochut. Quality of Service for Workflows and Web Service Processes. *Journal of Web Semantics*, July/August, 2004.

16. A.P. Sheth, J.A. Miller, Z. Wu, K. Verma, and K. Gomadam. The METEOR-S Approach for Configuring and Executing Dynamic Web Processes. Technical report, June 2005.
17. D. Krafzig, K. Banke, and D. Slama, editors. *Enterprise SOA: Service Oriented Architecture Best Practices*. Prentice Hall PTR, 2004.
18. P.W. Lord, S. Bechhofer, M. D. Wilkinson, G. Schiltz, D. Gessler, D. Hull, C.A. Goble, and L. Stein. Applying Semantic Web Services to Bioinformatics: Experiences Gained, Lessons Learnt. In McIlraith et al. [19], pages 350–364.
19. S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors. *The Semantic Web - ISWC 2004: Third International Semantic Web Conference*, volume 3298 of *LNCS*. Springer-Verlag, 2004.
20. S.A. McIlraith, T. Cao Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.
21. D. Oberle. *Semantic Management of Middleware.* Springer-Verlag, February 2006.
22. D. Oberle, S. Lamparter, S. Grimm, D. Vrandecic, S. Staab, and A. Gangemi. Towards Ontologies for Formalizing Modularization and Communication in Large Software Systems. *Journal of Applied Ontology*, 2006.
23. A. Polleres, I. Toma, and D. Fensel. Modeling Services for the Semantic Grid. In C. Goble, C. Kesselman, and Y. Sure, editors, *Semantic Grid: The Convergence of Technologies*, number 05271 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.
24. S. Potter and J.S. Aitken. A Semantic Service Environment: A Case Study in Bioinformatics. In Gómez-Pérez and Euzenat [13], pages 694–709.
25. C. Preist, J. Esplugas Cuadrado, S. Battle, S. Grimm, and S.K. Williams. Automated Business-to-Business Integration of a Logistics Supply Chain Using Semantic Web Services Technology. In Y. Gil, E. Motta, V.R. Benjamins, and M.A. Musen, editors, *International Semantic Web Conference*, volume 3729 of *LNCS*, pages 987–1001. Springer-Verlag, 2005.
26. N. Sadeh, F. Gandon, and O. Buyng Kwon. Ambient Intelligence: The MyCampus Experience. In T. Vasilakos and W. Pedrycz, editors, *Ambient Intelligence and Pervasive Computing*. ArTech House, 2006.
27. D. Sell, L. Cabral, E. Motta, J. Domingue, and F. Hakimpour. A Semantic Web based Architecture for Analytical Tools. In *7th International IEEE Conference on E Commerce Technology (IEEE CEC 2005)*, 2005.
28. S. Staab and H. Stuckenschmidt, editors. *Semantic Web and Peer-to-Peer*. Springer-Verlag, November 2005.
29. S. Staab and R. Studer. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer-Verlag, 2004.
30. K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated Discovery, Interaction and Composition of Semantic Web Services. *Journal of Web Semantics*, 1(1):27–46, September 2003.
31. V. Tanasescu, A. Gugliotta, J. Domingue, L. Gutiérrez Villarías, R. Davies, M. Rowlatt, and M. Richardson. A Semantic Web GIS based Emergency Management System. In *Workshop on Semantic Web for eGovernment Held in conjunction with ESWC 2006*, 2006.
32. J.M. Bradshaw, R. Jeffers, M. Johnson, A. Tate, J. Dalton, A. Uszok, and S. Aitken. KAoS Policy Management for Semantic Web Services. *IEEE Intelligent Systems*, 19, July/August 2004.
33. M. Zaremba, M. Moran, and T. Haselwanter. Applying Semantic Web Services to Virtual Travel Agency Case Study, Poster Presentation. In Y. Sure and J. Domingue, editors, *ESWC*, volume 4011 of *LNCS*, pages 782–796. Springer-Verlag, 2004.

Part I

Web Services Technology

1 Towards Service-Oriented Architectures

Stefan Fischer and Christian Werner

Institute for Telematics, University of Lübeck
{fischer,werner}@itm.uni-luebeck.de

Summary. This chapter is meant as a motivation of why and how Web Services have evolved. Starting from the increasing need for integration of IT solutions, we argue that Web Services have something to offer, especially for the important fields of Business-to-Business (B2B) and Enterprise Application Integration (EAI). However, this is only the beginning of a new road, leading to the radically new software technology of Service-Oriented Architectures (SOA).

1.1 Integration: The New Challenge

This book is about Semantic Web Services, and before we can talk about the new and fascinating "semantics" part, it will be helpful to consider the foundations, namely to look at the Web Service technology itself. Web Services themselves are a relatively new phenomenon and have been under development for only slightly more than five years. During this time, they have gained a lot of attention and have also already gone through their hype phase. Meanwhile, they are accepted as one of the most important technologies when talking about *application integration*. In this context, they have been brought together with many other buzzwords that have been coming up in recent years.

1.1.1 The Need for Integration

First, Web Services have been considered as a new kind of *middleware*, taking their place between application and network. Here, they are in competition with other similar approaches such as CORBA, Java RMI, OSF DCE, etc. One can very well argue that Web Services have an excellent chance to become the dominating middleware, due to their extensive support of Internet technologies – one of the most important and relatively rarely mentioned being the use of URIs/URLs as addressing scheme – and the massive support by IT industry, resulting, for instance, in an excellent tool chain support throughout the software life cycle.

Second, Web Services have been chosen as one of the base technologies in *grid computing*, another major recent buzzword. A computing or storage grid works just like a water or electrical grid – just press a button and the grid delivers as much water or power as you need. In IT terms, you just plug in your terminal and get as much storage capacity or computing power as you need. The resources will be provided by the grid, which consists of some cooperation software and lots of more or less powerful computers. As a user, you do not see the computers, you just see the grid (or the plug) and its services. And grid services are provided as Web Services, so that is where they come into the game. Whenever your grid application makes use of one of the grid's services, it calls a Web Service.

Third and final example, Web Services are the basic component in most *SOA* approaches. SOA means *Service-Oriented Architecture*, and it is the latest hype in enterprise application software architecture design. SOA will most likely become one of the most important technologies within the next few years. Due to its importance, we will come back to it at the end of this chapter.

All these technologies are related to integration. Why is integration obviously so important that it triggers the development of so many new buzzwords, hypes, and serious new technologies?

This has to do with the famous real-world phenomenon of *globalisation*. Today, goods and services are traded and provided worldwide. Companies are no longer restricted to their home base, but often produce their products in different countries. There, they cooperate with other companies they might not have heard of a few weeks before. Or they might even buy other companies which fit into their portfolio or provide a certain service that the buying company urgently needs.

In order to survive in a globalised world, the IT infrastructure of such companies needs to be adjusted to the new requirements. This basically means two things. First, integration has to take place on an internal level. It will be necessary that applications in different domains can work together, based on the same stock of data. This is not as simple as it might sound: just consider as an example the merger of Daimler and Chrysler and the need to integrate these two completely different IT worlds. Second, integration has to take place on an external level. Applications of different business partners have to cooperate, e.g. in a selling–buying process. This is a major challenge based on heterogeneous technologies, but a major success factor for a globally operating company.

How can integration be achieved? Let us first have a look at an obvious candidate – the Internet and especially the World Wide Web.

1.1.2 B2C: Great New World?

One could well argue that the Web offers everything you need in order to conduct business. In fact, probably millions of web-based applications are in use today, so a lot of business is going on already. Looking closely at these applications, one will realise that they all have one very specific property: they are *interactive* applications, which means that there is a always a human on one end of the line (*human–machine interaction*). Take, e.g., all the well-known ticket-booking applications for flights, railway travel, etc. They are very well suited for interaction between a user and the application, providing a usually very nice and stylish user interface. Most of them are the so-called *business-to-consumer (B2C)* applications, i.e. individuals use them to conduct all kinds of businesses on the web.

However, when we talk about globalisation and integration, we usually mean something else: it is the companies that need to cooperate. Usually, we then talk about B2B applications. So, can we use the same technologies to organise this cooperation, i.e. to implement B2B applications?

The answer is clearly no, and the reason for this is the already mentioned interactivity of B2C applications. B2B applications are inherently *non-interactive*; instead, nearly every transaction is expected to be executed automatically. Only then, such applications really make sense, because of the gain of speed and efficiency. What therefore is needed is not human–machine, but *machine–machine interaction*. Why does it not make sense to use the existing interface of web applications for B2B applications?

B2C applications use a standardised page description language to create these interfaces the name of which is *HTML*. HTML does a great job: it is simple, flexible, robust and very expressive, especially with its partner *Cascading Style Sheets (CSS)*, but it does just that: it describes page layouts. This is great for user interfaces, but it is not good for automatised interactions between applications. Applications need to exchange clean data, just describing the objects of the application domain which need to be exchanged. Let us look at an example.

When you book a flight on the Lufthansa or British Airways website, you will typically be presented a list of available flights. For a human user, it is absolutely no problem to understand the content of the page, since it has been nicely rendered by the web browser. The web browser got, from the Lufthansa web server, something like

```
<tr><td>FRA</td><td>SFR</td><td>10:00</td><td>12:00</td></tr>
```

The browser need not understand the application-specific semantics of this code, it just needs to understand the meaning of *tags* such as `<tr>` – in this case indicating a new line in the table displaying all flights. For an automated application-specific processing (i.e. not simply layout oriented), however, this is not sufficient, since there is absolutely no information available on what the application domain of this code is. We as humans can tell that most likely this describes a flight from Frankfurt to San Francisco which leaves at 10 in the morning and arrives at noon, but the “dumb” computer can not.

To summarise, HTML is not enough for B2B applications. If we want to make use of applications which are available on the Internet today in B2B contexts, we need to provide different interfaces.

1.1.3 B2B and EAI: Today’s Solutions

Before we see how the new integration solution works, let us have a look at some earlier approaches that have been developed in order to create interoperable applications in the business world. We already called the interaction between companies B2B. There was (and still is) a second big movement that covered the integration question on an inter-company level. It is called *Enterprise Application Integration (EAI)*, but in essence, from a technical point of view, it is the same as B2B.

The name of these solutions has already been mentioned: middleware. The purpose of middleware is usually twofold. First, it is meant to abstract away from the details and the complexity of network programming which includes bit-wise encoding of messages and their transfer to specific destinations. Second, it provides a uniform way of describing interfaces of objects relevant in a certain application. Based on these interfaces, services provided by one object can be used by other objects using the interface's description.

First solutions such as Sun's Remote Procedure Call (RPC) were simple and straight-forward, but today's dominant technologies such as the Common Object Request Broker Architecture (CORBA) developed by the OMG or Sun's Enterprise Java Beans provide a full-featured framework for creating powerful distributed applications. And even more important, this framework can be very well used to integrate existing applications by providing them with a new wrap-around interface which can then be used by other applications.

This sounds good, but it did not really work out. Mainly, three types of reasons can be given:

1. Complexity
   Most of today's middleware approaches employ complex communication protocols which make it rather difficult to implement them. Consequently, there are typically rather few implementations to choose from. In addition, they are often incompatible, as has been reported for a number of CORBA implementations. This is critical for any kind of integration approaches for applications created by different companies.
2. Lack of standardisation
   Whenever a new middleware was invented, most of the important underlying technologies were invented too. This includes, for instance, all the communication protocols between application components, all the pre-defined services (such as name service or trader service), and also basic features such as the scheme for addressing objects of the system. It is obviously hard to convince others who have been using a certain scheme all the time to use a new (just invented) one when nobody else is doing that.
3. Political reasons
   Information technology's short history has already shown that technologies invented by one company are rarely adopted by their competitors. A typical example is the programming language Java which has been developed by Sun while, in turn, Microsoft released a competing language realising similar concepts. Another example is middleware: Microsofts DCOM has never been accepted or supported by Sun, and Suns Java 2 Enterprise Edition (J2EE, which includes Enterprise Java Beans) has never been supported by Microsoft.

   A typical example is the programming language Java which has been developed by Sun and never been liked by Microsoft. So they invented their own Java but gave it a different name. The same is true for middleware: Microsoft's

> DCOM has never been accepted or supported by Sun, and Sun's Java 2 Enterprise Edition (J2EE, which includes Enterprise Java Beans) has never been supported by Microsoft.

As a result, instead of integration, we got many IT islands in the 1990s, and there was no simple way to let them cooperate. Shortly before the year 2000, however, the pressure by IT industry's customers became big enough to make it think about a solution to these problems. In the next section, we will see how Web Services addresses them. The next chapter will then give a more technical, in-depth introduction into the most important components of Web Services.

1.2 Web Services as a New Solution

As one may have guessed by now, the Web Service technology tackles the major problems that come with other technologies, as mentioned above. And this is certainly by design and not by accident. Here is what one of the standard documents says what a Web Service is: "A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web Service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialisation in conjunction with other Web-related standards" [W3C Web Services Architecture].[1]

From this definition, two of the three main questions are already answered: Web Services use standardised and open web technology wherever possible, from URIs as the basic addressing scheme over XML as the basic description language to the use of Internet protocols for message transport. The next few paragraphs describe why these are advantages.

1.2.1 URI as Addressing Scheme

The concept of Universal Resource Identifiers and their more practicable subset, the Uniform Resource Locator, was developed for the World Wide Web. It is used to uniquely identify any single resource on the web, especially documents and applications. The concept is very well introduced and well under stood. In addition, there is massive infrastructure available that supports all kinds of operations on URIs, namely their mapping on more concrete addresses as used by computers to find the resource identified by a URI. The most important part of this infrastructure is the *Domain Name System (DNS)* which maps host names to IP addresses. Another important role is played by the web server which maps the rest of the URL to a local path in the file system.

Why is this so important? Obviously, this infrastructure can simply be used for Web Services. When a service user calls a specific Web Service, the URI of this service is known. Then, the features of DNS and the existing web servers can easily be used to find this service. There is absolutely no need to develop something like a new name service: everything is already there.

[1] `http://www.w3.org/TR/ws-arch/`

1.2.2 XML as the New Lingua Franca

XML as such is just another data description language, and this is exactly what is needed: a simple and standardised language that can be used to describe data structures. It is not that such languages have not existed before: just think of ASN.1, the data description language coming from the OSI world. As with many things, XML came just at the right point in time, and it provided a number of features that quickly made it popular with a huge community:

- XML is simple. It is very easy to define a data structure in XML, due to its intuitive hierarchical structure.
- XML is ASCII. Every XML data structure is human-readable. This might result in performance problems, but it brings a lot of advantages, for instance when debugging a service.
- XML is self-describing. An XML data structure contains both a description of its structure and the content itself.
- It has been standardised by the World Wide Web Consortium (W3C), which is the most important standardisation body in the context of World Wide Web protocols and languages.

In Web Services, everything is based on XML. As can be seen from the above definition, it is not only the definition of data structures, but all message exchanges and also the service descriptions are based on this new universal data description language.

1.2.3 Exchanging XML Messages

In order for two distributed application components to communicate with each other, they need to exchange messages. As we already know, the messages are described in XML, but how does the exchange protocol look like? How are messages encoded? How are they finally transported over the network?

Many people say that these are the core questions when designing a middleware, so one can argue well that the solutions in this field belong to the core of Web Service technology. The protocol that is used to encode XML messages is called *Simple Object Access Protocol (SOAP)*, or, more correctly as of today, the *XML Protocol (XMLP)*. And it is really simple: it just defines a general pattern of how XML Web Service messages have to look like, it defines a few so-called *message exchange pattern* that can be used by Web Service partners, and it defines how XML type information can be encoded in such messages. It also gives some hints how such messages can be transported over the Internet: just use one of the existing application-level protocols such as the web protocol HTTP or the email protocol SMTP.

Why is it good to use such existing protocols? Obviously again, these are well-established technologies, and there is a mass of products available which can simply be used. Most importantly, one can use web or email servers in order to receive Web Service calls and forward them to their appropriate end points. Such servers are available everywhere, and every system administrator knows how to configure and administrate them. So, we have the same advantage as with DNS: the infrastructure is there and just waits to be used. And protocols such as HTTP and SMTP have been verified over and over again: they simply work, and interoperability questions simply do not arise anymore today.

1.2.4 Creating Services Based on XML Messages

How, after all, does a service user know how a message has to look like in order to use a provided service? Here, we have to mention the second most important Web Service technology, the *Web Service Description Language (WSDL)*. WSDL is defined in XML, and it is used to define how Web Service interfaces look like. This basically means that it describes how incoming and outgoing messages look like and where such services are available (in terms of a URI). Once such a definition exists and gets published, a service user can read it and then knows how to call a Web Service. Since everything is again in XML, the messages to be transferred can be immediately deduced from the service description. And on the service side, it is easy to decode incoming service messages: just use one of the many existing XML parsers and read the message into your service program. All in all, creating a Web Service usually only means copying an already existing object into a certain location – and that is it.

1.2.5 And Politics?

What is not obvious from the above definition is the political question. Why are Web Services much better accepted and supported within the IT industry than any other middleware technology?

One may well argue that Web Services have been invented by Microsoft. The above-mentioned SOAP has been brought up by Microsoft. And right from the beginning, Microsoft was interested in making this an open standard, which had so far not exactly been one of Microsoft's core strategies. As a result, other companies such as IBM and Hewlett-Packard and later on Sun Microsystems jumped onto the bandwaggon. With this support, Web Services on the one hand get a lot of publicity and on the other hundreds of developers started to create languages, protocols and, most importantly, tools. Today, from all middleware and integration technologies, Web Services get the best tool support along the life cycle. It has been the core or it has been fully integrated into today's most relevant enterprise application architectures, namely Microsoft's .Net and Sun's J2EE. Meanwhile, we see applications being created that consist of both .Net and J2EE (and other) components, so it obviously works. Still, a few things are missing.

1.2.6 What is Missing?

Since their creation in the late 1990s, Web Services have gained a lot of momentum. Many people are sure that this is the new integration technology. However, there are still a few things to do in order to make it real and have the Web Service technology universally accepted. From the authors' point of view, there are at least the following three points to mention:

1. Well-established directory services
   Directory services are needed in order to find available Web Services. Web Service providers publish their services in such a directory, and Web Service users look for services that best fit their needs. In the best case, applications know what they need and then automatically check directories for corresponding services (we will later in this book see how semantics help in this respect). Surely, this need has been openly visible from early on, and with the specification of *Universal Description, Discovery, and Integration (UDDI)*, there is a solution available. However, UDDI in its use as a global Web Service directory today does not have the best image. This is due to the fact that basically everybody is allowed to publish new service entries. Sounding good at first hand, this quickly results in lots of dummy entries and dangling pointers to no longer existing services. As a result, the quality of the global UDDI directory is not good, so that UDDI is not often used for serious applications (though on an enterprise-level, the UDDI technology is widely adopted, since here, the entries can be controlled). In order to create a really useful global service directory, some kind of quality management needs to be put into effect.
2. Security
   In Web Service technology's early days, security never has been a big issue, due to the need to first make Web Services really work. The typical way of talking about it was "Great, Web Services work over firewalls". This is true, because Web Services are typically transported using the HTTP protocol, and the HTTP port is usually open on a firewall, but certainly is a bad argument – which administrator likes the thought that all kinds of active codes can be transported into his systems? Today, many solutions around Web Service security exist, including something like SOAP proxies in order to allow security checks on incoming Web Service calls, XML encryption which allows confidential calls of Web Services, and XML signature for Web Service message authentication. Actually, there are so many security standards available right now that it is already too much. For making Web Services really happen, a small set of security standards has to be identified that needs to be supported by all serious Web Service users and providers. A first step has been done by the so-called WS-Security standard that provides such a basic set of services. It is now necessary to promote this approach more actively.
3. Interoperability
   It certainly sounds strange that interoperability is one of the major problems with Web Services, when we just said that Web Services are all about integration. It is true, there is a number of open standards which are easy to implement and ubiquitously available. The bad thing is there are already too many of these standards. We mentioned this above for the field of security, but this is also already true for the basic standards such as SOAP/XMLP, WSDL, or UDDI; in other words those technologies that need to be present in order to make Web Services run at all. Basically, there are two problems. First, some companies might tweak the standard just a little bit in order to make them work better with their own tools. Second, there are different versions of these basic standards. Unfor-

tunately, they are usually not interoperable. The newest WSDL standard, for instance, uses keywords which have not been available in earlier standards, and discards others. As a result, a WSDL 1.1 interpreter will not be able to decode a WSDL 2.0 description, making it impossible for the user to call this specific service.

There is already a solution for this problem, provided by the organisation *WS-Interoperability (WS-I)*. WS-I defines the so-called *profiles* which contain a set of standards. Whenever a company declares that its services are compatible with a certain WS-I profile, it guarantees that all the relevant standards are implemented in a standard-conforming way. As of today, WS-I has published the Basic Interoperability Profile. Companies which are really interested in global and automatised interaction with other companies will have to make their services compatible to these profiles.

And after all, this book is about Semantic Web Services. In the last section of this chapter, we will look, as promised, at the new idea of service-oriented architectures and explain how a formally described semantics may play a major role in making them real.

1.3 The Future: SOA

Web Services are a basic building block in the creation of SOA. These SOAs are expected to be the future architecture of enterprise applications. As can be told from the name, the idea is that future applications will be built upon services. This is, however, not the whole picture; service-oriented computing has already been the concept of CORBA and similar approaches. SOA go a step further and propose a completely new way of creating applications. In the SOA vision, they will no longer be programmed, but instead *composed* of loosely coupled components which will be imported from servers from all over the world. Required services will be dynamically – potentially during run time – searched and called when needed. Such an architecture is well suited to map the dynamic environment that enterprises are confronted with in today's globalised world. If, for instance, two companies form a new strategic alliance or just create a new customer–supplier relationship in the real world, the vision says that in the IT world this will simply mean abandoning a few services and selecting a few new ones.

The question certainly is, how realistic this vision is today. Above, we have already discussed a few obstacles such as unused security features, lack of interoperability, or missing high-quality directory services. However, even when all these are available, would you, as the Chief Information Officer, lay the fate of your company in the hands of some obscure services that you probably do not really know anything about?

The probably much more realistic scenario is an implementation of SOAs within the boundaries of an enterprise, i.e. as a new approach towards EAI. Here, the company has full control over all services and service offers and can thus make sure that all applications that the company relies on will really be operational.

Since the rest of this book is on semantics and Web Services, we consider it useful to provide a first hint at how these two big trends in Web Services – SOA and semantics – fit together.

We have mentioned several times now that Web Services is on integration, automatisation, and machine–machine interaction. In order to fully automate the communication between application components, the search for new Web Services also needs to go into this direction. In a perfect SOA world, an application component in need for a specific service describes this need in its problem domain and sends it to a directory service. This service will "understand" the need of the component and look for matching services. From the list of found matches, the component selects one service and automatically binds it to the running application.

Today, this is not possible, because a formal description of the functionality of a service is not available. This book shows what needs to be done in order to make the vision real.

2

Architecture and Standardisation of Web Services

Christian Werner and Stefan Fischer

Institute for Telematics, University of Lübeck
{werner, fischer}@itm.uni-luebeck.de

Summary. Since Web Services are complex artefacts that rely on sophisticated protocols and data formats, it is important to have effective strategies for dealing with this complexity. As a basic concept, the Web Service technologies are structured in a stack model. It is crucial for every Web Service developer to have this model in mind and to have a clear understanding how the single items work together. In this chapter, we will first give an overview of the Web Service technology stack. Then, we will step through this model and discuss the different core technologies in detail. This includes different variants of Web Service transport bindings, SOAP, WSDL and UDDI.

2.1 Web Services Technology Stack

The *W3C Web Service Architecture Working Group* has developed a model that describes how Web Services are generally structured, called the Web Service Technology Stack. However, in order not to limit the scope of Web Service technology, this model has been purposely designed on a very abstract level, i.e. without specifying technologies used for the implementation. Other W3C working groups are providing such technology specific bindings.

The current version of the *Web Service Architecture Document* has been released on 11 February 2004 and is publicly available at `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/`. Figure 2.1 depicts the basic structure of the Web Service technology stack.

The "Communications" block, which some authors also call "Wire Stack", is the basis for all other layers. It comprises generic transport mechanisms that can be used to send messages over the Internet. In terms of classical network architecture these technologies are located on the "Application Layer" (or "Layer 7") of the ISO/OSI Protocol Reference Model. Typical examples would be HTTP, SMTP, FTP, etc. All these protocols can be used with Web Services and each protocol does provide specific benefits and drawbacks. The Web Service Technology Stack does not determine which transport mechanism should be used, since the optimal choice may heavily depend on the specific use case. We will have a detailed look on this topic in the following section.

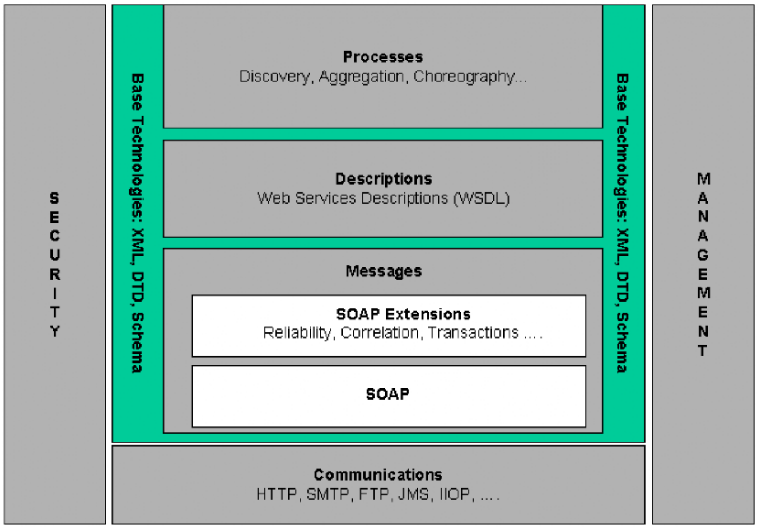
SECURITY
Base Technologies: XML, DTD, Schema
Processes
Discovery, Aggregation, Choreography...
Descriptions
Web Services Descriptions (WSDL)
Messages
SOAP Extensions
Reliability, Correlation, Transactions ....
SOAP
Base Technologies: XML, DTD, Schema
MANAGEMENT
Communications
HTTP, SMTP, FTP, JMS, IIOP, ...

Fig. 2.1. The web service technology stack (*taken from the W3C web service architecture document*)

The core technology of Web Services is located on the next higher layer. The "Messages" block provides basic functionalities for encapsulating network messages in a neutral way that is independent from a certain programming language or operating systems. The goal here is to find message representation that can be syntactically understood by humans as well as computers. Unfortunately, this does not mean that the meaning of a message can be understood by everybody. Technologies for semantic interoperability will be focussed in the following parts of this book. The key technology for achieving syntactical interoperability in the Web Service world is XML. In Fig. 2.1 this is indicated by the dark box around *Messages*, *Descriptions* and *Processes*.

The *Simple Object Access Protocol (SOAP)* is an XML language itself. It provides a composable framework for packaging and exchanging XML messages. In particular, it provides a platform- and application- independent message format. Furthermore, a number of extensions have been developed for SOAP. With these it is possible to provide additional protocol features, such as reliable message transport or support for transactions.

The layer above is named *Descriptions*. Here we find technologies that are used for describing Web Services in a formal way. Such a description is crucial because a Web Service consumer needs exact access parameters before the service can be used. This includes data about the service location as well as a specification of supported data types and operations. The most widely used solution here is the *Web Service Description Language (WSDL)*. It is important to note that WSDL does cover only the technical but not the semantical description of a service. So with WSDL it is possible to express that "This service has an operation `echo` which takes a string as parameter and returns another string." A human can guess from the operation's name that the two strings will be identical, but machines do need an additional semantic service description in order to discover services that provide suitable operations for a given problem.

The very top layer comprises *Processes*. One of the most important processes in the field of Web Services here is the discovery of a service. Web Services can be distributed all over the world and they might be used from all over the world too. How can we locate a Web Service that fits the user's needs? The most popular solution here is *Universal Description, Discovery and Integration (UDDI)*. This worldwide service registry can be visualised as a huge phone book. A service provider can publish a Web Service in this registry and if somebody is looking for a specific Web Service he or she can query the UDDI registry by specifying certain search criteria. Although there are mechanisms in UDDI for realising things like data replication, it is basically a centralised approach and therefore UDDI contradicts the concept of service distribution in some way. Anyhow, distributed service registries are harder to maintain and no solution for practical usage have been developed yet. A more active approach for building up service registries is called *Web Service Inspection (WS-Inspection)*. Here the service directory looks actively for new services and registers them.

Besides service discovery, there are more *Processes* that are important in a Web Service world. For instance, it is possible to combine a number of services in order to complete a certain task. Here we are talking about Web Service aggregation or Web Service composition.

In addition to the concepts and tasks that are located on the different layers in the Web Service Technology Stack, there are some issues that are relevant to all layers. The most important one here certainly is *Security*, shown as a column on the very right side of Fig. 2.1. In April 2002, Microsoft and IBM introduced the *Web Service Security (WS-Security)* specification. It provides a comprehensive security framework that is based on two other W3C standards as core components, namely *XML Encryption* and *XML Signature*.

A second area that is relevant to all layers of the Web Service technology stack is *Management*, shown in the very right side of Fig. 2.1. Since the Web Service technology primarily targets the domain of business applications, where the availability and reliability of a service might be crucial, it is very important that there are capable measures for monitoring and controlling the state of a Web Service. If we think of "pay per use" scenarios, it is also desirable that the service provides a certain *Quality of Service (QoS)*, e.g. by sending back query results within a given time interval. IBM addresses this issue in a framework called *Web Service Level Agreement (WSLA)*, which has been introduced in 2003. We will not discuss Web Service management here in detail, because it is still a very active area in research and therefore out of scope for this chapter.

In the remainder of this chapter, we will discuss the different layers of the Web Service Technology Stack, from bottom to top, more in detail.