4

Introduction to Semantically Enabled Service-oriented Architectures

Computer science is on the edge of an important new period of abstraction. A generation ago we learned to abstract from hardware and currently we are learning to abstract from software in terms of Service-Oriented Architectures (SOA). A SOA is essentially a collection of services. It is the service that counts for a customer and not the specific software or hardware components that are used to implement this service. It is a common expectation that SOAs will quickly become the leading software paradigm. However, we believe that these SOAs will not scale without significant mechanization of service discovery, service adapation, service negotiation, service composition, service invocation, and service monitoring, as well as data, protocol, and process mediation. We envisage the future of applied computer science in terms of SOAs which are empowered by adding semantics as a means to deal with heterogeneity and mechanization of service usage. This approach is called Semantically Enabled Service-oriented Architectures or SESA for short.

4.1 SESA Background

In this section we provide an overview of the fundamental elements of the SESA architecture, one which enables the execution of Semantic Web Services and resolves the fundamental challenges related to the open SOA environment. We expect that in the near future a service-oriented world will consist of an "uncountable" number of services. Their computation will involve services searching for other services based on functional and nonfunctional requirements, and then resolving any interoperability conflicts from those services selected. However, services will not be able to interact automatically and existing SOA solutions will not scale without significant mechanization of the service provisioning process. Hence, machine processable semantics is essential for the next generation of Service-Oriented Computing (SOC) to reach its full potential. In this chapter we define methods, algorithms, and tools forming a skeleton of SESA, introducing automation to the service provisioning process, including service discovery, negotiation, adaptation, composition, invocation, and monitoring, as well as service interaction requiring data and process mediation.

SOA outside a tightly controlled environment cannot succeed until semantic issues are addressed and critical tasks within the service provisioning process are automated leaving humans to focus on higher-level problems. While this chapter describes how these building blocks are consolidated into a coherent software architecture, which can be used as a blueprint for implementation, following chapters present the basic conceptual and technical building blocks required to set up the SESA infrastructure.

SESA has evolved from the collaborative efforts of three research/standardization groups: OASIS Semantic Execution Environment Technical Committee (SEE TC), Web Service Modeling Execution Environment (WSMX) Working Group, and NESSI. The aim of the OASIS SEE TC is to provide guidelines, justifications, and implementation directions for an execution environment for Semantic Web Services. The resulting infrastructure incorporates the application of semantics to service-oriented systems and provides mechanisms for consuming Semantic Web Services. WSMX is the reference implementation of Web Service Modeling Ontology (WSMO) and SEE. It is an execution environment for business application integration where enhanced Web Services are integrated for various business applications. The aim is to increase business processes automation in a very flexible manner while providing scalable integration solutions. The WSMX Working Group builds a prototypical execution infrastructure for Semantic Web Services based on the SOA paradigm of loosely coupled components. Finally SESA also relates to the work carried out by the NESSI initiative addressing semantic aspects of the NESSI platform. NESSI semantic technologies provide the semantics-based solutions for search and integration, which aim to enable progressive development of SESA. The NESSI Semantic Technology Working Group aims to use SESA as its roadmap defining the development of its semantic technologies.

4.2 Service Orientation

The design of enterprise information systems has gone through several changes in recent years. In order to respond to the requirements of enterprises for flexibility and dynamism, the traditional monolithic applications have become substituted by smaller composable units of functionality known as services. Information systems must then be retailored to fit this paradigm, with new applications developed as services, and legacy systems to be updated in order to expose service interfaces. The drive is towards a design of information systems which adopt paradigms of SOC together with the SOA implementation architecture and relevant Web service technologies.

4.2.1 Service-Oriented Computing

SOC is a new computing paradigm that utilizes services as the fundamental elements for the development of rapid, low-cost, and easily integrable enterprise applications [171, 173]. One of the main goals of SOC is to enable the development of networks of integrated and collaborative applications, regardless of both the platform on which

applications or services run (e.g., the operating system) and the programming languages used to develop them.

In this paradigm, services are autonomous, self-describing, and platform-independent computational entities, which provide a uniform and ubiquitous access to information for a wide range of computing devices (such as desktop computers, PDAs, cellular phones) and software programs across different platforms. Any piece of code and any application component deployed on a system can be reused and transformed into a network-available service. Service providers and service consumers remain loosely coupled as services, independent of the context in which they are used. Since these services are based on the service-orientation paradigm and distinguish characteristics of the service, they can be easily described, published, discovered, and dynamically assembled for developing distributed and interoperable systems.

The main goal of SOC is to facilitate the integration of newly built and legacy applications, which exist both within and across organizational boundaries. SOC must overcome and resolve heterogeneous conflicts due to different platforms, programming languages, security firewalls, etc. The basic idea behind this orientation is to allow applications which were developed independently (using different languages, technologies, or platforms) to be exposed as services and then interconnect them exploiting the Web infrastructure with respective standards such as HTTP, XML, SOAP, and WSDL and even some complex service orchestration standards like BPEL.

4.2.2 Service-Oriented Architecture

The service-oriented paradigm of computation can be abstractly implemented by the system architecture called SOA [24, 30]. The purpose of this architecture is to address the requirements of loosely coupled, standard-based, and protocol-independent distributed computing, mapping enterprise information systems isomorphically to the overall business process flow [174]. This attempt is considered to be the latest development of a long series of advancements in software engineering addressing the reuse of software components.

Historically, the first major step of this evolution was the development of the concept of *function*. Using functions, one decomposes a program into smaller subprograms and writing code is focused on the idea of the Application Programming interface (API). An API, practically, represents the contract to which a software component has to commit. The second major step was the development of the concept of *object*. An object is a basic building block which contains both data and functions within a single encapsulated unit. With the object-oriented paradigm, the notions of classes, inheritance, and polymorphism are introduced. In this way classes can be viewed as a lattice. The concept of *service* becomes the next evolutionary step introduced with the advent of SOC and its SOA implementation architecture.

Figure 4.1 shows the the *Web Services programming model*, which consists of three components: service consumers, service providers, and service registrars. Ignoring the detailed techniques for the connection of the three components, this model

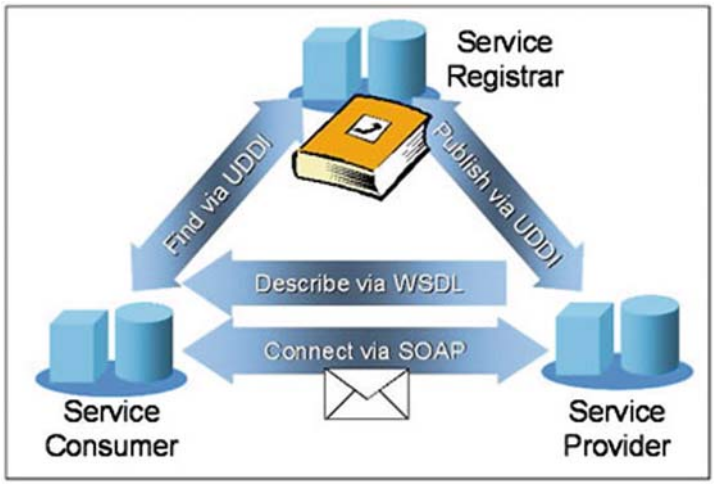

Figure 4.1. Web Services programming model

represents also the SOA basic model. A service registrar (also called service broker) acts as an intermediary between the provider and the consumer, so they are able to find each other. A service provider simply publishes the service. A service consumer tries to find services using the registrar; if it finds the desired service it can set up a contract with the provider in order to consume such a service and thus to do business.

The fundamental logic view of services in SOA is based on the division of service description (called usually interface) and service implementation [174]. Service interface defines the identity of a service and its invocation logistics. Service implementation implements the work that the service is designated to do. Based on this division, service providers and services consumers are loosely coupled. Furthermore, the services can be significantly reused and adapted according to certain requirements. Because service interfaces are platform-independent and implementation is transparent for the service consumers, a client from any communication device using any computational platform, operating system, and any programming language should be capable of using the service. The two facets of the service are distinct; they are designed and maintained as distinct items, though their existence is highly interrelated.

Based on the service autonomy and the clean separation of service interfaces from internal implementation, SOA provides a more flexible architecture that unifies business processes by modularizing large applications into services. Furthermore, enterprise-wide or even cross-enterprise applications can be realized by means of services development, integration, and adaptation. Some SOA distinguished requirements (or rather advantages) have been analyzed as follows [195]:

- **Loose coupling.** Interacting services are loosely coupled by nature. They run on different platforms, are implemented independently, and have different owners. The model has to consider the loose coupling of services with respect to one another.

- **Implementation neutrality.** The interface should only matter, not the implementation. Services are defined independently of their implementation and should behave neutrally with respect to it.
- **Flexible configuration.** Services are invoked dynamically after the discovery process through the service requester. That is, the binding of a service provider to the requester occurs at run time at the final phase.
- **Long lifetime.** Components/services should exist long enough to be discovered, to be relied upon, and to engender trust in their behavior.
- **Granularity.** The granularity of a service defines the complexity and number of functionalities defined by an individual service and is thus part of the service model. An appropriate balance between coarse-grained and fine-grained services depends on the way services are modeled. For too fine grained services, problems arise when there are frequent and rapid changes.
- **Teams.** Computation in open systems should be conceptualized as business partners working as a team. Therefore, a team of cooperating autonomous components/services is a better modeling unit, instead of framing computations centrally.

Besides the basic SOA model shown in Fig. 4.1, there are some extension works towards SOA which depict more concepts than service registration, discovery, and invocation. The extended SOA (xSOA) accounts for SOA deficiencies in such areas as management, security, service choreography and orchestration, and service transaction management and coordination [170]. The xSOA is an attempt to streamline, group together, and logically structure the functional requirements of complex applications that make use of the SOC paradigm.

4.2.3 SOA Implementations

Basically, Web Services seem to be becoming the preferred implementation technology for realizing the SOA promise of maximum service sharing, reuse, and interoperability. From Fig. 4.1, the Web service programming model is a typical SOA implementation. Defined by W3C, it is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML-based messages conveyed by Internet protocols. Interactions between Web Services typically occur as Simple Object Access Protocol (SOAP) calls carrying XML data content. Interface descriptions of the Web Services are expressed using Web Services Definition Language (WSDL). The Universal Description, Discovery, and Integration (UDDI) standard defines a protocol for directory services that contain Web service descriptions. UDDI enables Web service clients to locate candidate services and discover their details. Service clients and service providers utilize these standards to perform SOA's basic operations. Service aggregators may use the Business Process Execution Language for Web Services (BPEL4WS) to create new Web Services by defining corresponding compositions of the interfaces and internal processes of existing services.

Web Services technology is simply a widespread accepted instantiation of SOC providing a platform on which it is possible to develop applications taking advantage of the already existing Internet infrastructure. This does not mean that the set of technologies we present here are the only ones which make it possible to realize SOC and implement SOA. Some other more conventional programming languages or middleware platforms may be adopted as well, such as, for instance, established middleware technologies like J2EE, CORBA, and IBM's WebSphere MQ, and can now also participate in a SOA, using new features that work with WSDL.

The broad use of Enterprise Application Integration (EAI) middleware supports a variety of hub-and-spoke integration patterns [172]. EAI comprises message acceptance, transformation, translation, routing, message delivery, and business process management. All of these can be used to develop services and then it also becomes service-orientated architecture. The Enterprise Service Bus (ESB) is an open, standards-based message bus designed to enable the implementation, deployment, and management of SOA-based solutions with a focus on assembling, deploying, and managing distributed SOAs. The ESB provides the distributed processing, standards-based integration, and enterprise-class backbone required by the extended enterprise [112].

4.3 Execution Environment for Semantic Web Services

With the underpinning computation approach SOC, SOA is one of the most promising software engineering trends for future distributed systems. Pushed by major industry players and supported by many standardization efforts, Web Services are a prominent implementation of the service-oriented paradigm. They promise to foster reuse and to ease the implementation of loosely coupled distributed applications.

Although the idea of SOA targets the need for integration that is more adaptive to changes in business requirements, existing SOA solutions will prove difficult to scale without a proper degree of automation. While today's service technologies around WSDL, SOAP, UDDI, and BPEL have certainly brought a new potential to SOA, they only provide a partial solution to interoperability, mainly by means of unified technological environments. Where content and process level interoperability is to be solved, ad hoc solutions are often hard-wired in manual configuration of services or workflows, while at the same time they are hindered by dependence on XML-only descriptions. Although flexible and extensible, XML can only define the structure and syntax of data. Without machine-understandable semantics, services must be located and bound to service requesters at design time, which in turn limits possibilities for automation. In order to address these drawbacks, the extension of SOA with semantics offers a scalable integration, more adaptive to changes that might occur over a software system's lifetime. Semantics for SOA allows the definition of semantically rich and formal service models where semantics can be used to describe both services offered and capabilities required by potential consumers of those services. Also the data to be exchanged between business partners can be semantically described in an unambiguous manner in terms of ontologies. By means

of logical reasoning, semantic SOA thus promotes a total or partial automation of service discovery, mediation, composition, and invocation. Semantic SOA does not, however, mean replacing existing integration technologies. The goal is to build a new layer on the top of the existing service stack while at the same time adopting existing industry standards and technologies being used within existing enterprise infrastructures.

4.3.1 Challenges for the Semantic Web Service Execution Environment

Talking about technology challenges in the semantic domain, we consider semantics as an enabling technology to allow integration and interoperability of the heterogenous systems. What makes the integration based on semantics a challenge is twofold: first, the representation of information and the information itself used to be bound tightly together; and, second, that information frequently lacks any context. The programmer often thinks not of the data itself but rather of the structure of the data such as schemas, data types, relational database constructs, etc. These structures do not relate directly to the information, but rather to the assumption of what the data should look like. In tightly coupled architectures, data structures are absolutely necessary; since they provide for systems a way of coping with the information they are being fed. But this assumption does not hold for distributed systems, such as the SESA platform aimed to be.

In the upcoming years, semantics will be used more and more often as an enabling technology for new ways of assembling software on the fly in order to create ad hoc systems. Computer science is entering into a new phase, where semantics starts to play a major role. The previous generation of distributed systems was based on abstracting from hardware, but the emerging generation is already based on abstracting from software. In a world of distributed computing, it is the service that counts for a customer and not the software or hardware components that implement the service. However, current technological platforms are still restricted in their application context to in-house solutions.

Future enterprise systems will not scale without properly incorporating principles that made the Web scale to a worldwide communication infrastructure. There is a strong need for significant mechanization of service discovery, negotiation, adaptation, composition, invocation, and monitoring as well as service interaction requiring data, protocol, and process mediation; as well as a balanced integration of services provided by human and machines. All of these technologies can be only fully automated if the semantics is considered as the core enabling technology.

In a semantics-enabled world, the coordination between systems is executed through the use of well (semantically) described services, meaning discovered and selected on the basis of requirements, then orchestrated and adapted or integrated. Solving these problems is a major prerequisite to address technology challenges where the Web of services interconnects billions of entities without the need of using hard-coded adapters between these systems (as the current Web does for information sources) effectively that enable context awareness and discovery, advertising, personalization, and dynamic composition of services.

4.4 Governing Principles

We have identified a number of underlying principles which govern the design of the architecture, its middleware, as well as modeling of business services. These principles reflect fundamental aspects for a service-oriented and distributed environment which all promote intelligent and seamless integration and provisioning of business services. These principles include the following:

- **Service-oriented principle** represents a distinct approach for analysis, design, and implementation which further introduces particular principles that govern aspects of communication, architecture, and processing logic. This includes service reusability, loose coupling, abstraction, composability, autonomy, and discoverability.
- **Semantic principle** allows a rich and formal description of information and behavioral models enabling automation of certain tasks by means of logical reasoning. Combined with the service-oriented principle, semantics allows one to define scalable, semantically rich and formal service models and ontologies allowing one to promote total or partial automation of tasks such as service discovery, contracting, negotiation, mediation, composition, invocation, etc.
- **Problem-solving principle** reflects problem-solving methods as one of the fundamental concepts of artificial intelligence. It underpins the ultimate goal of the architecture which lies in so-called *goal-based discovery and invocation of services*. Users (service requesters) describe requests as goals semantically and independently of services, while architecture solves those goals by means of logical reasoning over descriptions of goals and services. Ultimately, users do not need to be aware of processing logic but only need to care about the result and its desired quality.
- **Distributed principle** allows one to aggregate the power of several computing entities to collaboratively run a task in a transparent and coherent way, so that from a service requester's perspective they can appear as a single and centralized system. This principle allows one to execute a process across a number of components/services over the network, which in turn can promote scalability and quality of the process.

4.5 SESA Vision – Global View

The global view of the architecture, depicted in Fig. 4.2, comprises several layers, namely, (1) stakeholders forming several groups of users of the architecture, (2) problem-solving layer building the environment for stakeholder access to the architecture, (3) service requesters as client systems of the architecture, (4) middleware providing the intelligence for the integration and interoperation of business services, and (5) service providers exposing the functionality of back-end systems as business services. In this section we describe these layers and define service types in the architecture and underlying concepts and technology we use for architecture implementation.

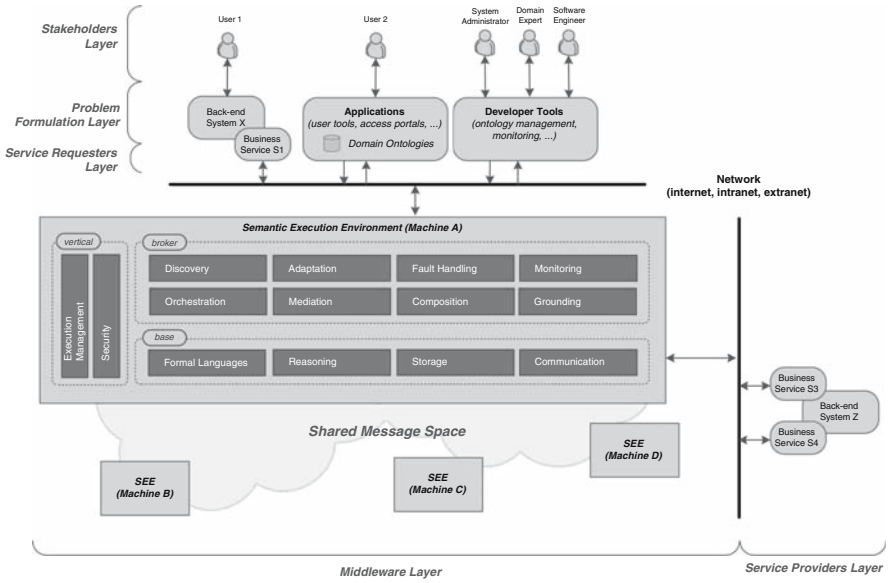

Figure 4.2. Global view of Semantically Enabled Service-oriented Architecture (SESA)

Realizing SESA principles and providing a platform incorporating them is the major necessity to implement the vision of Web Services. There are four types of business services of an infrastructure which are a must for Web Services to deliver their promises:

1. **The stakeholders layer**, which consists of ontologies, applications (e.g., e-tourism, e-government), and developer tools (GUI tools such as those for engineering ontology/Web service descriptions; generic developer tools such as language APIs, parsers/serializers, converters, etc.).
2. **The broker layer**, which consists of discovery, adaptation (including selection and negotiation), composition (Web service composition techniques such as planning), choreography, mediation (ontology mediation – techniques for combining ontologies and for overcoming differences between ontologies – and process mediation – overcoming differences in message ordering, etc.), grounding, fault handling (transactionality, compensation, etc.), and monitoring.
3. **The base layer**, which provides the exchange formalism used by the architecture, i.e., formal languages (static ontology and behavioral, i.e., capability/choreography/orchestration, languages, connection between higher-level descriptions, e.g., WSML), reasoning (techniques for reasoning over formal descriptions; logic programming, description logics, first-order logics, behavioral languages, etc.), and storage and communication.
4. **Vertical services** such as execution management and security (authentication/authorization, encryption, trust/certification).

4.5.1 Stakeholders Layer

Stakeholders form the group of various users which use the functionality of the architecture for various purposes. Two basic groups of stakeholders are identified: users and engineers. Users form the group of those stakeholders for which the architecture provides end-user functionality through specialized applications. For example, users can perform electronic exchange of information to acquire or provide products or services, to place or receive orders, or to perform financial transactions. In general, the goal is to allow users to interact with business processes on-line while at the same time reducing their physical interactions with back-office operations. On the other hand, the group of engineers form those stakeholders who perform development and administrative tasks in the architecture. These tasks support the whole SOA life cycle, including service modeling, creation (assembling), deployment (publishing), and management. Different types of engineers could be involved in this process, ranging from domain experts (modeling, creation), to system administrators (deployment, management), to software engineers.

4.5.2 Problem-Solving Layer

The problem-solving layer contains applications and tools which support stakeholders during formulation of problems/requests and generates descriptions of such requests in the form of user goals.Through the problem-solving layer, the user will be able to solve his/her problems, i.e., formulate a problem, interact with the architecture during processing, and get his/her desired results. This layer contains back-end systems which directly interface the middleware within business processes, specialized applications built for specific purpose in a particular domain, which also provide specific domain ontologies, and developer tools providing functionality for development and administrative tasks within the architecture. Developer tools provide a specific functionality for engineers, i.e., domain experts, system administrators, and software engineers. The functionality of developer tools covers the whole SOA life cycle, including service modeling, creation (assembling), deployment (publishing), and management. The vision is to have an integrated development environment (IDE) for management of the architecture. It should aid the developers through the development process, including engineering of semantic descriptions (services, goals, and ontologies), creation of mediation mappings, and interfacing with architecture middleware and external systems. By combining this functionality, a developer will be allowed to create and manage ontologies, Web Services, goals and mediators, create ontology-to-ontology mediation mappings, and deploy these mappings to the middleware. Applications provide a specialized functionality for architecture end-users. They provide specialized domain-specific ontologies, user interfaces, and application functionality through which stakeholders interact with the architecture and its processes. Through specialized applications in a particular application settings, the technology and its functionality are validated and evaluated.

4.5.3 Service Requesters Layer

Service requesters act as client systems in a client–server settings of the architecture. They are represented by goals created through problem/request formulation by which they describe requests as well as interfaces through which they wish to perform conversation with potential services. Service requesters are present for all applications and tools from the problem-solving layer and are bound to a specific service semantics specification.

4.5.4 Middleware Layer

Middleware is the core of the architecture providing the main intelligence for the integration and interoperation of business services. For purposes of the SESA, we call this middleware "semantic execution environment." The architecture defines the necessary conceptual functionality that is imposed on the architecture through the underlying principles defined in Sect. 4.2. Each such functionality could be realized (totally or partially) by a number of so-called middleware services (see Sect. 4.5.6) We further distinguish this functionality in the following layers: base layer, broker layer, and vertical layer.

The vertical layer defines the middleware framework functionality that is used across the broker and base Layers but which remains invisible to them. This technique is best understood through the so-called "Hollywood principle" that basically means "Don't call us, we'll call you." In this respect, framework functionality always consumes the functionality of broker and base layers, coordinating and managing overall execution processes in the middleware. For example, discovery or data mediation is not aware of the overall coordination and distributed mechanism of the execution management.

- **Execution management** defines the control of various execution scenarios (called execution semantics) and handles distributed execution of middleware services.
- **Security** defines a secure communication, i.e., authentication, authorization, confidentiality, data encryption, traceability, or nonrepudiation support applied within execution scenarios in the architecture.

The broker layer defines the functionality which is directly required for a goal-based invocation of Semantic Web Services. The broker layer includes:

- **Discovery**, which defines tasks for identifying and locating business services which can achieve a requester's goal.
- **Choreography**, which defines formal specifications of interactions and processes between the service providers and the client.
- **Monitoring**, which defines a monitoring of the execution of end-point services. This monitoring may be used for gathering information on invoked services, e.g., quality of service (QoS) related or for identifying faults during execution.
- **Fault handling**, which defines the handling of faults occurring within the execution of end-point Web Services.

- **Adaptation**, which defines an adaptation within a particular execution scenario according to the user's preferences (e.g., service selection, negotiation, contracting).
- **Mediation**, which defines interoperability at the functional, data, and process levels.
- **Composition**, which defines a composition of services into an executable workflow (business process). It also includes orchestration, which defines the execution of a composite process (business process) together with a conversation between a service requester and a service provider within that process.
- **Grounding**, which defines a link between a semantic level and a nonsemantic level (e.g., WSDL) used for service invocation.

The base layer defines functionality that is not directly required in a goal-based invocation of business services; however, they are required by the broker layer for successful operation. The base layer includes:

- **Formal languages**, which define syntactical operations (e.g., parsing), with semantic languages used for semantic description of services, goals, and ontologies.
- **Reasoning**, which defines reasoning functionality over semantic descriptions.
- **Storage and communication**, which defines persistence mechanism for various elements (e.g., services, ontologies) as well as inbound and outbound communication of the middleware.

The SESA middleware can operate in a distributed manner when a number of middleware systems connected using a shared message space operate within a network of middleware systems and empowering this way a scalability of integration processes. The SESA consists of several decoupled services allowing independent refinement of these services – each of them can have its own structure without hindering the overall SESA. Following the SOA design principles, the SESA separates concerns of individual middleware services, thereby separating service descriptions and their interfaces from the implementation. This adds flexibility and scalability for upgrading or replacing the implementation of middleware services which adhere to required interfaces.

4.5.5 Service Providers Layer

Service providers represent various back-end systems. Unlike back-end systems in the service requesters layer which act as clients in a client–server setting of the architecture, the back-end systems in the service providers layer act as servers which provide certain functionality for certain purposes exposed as a business service to the architecture. Depending on the particular architecture deployment and integration scenarios, the back-end systems could originate from one organization (one service provider) or multiple organizations (more service providers) interconnected over the network (Internet, intranet, or extranet). The architecture thus can serve various requirements for business-to-business (B2B) integration, EAI, or

application-to-application (A2A) integration. In all cases, functionality of back-end systems is exposed as semantically described business services.

4.5.6 Services in SESA

While some of the SESA functionality is provided as services, the rest remain as the entities required to let the overall system function – they are not services in terms of a SOA. While the middleware and service requesters layers build SESA in terms of services (with some exceptions), the problem-solving layer adds the set of tools and entities which makes SESA a fully fledged semantic SOA.

The core aspect of SOA is the service. In this respect, we distinguish two types of services in SESA, namely, middleware services and business services:

1. **Middleware services** are necessary to enable particular functionality of the architecture – they are the main facilitators for integration, search, and mediation of business services.
2. **Business services** are exposed by service providers, their back-end systems which are external to SESA. Business services are subject of integration and interoperation within the architecture (facilitated by the middleware services) and usually provide a certain value for architecture stakeholders. Such services are in SESA semantically described conforming to a specific semantic service model.

In this respect, the SESA defines *the scope of particular middleware services* in terms of the functionality they should provide. In addition, the SESA defines a *semantic service model for the business services* on which the SESA operates. Particular business services are, however, subject to modeling in application-oriented scenarios. For this purpose, domain-specific ontologies can be designed as part of SESA application design or evaluation. With respect to the distinction between middleware services and business services , the SESA middleware is designed as the SOA on its own. However, it is designed as the facilitator for the integration of semantic business services and as such is not currently considered as semantically enabled but rather as semantically enabling. In order to illustrate the above, Fig. 4.3 depicts middleware and business services within the scope of SEE architecture and SESA, respectively.

4.5.7 Underlying Concepts and Technology

The SESA implementation will build on the underlying concepts and technology provided as the essential input for the work outlined by the roadmap. Development of these concepts and technology started before 2007. SESA builds on and further extends specifications around the conceptual model defined by WSMO, WSML, and reference implementation for the middleware called WSMX. WSMO provides a conceptual model describing all relevant aspects of Web Services in order to facilitate the total or partial automation of service discovery, composition, invocation, etc. The description of WSMO elements is represented using the WSML family of ontology

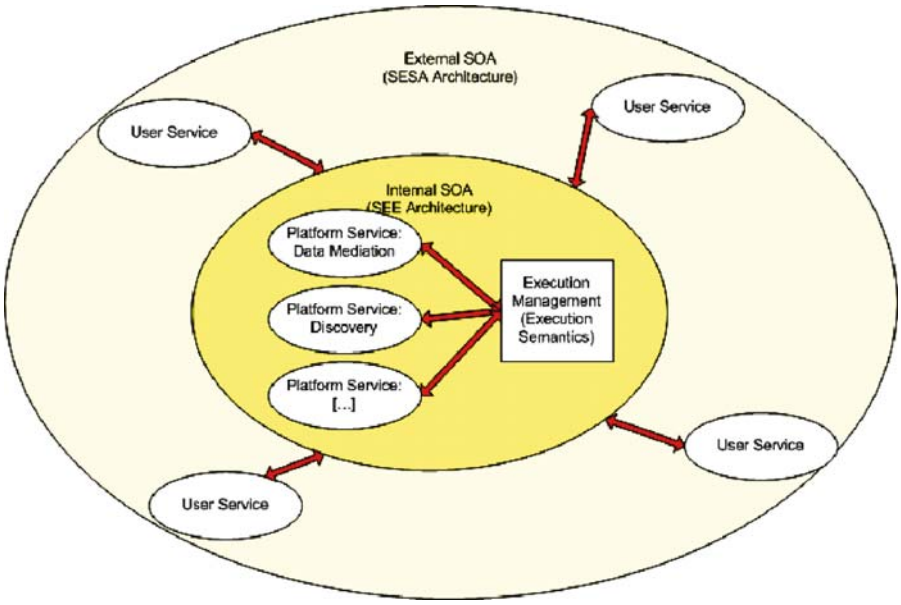

Figure 4.3. Middleware services and business services

languages which consists of a number of variants based on different logical formalisms and different levels of logical expressiveness. WSMO and WSML thus provide grounds for semantic modeling of services as well as for a middleware system which specifically enacts semantic SOAs. Reference implementation of this middleware system is called WSMX. WSMO is the model being developed for purposes of modeling of business services. WSMX is the reference implementation of the SESA providing various functionalities in a form of middleware services which facilitates integration of semantic business services. In addition, the Web Service Modeling Toolkit (WSMT) provides an end-user IDE for modeling of business services and run-time management of the middleware environment.

4.6 SESA Roadmap

With respect to the architecture vision described, we present the scope of the research roadmap for the upcoming years. The research roadmap is defined in a number of research areas, each having defined its goals for the period of the roadmap. Each research goal usually combines major research challenges in Semantic Web Services and SESA together with an implementation effort related to it. The research area and its goals have a corresponding architecture component. Thus, on the basis of the architecture vision and the global view, we identify the following research areas as architectural components which are further distinguished in layers as depicted in Fig. 4.4:

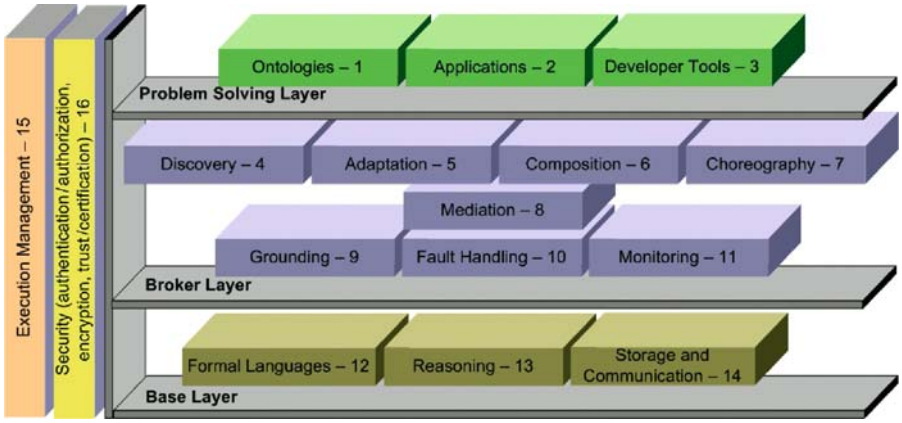

Figure 4.4. SESA components

- The problem-solving layer, which consists of ontologies, applications (e.g., e-tourism, e-business, e-government), and developer tools (GUI tools such as those for engineering ontology/Web service descriptions; generic developer tools such as language APIs, parsers/serializes, converters, etc.).
- The broker layer, which consists of discovery, adaptation (including selection and negotiation), composition (Web service composition techniques such as planning), choreography, mediation (ontology mediation – techniques for combining ontologies and for overcoming differences between ontologies – and process mediation – overcoming differences in message ordering, etc.), grounding, fault handling (compensation, etc.), and monitoring.
- The base layer, which provides the exchange formalism used by the architecture, i.e., formal languages (static ontology and behavioral, i.e., capability/choreography/orchestration, languages, connection between higher-level descriptions), reasoning (techniques for reasoning over formal descriptions; logic programming, description logic, first-order logic, behavioral languages, etc.), and storage and communication.
- Vertical services such as execution management and security (authentication/authorization, encryption, trust/certification).

Each of these components forms a research area for which further goals are identified in following sections.

4.7 SESA Research Areas and Goals

This section describes in more detail research areas which as a result build the functional components that play a role in the SESA. For each research area, the description of the area and the goals are described.

4.7.1 Ontologies

Ontologies are community contracts about a representation of a domain of discourse. Representation here includes (1) formal parts that can be used for machine reasoning and (2) informal parts like natural language descriptions and multimedia elements that help humans establish, maintain, and renew consensus about the meaning of concepts. Ontologies as formal representations of a domain have been proposed for quite a long time as a cure to interoperability problems and problems of application integration, and the Semantic Web community has made a lot of progress in developing stable infrastructure and standardized languages for the representation of ontologies. Also, impressive tools and validated methodologies are available. However, a major bottleneck towards business applications of Semantic Web technology and machine reasoning is the lack of industry-strength ontologies that go beyond academic prototypes. The design of such ontologies from scratch in a textbook-style ontology engineering process is in many cases unattractive, for it would require significant effort, and because the resulting ontologies could not build on top of existing community commitment. Also, real-world problems of data and systems interoperability can only be overcome using Semantic Web technology if ontologies exist that represent the very standards currently in use in systems and databases. Such standards, though mostly informal in nature, are likely the most valuable asset on the way to real business ontologies that can help solve real business interoperability problems, since they reflect some degree of community consensus and contain, readily available, a wealth of concept definitions. However, the transformation of such standards into useful ontologies is not as straightforward as it appears.

Goals and Tasks

- **Maturing Semantic Web foundations**, so that they become compatible with the real-world complexity and scale. In particular, the social interaction and economic dimension of ontologies must be defined.
  - *Ontology engineering*. Methodologies for and prototypes of industry-strength business ontologies, e.g. the gen/tax methodology for deriving ontologies from existing hierarchical standards and taxonomies (UNSPSC, eCl@ss, ...) and eClassOWL, the first serious attempt at building an ontology for e-business applications; and in general advancing the state of the art in e-business data and knowledge engineering, including metrics for content.
  - *Community-driven ontology building*. Methodologies and techniques for evolution of ontologies managed by the user community, including semi-automated approaches and OntoWiki – a Wiki-centric ontology building environment.
  - *Economic aspects of ontology building and usage*. Building ontologies consumes resources, and in an economic setting, these resources are justified and will be spent (by rational economic actors, at least) only if the effort needed to establish and keep alive a consensual representation of a domain of discourse is outweighed by the business gain, in terms of cost, added value, or

strategic dimensions, e.g., process agility. This research branch fuels the use of ontologies in business applications.

- **Building ontologies for core challenges of information systems** in order to realize and evaluate the business benefits and to identify the open research challenges.
  - *Application ontologies*. Ontologies developed for a particular domain such as in e-business, e-government, e-tourism, etc. This includes semantics-supported business process management for mechanization of business process management, ontology-supported electronic procurement, and analysis of the true complexity of business matchmaking, financial reporting, etc.

4.7.2 Applications

All the activities gathered around development of the SESA framework must be tightly coupled to the development and the implementations of the use cases proving the viability of the SESA framework. There are many technologies in the area of Semantic Web Services mainly presented in academic workshops and conferences where various use cases are being defined in alienation. In addition, there does not exist any unified methodology which could be used for comparing these technologies and more importantly, there is no way for industry to evaluate the robustness, applicability, and added values of these technologies. Therefore, progress in scientific development and in industrial adoption is thereby hindered.

Goals and Tasks
It is the goal of applications to analyze processes, infrastructures, and the results of existing real-world scenarios, and to develop a standard set of problems and a public repository for such problems. In addition, the goal is to develop and standardize a community-agreed evaluation methodology:

- **Define the set of standard problems and their levels.** Develop a methodology for evaluating the functionality (versus performance) of semantic service technologies. On the basis of our experience with a group working on the SWS Challenge initiative, applications aim to be involved in a larger community through the W3C Test-Bed Incubator Group.
- **Support of scalable public collaborative development of new problem scenarios and associated services.** In this development, the aim is to standardize the methodology and the infrastructure.
- **The standard methodology for peer review of solutions.** The applications should refrain from recommending technologies or from providing solutions to the Semantic Web Service problems. The focus should be on standardizing the evaluation methodology.

4.7.3 Developer Tools

An IDE is defined as a type of computer software that assists computer programmers to develop software and IDEs such as the Eclipse Java Development Toolkit (JDT) or

the NetBeans IDE for developing software in the Java programming language have proven that the productivity of the Java developer can be improved by providing all the tools required by the developer side by side and integrated with one another. The breadth of the field of semantics means that one IDE for all the different languages and technologies is unlikely to happen; however, using technologies like the Eclipse platform will allow the developer to place individual tools or indeed individual IDEs together in order to build the IDE with the tools needed to perform the job at hand. A good example of where such a combination of IDEs would be useful is the Semantic Web Service field. The Eclipse Web Tool Platform (WTP) provides a collection of tools for simplifying the process of building Web service based applications, combining the WTP with an IDE for describing Web Services semantically. With semantics becoming more centric to modern computer science, many different combinations of tools that at this stage cannot even be contemplated will be required. The flexibility of platforms like Eclipse gives a form of future-proofing and puts the design and scale of the resulting IDE into the hands of the user. When describing tools for semantic technologies it is very easy to become focused primarily on ontologies and to forget that there are many different technologies that require tool support. Research topics like Semantic Web Services and semantic business processes are producing many forms of semantic description that must be created by some developer in order to deploy such technologies. Only now are developers of semantic descriptions receiving limited tool support that allows them to focus on the problem at hand and stop grappling with low-level problems like syntax, testing, and deployment of their descriptions. Within the scope of this roadmap the aim is to provide guidelines for the types of tools that should exist within an IDE for a given semantic technology.

Goals and Tasks

- **Creation and maintenance.** Tool support must be available for creating the actual descriptions themselves. It is important that users of different skill levels are supported within the IDE; thus, editing support at different levels of abstraction should be provided. Some users may be very comfortable dealing with textual semantic descriptions, while others may require more visual paradigms for creating descriptions. These different levels of abstraction can also benefit the skilled engineer. Considering ontologies, it may be more convenient for the engineer to create an ontology using a textual representation within a text editor and then to use a graph-based ontology engineering solution to learn more about the ontology and tweak the model that has been created.
- **Validation.** The most common problem that occurs when creating semantic descriptions is incorrect modeling. It can be very easy for an engineer to make a mistake without any tool support. Validation of semantic descriptions is a nontrivial task and validation at both the syntactic and the semantic level can vastly reduce the time developers spend debugging their descriptions. By syntactic validation we mean checking that the actual syntactic structure of the semantic description is correct and by semantic validation we refer to checking that syntactically correct descriptions are semantically valid.

- **Testing.** Once valid semantic descriptions exist the engineer needs to ensure that they behave in the expected manner in their intended environment prior to deploying them. Having testing integrated into the development environment reduces the overhead of the user performing a lengthy, iterative, deploy-test scenario. The engineer will more than likely perform a deploy-test scenario anyway, but having an initial cycle within the development environment can significantly reduce the length of this cycle and the time taken to perform it.
- **Deployment.** Ultimately the descriptions created within the development environment must be used in some run-time system. Deploying descriptions can also be a huge overhead for the engineer and having tool support in an IDE can prevent mistakes occurring at this crucial stage of the process.

4.7.4 Discovery

Within a SOA the discovery of services is the essential building block for creating and utilizing dynamically created applications. However, current technologies only provide a means to describe service interfaces on a syntactical level, providing only limited automation support. Existing solutions for service discovery include UDDI, a standard that allows programmatically publishing and retrieving a set of structured information belonging to a Web service; however, it allows one to retrieve services by predefined categories and keywords, not by their actual semantic properties. There is a lack of means that allow the description of functional and nonfunctional properties of a service on a semantic level. Only with such descriptions is a precise discovery possible.

Given a description of a service, the problem of discovering a desired service can be seen as an information-retrieval problem, which uses keywords to express the desire and uses a document index to match them. However, for mechanizing the discovery task a more fine-grained approach to discovery is required, e.g., to restrict the search space along specific parameters, like location, provider, price, etc. It can be seen as a search for a semistructured entity. Here approaches developed in the context of the Semantic Web, in particular the use of ontologies are a promising approach.

Goals and Tasks

- **Service and domain ontologies.** In order to provide a semantic discovery service both service requests and offers need to be described on a semantic level. While there exist some proposals for upper-level ontologies like WSMO and OWL-S, we need to refine them and provide guidelines for their usage and accompanying domain ontologies.
- **Language and reasoning integration.** Potentially many different logical formalisms can be used to annotate services. Each comes with a specific trade-off between expressivity and computational complexity. It has to be investigated for which use cases a particular formalism is suitable. In addition the reasoning engine for a particular formalism needs to be integrated into the discovery context, such that its usage becomes transparent to the user of a discovery engine.

- **Nonfunctional properties.** Research around service discovery has so far paid much attention to the specification of the functional properties of a service; however, only little effort has been spent on the investigation of the usage of nonfunctional properties within the discovery process. Specific ontologies and matchmaking techniques have to be developed in order to allow a semantic retrieval on nonfunctional properties.
- **Field deployment and verification of existing discovery strategies.** While many concrete formalisms have been proposed, only a few case studies have been performed to validate the appropriateness of a particular approach. Further real-world use cases are required in order to adopt the existing semantic discovery approaches for practical needs.

4.7.5 Adaptation

After discovering a set of potentially useful Web Services, a semantic user agent needs to find out the concrete offers available at the Web Services and that are relevant to the user's goal, generally by communicating with the Web service or with its provider. This process filters out the discovered Web Services that cannot fulfill the goal. This step is required as it is not feasible for a Web service to provide an exhaustive semantic description of all its potential offers. The process of checking whether and under what conditions a service can fulfill a concrete goal is called negotiation in SESA. The results of negotiation are filtered using the functionality of the discovery component to consider only the services that have the appropriate functionality and also nonfunctional properties acceptable to the user. Filtering is followed by building a ranking/order relation based on nonfunctional property criteria like price, availability, etc. Once a list of Web Services than can fulfill the user's concrete goal has been prepared, a SESA must then choose one of the services to invoke. It is important that this selection is tailored to the user's needs, as, for example, while one user may require high quality, another may prefer low price. This process is called selection. Negotiation, ranking, and selection are tasks of the Adaptation Working Group.

Goals and Tasks

- **Semantic and multicriteria based ranking and selection.** Ranking and selection of services could be done along more than one dimension. For example, users might be interested in the cheapest and fastest service providing a certain functionality. A ranking and selection solution which uses semantic descriptions of multiple nonfunctional properties and ranks the services on the basis of their attached logical expressions' nonfunctional properties representations should be developed. This involves development of semantic and multicriteria ranking algorithms and design and implementation of a semantic and multicriteria ranking component.
- **Context-based ranking and selection.** Service ranking and selection must consider contextual information in order to provide relevant results. The goal is to

develop models and algorithms for context-aware ranking and selection. This involves development of context-ranking algorithms.

- **Social-based ranking and selection.** Service ranking and selection remains an open and controversial problem in terms of both social and technical aspects. From a social point of view an honest and fair mechanism is required. An aspect which might be useful especially for ranking services is the "social" aspect of consuming services. Previous customers who have used a service could provide feedback about the service. Furthermore not only users but also groups of users, communities, can be used to "compute" the ranking values of services.
- **Negotiation algorithms.** Negotiation support with use of communication to establish details of Web service offers relevant to user's needs.

4.7.6 Composition

Composition involves methods for Web service composition (WSC), starting from Web service descriptions at various levels of abstraction, specifically, the functional level and the process level. The WSC area is stillat a very early stage of its development. Several techniques have been proposed, mostly based on AI planning or on logical deduction, but all of those still have severe shortcomings. The existing techniques (1) largely or even completely ignore the constraints given in the background ontology in which the Web Services are specified, or (2) largely or even completely ignore the complex inner behavior and interfaces of Web Services, or (3) have severe scalability problems, solving only toy problems with few services, or they suffer from several of these deficiencies.

Goals and Tasks

- **Development of a scalable tool for WSC with powerful background ontologies and partial matches.** The goal is to overcome the lack of a technique that combines adequate treatment of background ontologies (deficiency 1 described above) with scalability (deficiency 3 described above). This will be achieved by building on logics-based search space representations and heuristic functions originating in the area of AI planning.
- **Development of a scalable tool for WSC with plug-in matches, dealing with business policies.** The goal is to overcome the lack of a combination of deficiencies 1 and 3 described above through a particular focus on business process management scenarios. In those, scalability is particularly urgent, since enterprises deal with thousands of services from which the composed service should be combined. Further, business policies – rules governing how services can be executed within or between enterprises – are of paramount importance. Scalability, even in the presence of business policies, will be achieved by exploiting plug-in matches rather than partial matches, and by exploiting the typical forms of ontologies occurring in practice.
- **Integration of techniques for functional-level and process-level WSC.** The goal is to overcome the lack of a combination of deficiencies 2 and 3 described

above, and potentially to build technology that overcomes all three deficiencies (1–3). The idea is to combine techniques from functional-level and process-level WSC, first by establishing ways for their interplay, and later by integrating their underlying core principles. Regarding their interplay, functional-level and process-level WSC essentially provide different trade-offs between accuracy and computational cost; they can be combined to mutually profit from each other's benefits. Regarding the underlying principles, the most effective process-level composition methods today are based on binary decision diagrams; this will be replaced with the logics-based search space representation underlying advanced functional-level composition. This approach allows for more flexibility and can be used to model and take into account also the background ontology, while at the same time reducing computational costs through consequent exploitation of problem structure.

4.7.7 Choreography

Techniques for service choreography play a key role in creation of new opportunities for collaborations between service requesters and providers, and thus for creation of new services. The choreography part of SESA defines formal specifications of interactions and processes between the service providers and clients. Current approaches to service choreography languages have been criticized for being too procedurally oriented. With the move towards service orientation, where entities are autonomous and need to agree on the collaborations between them, where no central point of control might exist, a more declarative modeling style for interactions is required (i.e., "what" without having to state the "how"). Moreover, reasoning techniques for such a language that would enable a flexible and dynamic integration of service requesters and providers in a collaborative environment are currently missing.

Goals and Tasks

- **Declarative choreography language.** The goal here is to develop a declarative process language which should allow for formal specifications of interactions and processes between the service providers and clients, Such a declarative language would enable non-IT experts to easily represent service behaviors and interactions, enabling a more flexible way of engaging in new collaborations.
- **Reasoning tasks for choreography.** The goal here is to define reasoning tasks that should be performed using the declarative language. Verification techniques such as contracting or enactments are examples of reasoning tasks. Such techniques will enable an automated, flexible, and dynamic integration of service requesters and providers in a collaborative environment.
- **Tool support for choreography.** The goal here is to implement an engine to support the execution of interactions, as well as to support reasoning in the proposed declarative language.

4.7.8 Mediation

Heterogeneity problems and mediators have been intensively investigated in the last decade but the key solution that would enable the decisive leap towards automation is yet to be found. Semantics is changing the problem specifications and service-orientation paradigms offer new ways of designing, deploying, and using mediators while at the same time posing new challenges and setting new requirements. That is, data and processes can be formally and unambiguously described, while services and SOAs allow the development of decoupled, transparent, and flexible software components, including mediators.

Goals and Tasks

- **Advanced support for data mediation.** Semiautomatic design-time tools should be developed in order to allow domain experts to identify and capture the heterogeneity problems between different models of overlapping domains. Special attention will be given to user profiles and expertise levels in order to separate the tools for trained domain experts and the tools designated for casual users of ontologies. Furthermore, at this level, alignments between various models will be part of a community validation process where users can add and remove links between the models in order to achieve and maintain agreed-upon interlinked models.
- **Advanced support for process mediation.** Heterogeneity appears on the process level as well no matter whether these processes are enterprise internal processes or public processes used in describing the visible behavior of particular services. Such heterogeneous processes need to be part of collaborative scenarios that can range from simple peer-to-peer interaction to complex compositions. Semiautomatic tool support allowing the tailoring of such processes in order to overcome the heterogeneity problems should be provided. Further, more such tools should support annotation of existing process representation standards with semantics based on ontological domain models.
- **Service mediation by mediation services.** Mediator systems able to resolve specific types of heterogeneity problems should be encapsulated and deployed as mediation services. Such services should be developed for well-defined mediation scenarios while preserving the generality of their offered functionality.
- **Semantic descriptions for the mediation services and mediation libraries.** Mediation services should be semantically described as any other resources. In this way their functionality can be properly advertised and their intended usage explicitly stated. Furthermore, they can become part of an intelligent mechanism for service discovery, composition, and invocation. Additionally, such semantically described mediation services will be organized in semantic mediation patterns that can be directly applied in complex heterogeneity scenarios. In addition, such services should be organized in libraries supporting intelligent mediation service retrieval (by providing customizable mediators classifications), patterns construction (based on the semantic descriptions of the mediation service and on the mediation goal to be achieved), and governance mechanisms (by exploring service and patterns dependencies and impact analysis).

4.7.9 Grounding

Legacy systems represent valuable assets for most of their owners and usually completely replacing them is not an option. As such, methodologies that will allow the integration of these systems with the new emerging paradigms and technologies have to be developed. For example, XML schemas and XML data have to be lifted to ontological level in order to allow semantic-aware systems to act on this data. In addition, since all tasks related to discovery, selection, composition, etc. operate in SESA on semantic descriptions, the link between a semantic service and the underlying technology for communication (e.g., HTTP, SOAP, etc.) needs to be defined. The basis for grounding has been established within the W3C Semantic Annotations for WSDL Working Group (SAWSDL WG), allowing hooking semantic descriptions with WSDL elements.

Goals and Tasks

- **Semiautomatic tools allowing the creation of transformation between syntax-based and semantic models.** Specifying the transformations between Web service XML messages and the semantic data currently requires deep knowledge both of the structure of the XML message and of the ontology. Methods for semiautomated creation of grounding should be created.
- **Grounding to other specifications.** Grounding definitions to specifications other than WSDL should be defined. It should be possible to use, e.g., REST services with semantic descriptions, etc.

4.7.10 Fault Handling

Fault handling defines the handling of faults or errors occurring within the execution of end-point Web Services. Fault handling is important in SESA as the effect of a Web service failure will reach beyond the scope of the individual service if it is part of a composition. The aim of fault handling is to ensure that the failure of the service has a minimal impact, and that the most appropriate action is taken to deal with it. This may be simply returning an error message to the user and terminating any further actions. It may involve more complex tasks such as replacement of a failed service with a substitute service, or rolling back actions previously carried out by other services in a composition .Currently the BPEL specification has some provision for fault handling via the catches and some support for rollback. The WS-Transaction family of specifications from OASIS also has provision for basic fault handling, including distinguishing application faults from communication faults.

Goals and Tasks

- **Increased automation of fault handling.** Semantic descriptions of services in SESA will enable an increased level of automation in fault handling. Current fault handling techniques allow either a very generic level of fault handling (e.g., gracefully exit and report errors) for all services, or require specific actions to be

hard-coded for each service. The aim will be able to make full use of the semantic descriptions and reasoning to make intelligent decisions about fault handling automatically (e.g., in this context a particular error can be ignored).

- **More complex fault handling tasks.** Tasks such as automatic service substitution are rarely carried out in current SOA-based systems, as there is not sufficient semantic information about services to allow an equivalent service to be identified and substituted. The aim will be to increase the use of more complex fault handling tasks in SESA with the aid of semantic descriptions and reasoning.

4.7.11 Monitoring

Monitoring is concerned with checking the progress of service execution and advising the user/agent of abnormal events. Within SESA, monitoring of services is essential as the behavior of individual services has an effect on successful completion of a composition. Detecting failure or abnormalities will require some action to be taken (e.g., rollback previous actions, or substitute failed service).

Currently the most popular methods for Web service monitoring are based around the Management Using Web Services (MUWS) OASIS specification, which defines a flexible, expandable approach to monitor manageable resources. Specifically Web service monitoring is handled under a subspecification called Management of Web Services (MOWS).

MUWS defines how the ability to manage, or, how the manageability of, an arbitrary resource can be made accessible via Web Services. Manageable resources can be accessed with a Web service end point. The manageability consumer exchanges messages with the end point in order to subscribe to events, get events, and request. The type of information available is things such as number of requests, number of failed requests, number of successful requests, service time, maximum response time, and last response time.

Goals and Tasks

- **Monitoring framework based on ontologies.** Currently MUWS defines parameters for monitoring based on a rigid XML schema. A move to a more flexible ontology-based definition would give more expressivity and allow monitoring to be tailored to a specific context.
- **Increased automation of monitoring and link to fault handling.** In a more automated SESA-based system, the system should be able to proactively monitor the execution of services, identify when a particular abnormality has occurred, and take the best action at the time to deal with it. For example, if a service failed the system should detect it and replace it with another service that performs the same task. Current methods of monitoring will identify problems with Web Services but there is no automatic next step to fault handing and recovery. This should be included in the scope of SESA.

4.7.12 Formal Languages

Descriptions in SESA need different formal languages for the specification of different aspects of knowledge and services. These descriptions can be decomposed into four dimensions: static knowledge (ontologies), functional description (capabilities), behavioral description, and nonfunctional properties. There are several knowledge-representation formalisms used for formal languages, including description logic and logic programming, datalog subset of F-logic, Horn subset of F-logic with negation under the well-founded semantics, description logic SHIQ, first-order language with nonmonotinic extensions, etc.

Goals and Tasks
The major objective is to integrate first order logic based and nonmonotonic logic programming based languages, the explicitization of context for use with scoped negation, and the development of rules for the Semantic Web (through the W3C RIF working group). Furthermore, requirements for the functional descriptions of services and as well as semantics for Web service functionality need to be devised. Requirements need to be gathered for the description of a choreography and an orchestration and a semantics needs to be devised. Finally, the purpose and usage of nonfunctional requirements need to be investigated. In particular, the goals will be:

- **Integrating knowledge based on classical first-order logic and nonmonotonic logic programming.** Important issues are the representational adequacy of the integration, as well as decidable subsets and a proof theory, so that reasoning becomes possible; scoped default negation; rules for the Semantic Web – RIF Working Group; connection between Semantic Web languages RDF, OWL.
- **Functional specification of services and a semantics needs to be devised** which can be combined with the language for the description of ontologies, in order to enable the use of ontologies for the description of Web service functionality. An important use case for the functional description of services is discovery. Therefore, it is expected that many requirements for the functional description of services will come from the discovery research goal.
- **Advanced behavioral description.** There exist several formal languages which are suitable for behavioral description. Examples are transaction logic, situation calculus, and action languages. Requirements need to be gathered for the description of choreography and an orchestration and semantics needs to be devised. A key challenge is the combination of this language with ontology languages in order to enable the reuse of ontology vocabulary in the choreography and orchestration descriptions. Finally, this language needs to be connected to the language for capability description in order to prove certain correspondences between the functional and behavioral descriptions of services.
- **Nonfunctional properties.** Nonfunctional properties can at least be divided into two categories: (1) metadata, e.g., author, description, etc., of the WSML statements in a description and (2) actual nonfunctional properties, i.e., actual properties of services (e.g., pricing, QoS, transactions). Nonfunctional properties require a deeper investigation into their purpose and their usage.

4.7.13 Reasoning

The SESA necessitates effective reasoning for different tasks such as service discovery, process and data mediation, and integration. To enable processing of these tasks in an automated manner, the SESA utilizes machine reasoning over formally represented service specifications. We are developing an Integrated Rule Inference System (IRIS) which is a scalable and extensible reasoner tool for WSML. The system implements different deductive database algorithms and novel optimization techniques.

Goals and Tasks

- **Reasoning techniques with large data sets.** In the context of the Semantic Web, applications might require vast volumes of data to be processed in a short time. Current reasoning algorithms are developed rather for small, closed, trustworthy, consistent, and static domains. Therefore, these algorithms need to be extended and adapted in order to be applied to large and dynamically changing knowledge bases. One challenging approach to achieve a scalable reasoning is to combine existing deductive and database techniques with methods for searching the Web (utilizing semantic annotations). This line of research considers reasoning in distributed environments as well.
- **New techniques for description logics reasoning.** Description logics are a family of knowledge-representation formalisms characterized by sound, complete, and (empirically) tractable reasoning. However, applications in areas such as e-science and the Semantic Web are already stretching the capabilities of existing description logics systems. Key issues here are the provision of efficient algorithms that allow (advanced) applications (1) to scale up to knowledge bases of practical relevance and (2) to leverage expressive languages for capturing domain knowledge.
- **Reasoning with integrating frameworks based on classical first-order logic and nonmonotonic logic programming.** Two lines of research will be explored:
  - Reasoning with decidable fragments of such integrating frameworks.
  - Reasoning with undecidable fragments using proof-theoretic techniques.

4.7.14 Storage and Communication

Storage and communication form the underlying mechanisms of the SESA needed for coordination of the execution of middleware services within the platform. The novel communication and coordination paradigm is called triple-space computing (TSC). TSC is recently receiving attention in open distributed systems like the World Wide Web and pervasive computing environments. TSC supports the Web's dissemination idea of persistently publishing and reading. Furthermore, it is based on the convergence of tuple-space technology (originating from Linda) and Semantic Web (service) technology where RDF triples provide the natural link from tuple spaces to triple spaces. Having machine-understandable semantics integrated in the

middleware makes this approach particularly useful for SESAs. TSC can be used for dynamic management of middleware services, coordination of middleware services, resource management and external communication with SESA.

Goals and Tasks

- **TSC establishment.** Interaction interfaces, specification of security and trust support, ontology-driven space management for the development of self-adaptive, and reflective triple-space kernels for the integration of scalable semantic clustering and distributed querying.
- **Dynamic management of middleware services.** Asynchronous coordination of middleware services through local triple spaces, running middleware processes over triple spaces.
- **Resource management.** A unified storage infrastructure with standardized access policies and interfaces replacing dedicated repositories and hiding the complexity of different types of resources. Thus, interfacing of triple-space kernels with the resource management, installing RDF-based access to resources, and management of distributed resources need to be developed.
- **External communication with SESA based on TSC.** SOAP-enabled Web service execution over triple spaces, grounding of SOAP messages to TSC, lifting of RDF-based messages to SOAP.

4.7.15 Execution Management

Execution management as the kernel of the SESA is responsible for the coordination of middleware services. It realizes the overall operational semantics of the middleware which lets the system achieve the functional semantics of its client-side interface. It orchestrates the functionality of the middleware services into a coherent process in an orderly and consistent fashion. This process is defined by so -alled execution semantics which defines how particular middleware services need to interact so that SESA can provide particular functionality to its users. The research focuses on the functional as well as the operational combination of the individuals services of the middleware.

Goals and Tasks

- **Definition and refinement of execution semantics.** Definition of various execution semantics for particular execution scenarios.
- **TSC Integration.** Interkernel communication and coordination, distributed execution of tasks;. The communication between execution management and middleware services through publishing and subscribing to the data as sets of triples over triple space. Other than the benefits of asynchronous communication, it will achieve decoupling of individual middleware services.

4.7.16 Security

In the context of SESA, security will cover many areas, including authenticating access to services, preventing misuse of services, encryption, and data privacy. Security will be an important concern to ensure that services are accessed correctly, by the authorized people, and that confidential or sensitive data is securely stored and transmitted. There are currently many Web Services standards relating to security, the most prominent being the recently completed WS-Security 1.1 specification from OASIS.

4.8 Summary

This chapter outlined a comprehensive framework that integrates two complimentary and revolutionary technical advances, SOAs and the Semantic Web, into a single computing architecture that we call SESA. We presented an emerging SOC paradigm and the implemental architecture SOA as the starting point. After analyzing the advantages and shortcomings of SOA, we provided the Semantic Web Services execution environment challenges and requirements, all of which are involved with the semantics extension on SOA. Based on those prerequisites, SESA is proposed as a Semantic Web Services execution environment. While SOA is widely acknowledged for its potential to revolutionize the world of computing, this success is dependent on resolving two fundamental challenges that SOA does not address, namely, integration, and search or mediation. In a service-oriented world, millions of services must be discovered and selected on the basis of requirements, then orchestrated and adapted or integrated. SOA depends on but does not address either search or integration. Basically, we provide SESA grounding principles and global views with various layers. More detailed component description and technical building blocks are provided in the following chapters.

http://www.springer.com/978-3-540-77019-0

Implementing Semantic Web Services
The SESA Framework
Fensel, D.; Kerrigan, M.; Zaremba, M. (Eds.)
2008, XVI, 322 p. 59 illus., Hardcover
ISBN: 978-3-540-77019-0