SicAri – A security architecture and its tools for ubiquitous Internet usage

# Deliverable MW1

Interoperability with
Component Standards and Web Services

Version 1.0,    December 31, 2004

Jan Oetting, usd.de
Jan Peters, Fraunhofer IGD

# Contents

# 1 Introduction

According to the web service architecture as defined by W3C [49], a *web service* is an abstract notation which is implemented by a concrete *agent* as computational resource, owned by and acting on behalf of a person or organization. An agent realizes one or more services and may request other services, in turn. The web service specification ensures the interoperability between systems through machine-to-machine interaction over a network, whereas it avoids any attempt to govern the implementation of agents.

On the one hand, this agent concept is defined on a high level of abstraction neglecting detailed characteristics of the numerous existing agents systems. Furthermore, most current applications either base on web service or on software agent technology, but not on both simultaneously. On the other hand, the web service specification already indicates the integration of both technologies resp. hints at similarities and possible extensions/improvements of one technology through the other.

Similar to web services, software agents can encapsulate business or application logic. Rather, software agents can dynamically discover, combine and execute such processes, and further offer multiple services or behaviors, that can be processed concurrently. In order to move from one system to another, or even to communicate with each other, mobile agents currently need a common platform on which they operate. Thus, they are useful for business partners only if these actually share a common platform. The consistent use of web service standards for description of capabilities, communication, and agent discovery would establish interoperability not only between differ agent platforms but also between agent platforms and traditional web services. Thus, the advantages of two worlds could be combined.

Furthermore, coherence specifically with *mobile* agents can already be seen in some of current industrial approaches of implementing services for the web resp. implementing web services [24, 18]. In contrast to static agents, mobile agents have the ability of processing information directly at the source of data, whereas bandwidth is conserved by the avoidance of unnecessary network communication. To dynamically deploy both technologies without much overhead for the application developer mobile agents and web services have to be integrated in a seamless manner, whereas mapping processes have to be fully automated.

In the remainder of this section, we give a short overview about both web service and mobile agent technology, subsequently we discuss a generic integration approach of these technologies, and then describe two concrete example scenarios. Related work is reviewed in Section 2 with respect to of agent interoperability, web service enhancements, and the integration of static as well as mobile agents and web services. The proposed architectural model is covered by Section 3, which is followed by integration details in Sections 4 and security concerns in Section 5. Finally, we discuss our approach in comparison with existing contributions in the field (see Section 6).

## 1.1 Web Service Technology

Web services are modular software components that can be invoked via the internet. A web service itself is not an independent application but an interface between the web and the application logic, whereas regular web servers provide the basic transport protocol (HTTP). The web service standards have been defined to provide the foundation for an open web service framework [49], and thereby to allow interoperable machine-to-machine interaction over a network.

Web services communicate via platform and programming language independent protocols that are based on XML [48]. The three basic web service standards specify self description, communication, and localization of web services. The *Web Service Description Language* (WSDL) [9] is used for a formal self description of the service; namely the protocol bindings and message formats required to interact with the web service. The *Simple Object Access Protocol* (SOAP) [47] is used for synchronous communication with web services by means of invoking its functionality. The third main component of the web service technology is the localization service named *Universal Description, Discovery and Integration* (UDDI) [33]. UDDI defines a platform-independent registry to list services on the internet. It is designed to be interrogated by SOAP messages and to provide access to WSDL documents.

Web services are close to the point of becoming a mainstream technology, whereas they already impact a broad range of commerce and industry. Even legacy applications are integrated into todays business logic by means of web service wrappers for their functionality. Emerging Grid computing frameworks, as there is the Globus Toolkit[1], often imply the use of web services and thereby support this process. Current research in the area of semantic web is also strongly related to web services.

## 1.2 Mobile Agent Technology

In our opinion, the most promising way to integrate mobile services is by means of mobile software agents. Franklin et al. [19] have reviewed and studied 11 definitions of agents in the context of information technology and computer science. Based on those specific definitions, we combined characteristic features to a basic agent type including: *reactivity*, agents react on environmental changes; *autonomy*, agents have control over their actions; *proactivity*, agents do not only react on changes of the environment; *continuousness*, the agent's process runs without interruption from its creation until the process finally dies. Talking of mobile agents in the following, we refer to an agent as a combination of these features adding *mobility* and the ability of *communication*. Such an agent can be seen as a tuple of program code, data and execution state, migrating from one execution environment (an agent server) to another. Accessing resources locally enables agents to collect, to process and to publish data efficiently within a network. Agents roam a network, seek information, and carry out tasks on behalf of their senders autonomously. Hence, mobile agents offer great benefits to applications in networks by adding client-side intelligence and functionality to server-side services.

Almost each mobile agent system is founded on research activities with a specific focus and therefore follows a different design goal. Hence, each mobile agent system offers different strengths, such as security, scalability, enhanced agent behavior, efficient migration, asynchronous messaging, etc.. A detailed view on the agent model presupposed for this work will be given in Section 4.1.

## 1.3 Integration

The integration of web service and mobile agent technology can be realized in many ways, depending on the specific application scenario. In this section we first give generic require-

[1]Globus Toolkit - The implementation of the Open Grid Service Architecture (OGSA). http://www-unix.globus.org/toolkit/

ments for the seamless integration of these two technologies. Then we present two integration scenarios as examples.

### 1.3.1 Assumptions, Requirements, and Definitions

To integrate mobile agent and web service technology in a seamless manner, components have to be designed, which map between the different mechanisms for *service description*, *service invocation*, and *service discovery*, in both worlds. In other words, messages resp. representations from the according web service protocols (WSDL, SOAP, UDDI) have to be translated into corresponding requests resp. data types of the agent system, and vice versa.

When we talk of a *web service engine* in the remainder of this report, we mean the summary of these components, which can be integrated into an existing agent system, and thereby extend this system with web service abilities.

Especially in some communication-centered agent systems, this engine further has to bridge the gap between the synchronous behavior of SOAP, and the asynchronous messaging paradigm as it is specified by FIPA with FIPA-ACL, for example (cf. Section 2).

Describing the level of integration, we want to define the following features for the integration of web services and agents:

**self-contained** The web service engine integrates all components for a seamless transition from one technology to the other, without the need of additional external resources.

**bidirectional** The web service engine supports the invocation of web services by agents as well as the provisioning of agent services by means of web services. Thus, it allows cross boundary interaction in both directions.

**automated** After being properly configured and started, the web service engine does not require any further manual steps executed by the user during runtime.

**transparent** The components of the web service engine are transparently integrated into the agent system resp. the web service infrastructure. Neither agents nor web services as the acting entities recognize the existence of the web service engine.

In the following, we illuminate some more details of the required integration processes. Assuming, that service description, discovery, and invocation has already been solved separately within both technologies, we only consider the two interesting cases which cross the virtual boundary. Furthermore, we assume the integration being automated and transparent:

**Agents offer web services (service provider)** An agent offering encapsulated functionality as service, generally registers the service within the local agent runtime environment, or at a dedicated directory server within the agent system infrastructure.

⇒ A *web service server stub* has to be created which accepts SOAP requests, transfers these to the agent service by means of the agent system's interaction paradigm (method invocation, message-based, event-based, etc.), waits for an optional result, and finally transfers the result back to the requester as SOAP message.

⇒ The web service stub has to be made available through a *web service gateway* which provides access to the stub over an appropriate transport protocol (HTTP, SMTP, FTP, etc.).

⇒ Simultaneously, a WSDL-based *web service description* has to be generated, which contains the service description and the link to the web service stub at the web service gateway. Dependent on the type of agent description, a syntactic service interface description can be enriched by semantic information from an ontology or resource definition associated with the agent service (OWL, RDF, etc.).

⇒ The WSDL description has to be transformed into an appropriate UDDI business entity containing the tModel (service interface) and binding template (pointer to web service stub). This business entity is then registered at a UDDI-compliant *web service registry*.

**Agents utilize web services (service client)** An agent planing to interact with another service, usually searches for a corresponding service instance by a given service description. For service discovery, the local agent runtime environment or the directory server of the agent system infrastructure is requested.

⇒ Either a UDDI-compliant *web service registry* automatically transforms the service descriptions of newly registered web services into the appropriate service description of the associated agent system (*active mapping*),

⇒ Or the agent system resp. agent directory server implicitly triggers associated web service registries upon search request from an agent (*passive mapping*). In this case, a given agent service description is transformed into a UDDI business entity, which is then mapped against already registered *web services descriptions*.

⇒ Dependent of the web service description a specific *web service client stub* is created, which accepts requests by means of the agent system's interaction paradigm, and forwards these as SOAP messages over the corresponding transport protocol to the associated web service. In case of active mapping, this web service stub is created and integrated into the agent system, when a new web service is registered. In case of passive mapping this process is executed upon search request.

⇒ In either case, the agent implicitly invokes the web service through the web service stub using the usual interaction paradigm of the agent system.

In a compound scenario, one agent might offer a functionality provided as web service which in turn can be utilized by another agent acting as web service client. Assuming *transparent integration*, as we have defined it, the service as well as the client agent should not recognize any "agent service to web service" resp. "web service to agent service" mappings during the service description, discovery, and invocation process. In other words, they should only need to know and utilize the usual mechanisms of their specific agent environment.

If this scenario is logically limited to one agent system or even limited physically to one agent platform, the agent system should provide internal mechanisms to circumvent cross-boundary mappings, and thereby optimize the single processes. If this scenario includes service and client agents from different agent systems, the web service integration engine does not only provide compatibility to the web service world, but implicitly represents a layer for *agent system interoperability*.

*Hybrid systems* are conceivable, whereas an external web service call from a legacy but web service enabled system is transfered to a mobile agent. This agent is subsequently initiated to fulfill its task within the mobile agent infrastructure and finally returns a result which is automatically transformed into a corresponding response message. Respecting given time constraints of the synchronous web service protocol, the agent might visit another agent platform, initiate and interact with further agents, or even invoke other web services. The other way round, an agent is able to initiate a workflow within a web service environment.

As we talk of mobile agents based web services, i.e. program code is transported to another host and executed locally within a foreign environment, we must not neglect the corresponding *security aspects*. Within the mobile agent community there has been elaborate discussion about security concerns in the past. A lot of solutions and protocols have been proposed which prevent or at least reveal malicious behavior dependent on the specific application. When integrating web services and mobile agents it has to be assured that the new mean of service interaction does not bypass or override existing security mechanisms of the agent system.

Although web services are usually meant to be stateless and static service components, we finally want to underline one of the mobile agent's benefits as there is the migration of context and state to a new location of execution, and thereby introduce *stateful* mobile web services as another enhancement of regular web service frameworks.

### 1.3.2 Example Scenarios

As shown in Figure 1 the first scenario distinguishes the following three entities within a network community:

**Provider** A *provider* allows access to locally hosted resources through web services, implemented as components within a regular application server. Further he runs a mobile agent platform which accepts agents from community members and grants them local access to the provided web services.

**Community Member** The *community member* invests his knowledge to implement a service with added value which is based on functionality of the provider's web services. Embedding this compound service in a mobile agent, and subsequently migrating this agent to the provider's agent platform, it is able to benefit from the optimized access to local services.

**Regular client** If the community member's agent in turn registers the API for its compound service within the agent platform, this service can subsequently be invoked as web service by the community member and other *regular clients* through the agent platform's web service engine.

In this provider-based network community (e.g. a gaming platform), a community member is able to improve the provider's web service framework for himself and other community members. On the one hand, the community member does not need to setup its own web service framework to provide his services with added value. The compound service further benefits of

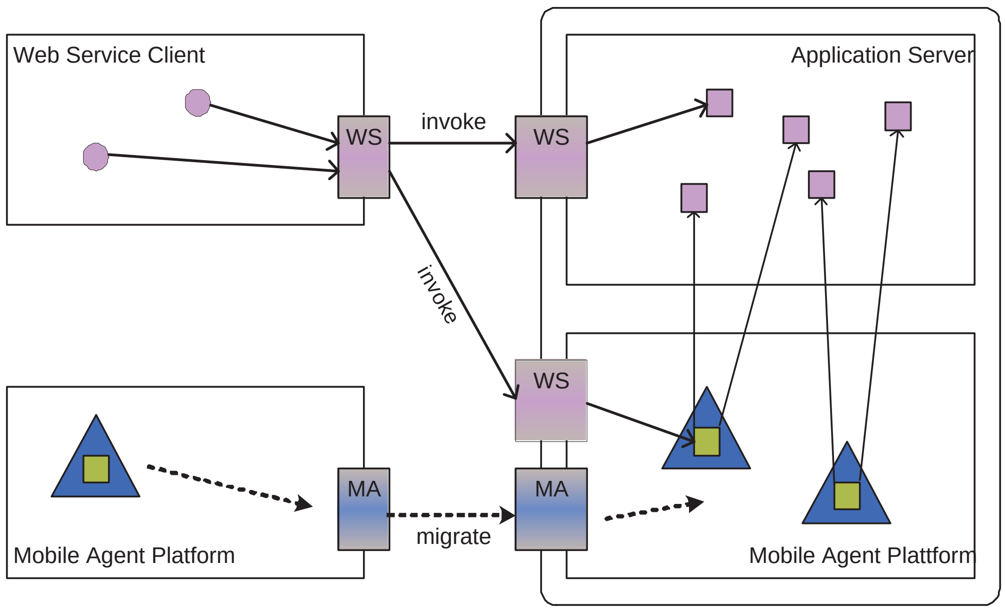


Figure 1: A mobile agent (△) embedding a compound service migrates from its origin to a provider's agent platform, and subsequently accesses services (□) within a locally running application server to fulfill its task. By registering the compound service within the agent platform this service with added value is provided as web service, not only to its owner, but to every regular web service client (○).

local access to resources. On the other hand, the provider is still able to select and encapsulate incoming agents by enforcing its security policy through the agent platform. Thereby, he benefits of community members improving the web service framework he provides, and is in control of the additional features at the same time.

The second scenario describes the integration of mobile agent's load balancing abilities (see [6]) into a web service framework. Figure 2 depicts a web service client on the left and a web service provider on the right.

Instead of running a number of application servers, the provider installs its services by means of mobile agents, running on distributed agent platforms. Transparently for the agents, the agent platforms provide these services as web services to the outside world. In case, the work load on a local platform exceeds a certain limit, single service agents might migrate to another platforms, while an UDDI registry hosted by the provider is implicitly informed about resulting service movements. A web service client, searching for a specific functionality, requests the UDDI registry for current information about the provided web services, and subsequently invokes the returned service instance. Especially when providing *stateful* services with respect to enhanced client session management, or server-sided service contexts, this approach has further advantages. In case, a single server needs to be shut down or restarted, all running services including their current context and state can be moved to a backup server beforehand.

As referred in the following section, these are just two examples among a couple of others. Nevertheless, the reader should have got a glance of the possibilities which open up to the

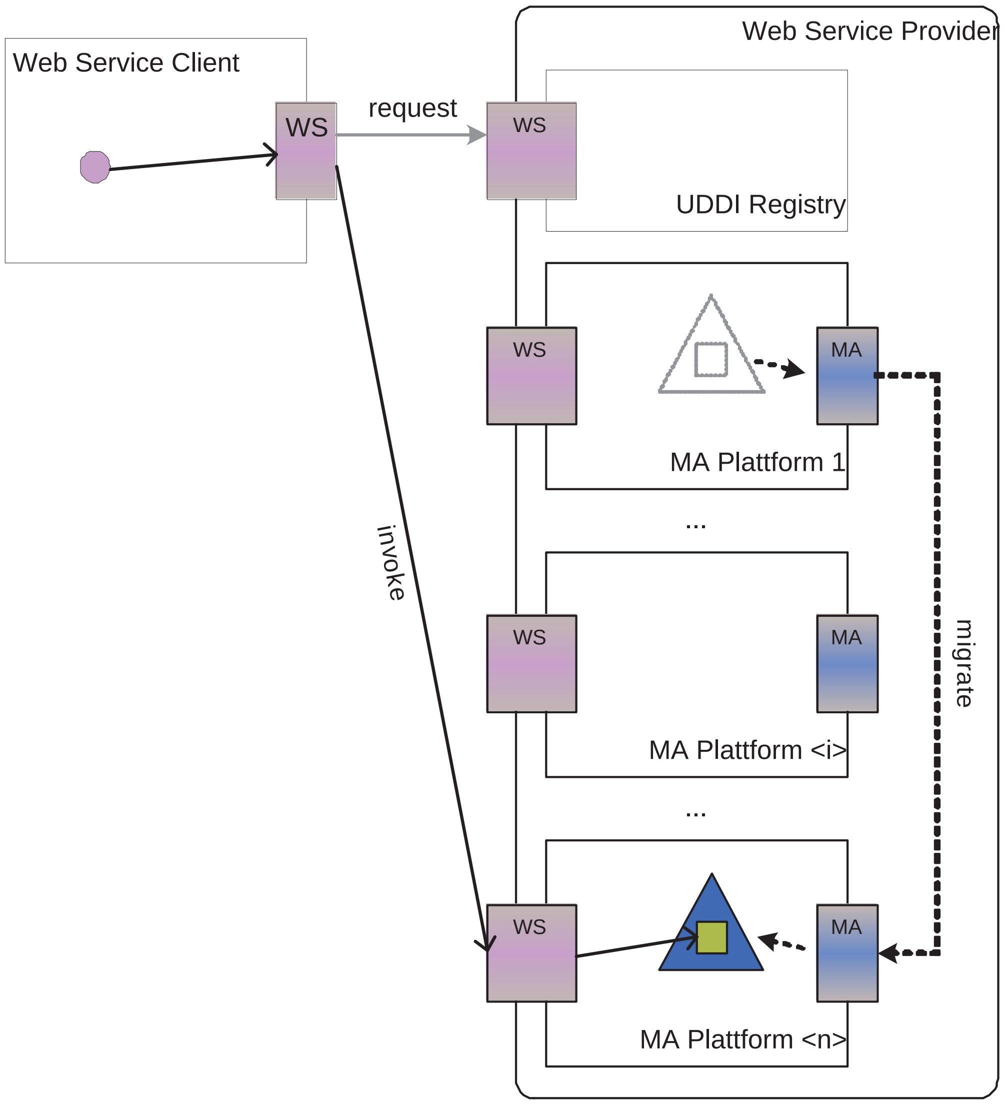


Figure 2: Load balancing of stateful web services can be provided by mobile agents embedding service functionality, dynamically distributed among agent platforms by means of migration. While the agent platforms transparently provide these services as web services, and inform a UDDI registry about the current location of a web service, a client is able to invoke the web service as usual, after having requested current service information from the registry.

developer, when combining (mobile) agents and web services.

## 2 Related Work

The work described in this report has been strongly influenced by resp. is evolved from several existing contributions to the field. In this section we want to give an overview about related work covering general agent interoperability, the enhancement of existing web service frameworks, the integration of static software agents and web services, and finally, existing papers about the integration of web services and mobile agents.

## 2.1 (Mobile) Agent Interoperability

In the area of agent interoperability only a small number of efforts have taken place so far. A well known approach for mobile agents is the MASIF proposal [31] by Milojicic et al. which suggests a standardization for an agent transport protocol. The activities of FIPA[2] in the standardization of software agent issues are rather known. FIPA's work focuses on a high level of abstraction, e.g. communication protocols, ontologies, etc., whereas specifically the FIPA standardization on agent communication issues [15] found broad attention.

Because of the variety of agent systems, another non standard-based approach to mobile agent interoperability describes a formal way to analyze the lifecycles and environments of foreign agent systems [42, 38, 37]. Foreign agents are subsequently integrated into a local agent environment by means of an appropriate wrapper for the foreign lifecycle, and by mapping the foreign environment's mechanisms (mainly service interaction, naming/tracking, migration, and communication) to the local environment's components. Thereby, *runtime interoperability* is defined as "the ability of a (mobile agent) system to start any software component of a different component-based system and act as full replacement for the component's original runtime environment". This mean of interoperability has successfully been integrated by the authors into an existing mobile agent system supporting JADE and Aglets agents resp. OSGi components.

## 2.2 Web Service Enhancements

With respect to web services there are a couple of approaches, which try to enhance partial aspects of existing web services framework as there is web service registration or client-sided web services invocation, for example.

In [46] a keyboard-controlled and browser-embedded console is presented, which allows an experienced web user to easily invoke single web services. Furthermore, a web-based IDE provides visual composition of web service calls and simple control structures to support automation without the need of software installation on the client. Combining known interaction techniques (HTML-Form based interfaces, graphical entering of workflows, code completion) with existing programming and web service technologies (JavaScript, Java, Apache AXIS) the developed framework creates a web service abstraction layer for users with little or even no programming skills.

While web services already provide distributed operation execution as well as service publication and discovery, UDDI is still based on a centralized design. In [50] the mechanisms of UDDI-based web service registries are combined with recent P2P systems offering distributed hash table functionality to dynamically process search queries in real time, within highly distributed environments. Furthermore, domain ontologies are used to annotate web services with semantic information. Thus, this approach provides a scalable mean of web service discovery without the need of a central registry and allows semantic classification based on domains.

[2]FIPA - Foundation for Intelligent Physical Agents. http://www.fipa.org/

## 2.3 Web Services and Static Agents

As listed in Section 1 there are several good reasons to integrate software agents and web services. Hence, it is not surprising that there already are a couple of approaches in this context.

Very early integration attempts are described by Moreau et al. who, step by step, ported the different mechanisms of their *Southampton Framework for Agent Research* (SOFAR) to XML-based protocols, and described their experiences [32, 3]. Since this extended agent platform was used in a grid computing scenario, the porting process was stated as one example for the convergence of three major technologies as there were software agents, web services, and grid computing.

The *Working Intelligent Simple E-commerce* (WISE) architecture wants to go half the way from current web services standards to the semantic web initiative [51]. Software agents are included on the client device and/or the web service server to add web service intelligence by means of assisting decision-makings. These agents are capable of managing context and profile data of the users resp. a provider's customers, and support the user resp. customer in selection of the right web service with respect to his current needs. Furthermore, quality of service (QoS) parameters are monitored by a specific QoS agent, either per service on the client or for a bunch of services on the server, which in turn provides valuable information for web service selection. The proposed architecture is implemented on top of J2EE web services as an extra layer within the J2EE infrastructure stack.

A mediator based solution which allows multi-agents systems to access web services is outlined in [5]. In the context of physicians, searching appropriate therapy protocols for the treatment of patients with cancerous diseases, the use of the FIPA-compliant multi-agent system JADE[3] is proposed, mapping confidential patient data to available therapy protocols collected via web services. The mediator is integrated by means of a gateway with the ability of transforming FIPA-ACL messages into corresponding SOAP messages and vice versa.

In the context of web service integration into FIPA-compliant software agent systems, [28] discusses pertinent design and usage factors. Both cases are considered, software agents as clients requesting external web services, and software agents as web service providers for non-agent clients. A J2EE-based framework is proposed which maps SOAP to FIPA-ACL messages (and vice versa), and bridges the gap between synchronous request processing on the web service side and asynchronous message processing within the agent system. Furthermore, components are suggested mapping FIPA service descriptions published within a FIPA directory facilitator to web service descriptions (WSDSL) published through UDDI registries. In either case a specific agent (a proxy agent resp. a web service agent) is created to provide external web service functionality within the agent system resp. expose agent-based services to external web service clients.

An approach to make web services available to agents in an Agentcities[4] environment is described in [14], whereas interoperability between FIPA agent service and web service environments is provided by two separate "agent to web service gateways" (one for each direction), intended for use by the JADE agent system. A reference model is defined with required and optional features. Since FIPA-ACL based inter-agent conversation is more complex and semantically richer, than simple request/response of web services, manual configuration steps are necessary to map this communication as part of the *Web Service Agent Gateway* (WSAG).

[3]JADE - Java Agent Development Framework. http://jade.tilab.com/

[4]Agentcities. http://www.agentcities.org/

However, the tool WSDL2JADE (cf. [43]) is provided to automatically generate an agent ontology as well as the agent deployment code from a given web service description. In [44] the agent-based approach of migrating web services to semantic web services using ontologies is outlined in more details.

As enhancement of the afore mentioned design ideas, [21] introduces the *Web Service Integration Gateway* (WSIGS) as gateway between FIPA-compliant agents and standard-conform web services. This architecture covers all necessary components, allowing web service invocation from an agent, agent service invocation from a web service client, as well as registry access for both web services and agents. Instead of partitioning the main functionality to distributed agents/components, this approach integrates mapping and translation of service description, service discovery and service invocation within one gateway instance. Thus, this gateway can be used in a transparent way by both technologies. Furthermore, WSIGS provides resp. enables some advanced features as there is fail-over redirection in case a service published is no longer available, active proxying to interface registered services by means of semantic service descriptions, and dynamic composition of web services with respect to a workflow.

## 2.4 Web Services and Mobile Agents

In contrast to partly sophisticated solutions of integrating web services and static software agents, there are only some first approaches of combining web services with mobile agents.

A loose integration of web services and mobile agents with respect to load balancing is described in [6]. In contrast to existing client-based, DNS-based, dispatcher-based, or server-based approaches for load balancing on distributed web servers, the authors propose a *Mobile Agent based Load Balancing Framework* (MALD). Three types of agents concurrently monitor workload on the local server, gather information from other web servers by traveling around, and execute selection policies on overloaded servers to redirect incoming jobs on the fly. The evaluation of the framework in the intranet as well as wide-area networks underlines the performance-improvement of this mobile agent-based load balancing in comparison with traditional approaches based on message passing.

Another approach suggests an agent-based multi-domain architecture [30] for service provision. The user connects to its user-domain through a web portal, and thereby accesses locally offered composite services. These services can be based on a set of primitive ones, which are associated with one or more provider-domains. To fulfill a composite service request, a local delegate agent is created together with a mobile user agent. While the delegate agent coordinates service selection, execution, and result composition, the user agent is interacting with provider agents within the provider domains, which offer web service functionality as primitive services. The user agent is instructed by the delegate agent, and subsequently executes primitive services either by using remote inter-agent communication, or by migrating to the provider-domain and then interacting with the provider agent locally. Thereby, composition and execution of services can be carried out concurrently. In [29] this concept is extended to physically mobile environments to provide mobile services (*M-services*), whereas mobile users are able to start agents on their current mobile device, which is bound to a service infrastructure via a wireless connection.

Similar work in the context of dynamic binding to specific web service instances is provided in [36]. Based on the assumption that composite web services entail the invocation of services from different service providers which in turn may invoke further web services, service dependencies can be presented in an *invocation tree*. The authors state, that a requesting client should have the possibility to set up constraints limiting the service execution time or the depth of the invocation tree. A RPC-based model, a plain mobile agent based model and a model using circulating mobile agents for service execution are compared with respect to availability, reliability and execution time.

Cooney and Roe state having chosen the simplest approach possible to integrate web services and mobile agents by an alternative that involves only few extra concepts [11]. Their prototype implementation uses the .NET framework for provision, description and invocation of static web services, with some mobile agent technology extension: A framework that provides a process migration service for .NET is reused to enable code mobility. Inter-agent communication is available but limited to the invocation of web services, whereas local web service invocation is optimized. Thereby, the caller as well as the callee cannot tell, if it has invoked resp. has been invoked by a legacy web service or mobile agent implementation. Instead of a one-to-one mapping between services and mobile agents, a new mobile agent instance is created for each incoming web service request. A drawback of this decision is the fact that there is no mean of service revocation (migration of a service agent causes its definition to spread among the hosts), and that it is not possible to create stateful web services (this would further require the support of multiple threads).

Another approach of web service and mobile agent synthesis describes a *Mobile Web Services* (MWS) by interaction protocols, internal behaviors, and migration behaviors [23]. In the current prototype a MWS is implemented as fix Bee-gent[5] agent which interprets a BPEL[6]-based process description for interaction, the Java-based service components, and a XML-based description of migration policies. The agent platform creates interpreter agents upon external web service requests for registered composite services, and supports basic web service functionality allowing these services to access external web services, in turn. Each migration rule annotates a migration behavior with an execution block, whereas real migration as well as cloning of single execution blocks is supported.

## 3 Architectural Model

Interoperability between web services and mobile agents will be enabled through a component named web service engine. The web service engine, containing all necessary modules to integrate mobile agents and web services in an automated and transparent way as defined in Section 1.3, bidirectionally maps between both technologies (cf. Figure 3). In addition to the functionality provided by an existent mobile agent gateway, agents (△) are enabled to request and interact with external web services (□), and external web service clients (○) are enabled to request and invoke services encapsulated by agents.

The Java-based engine internally uses and extends the AXIS[7] framework to dynamically create SOAP processors independent of the underlying transport protocol. Our prototype is integrated into the security-centric mobile agent system SeMoA[8]. Nevertheless, the web service engine

[5]Bee-gent - Bonding and Encapsulation Enhancement Agent. http://www.toshiba.co.jp/beegent/
[6]BPEL - Business Process Execution Language for Web Services. http://www.oasis-open.org/committees/wsbpel
[7]AXIS - Apache Extensible Interaction System. http://ws.apache.org/axis/
[8]SeMoA - Secure Mobile Agents. http://www.semoa.org

can easily be integrated into all Java-based (mobile) agent or component systems, and connect to all web service frameworks, which satisfy the assumptions listed in the following section.

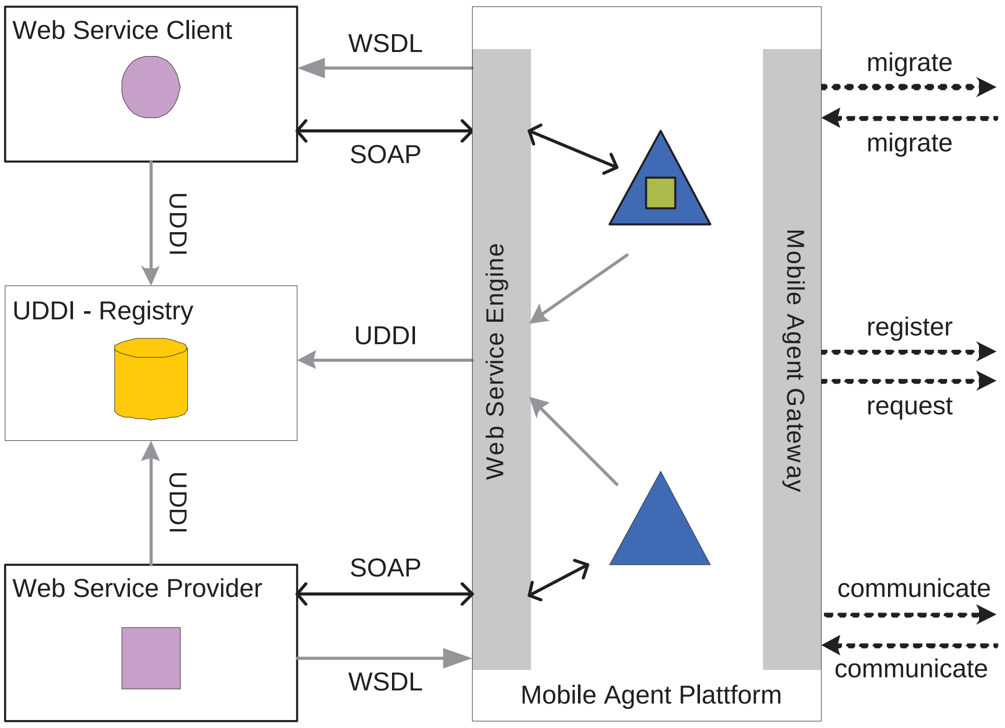


Figure 3: WSE Integration Architecture

## 3.1 Assumptions

Derived from [21], the following assumptions were made when designing the web service engine:

- The agent runtime environment provides mechanisms for service description, invocation, and discovery, which are based on direct (or remote) method invocation in Java resp. sharing of Java-APIs as service interfaces.
- All web services are assumed to use the standard web service stack consisting of WSDL for service description, SOAP for service invocation, and UDDI for service discovery.
- The web service engine is transparently integrated into the agent server by means of an extended service management layer, which *implicitly* transforms locally registered services into web services accessible from external web service clients (and vice versa).
- The web service engine is further visible as service within the agent server, which can *explicitly* be used by other services and agent (to search for and invoke resp. to register web services).
- The web service engine is visible from web service clients as gateway supporting various transport protocols (mainly HTTP and HTTPS). Service endpoints within this gateway are dynamically generated and publish via UDDI.

- If the agent system supports semantic agent service descriptions, these are available to the web service engine.

## 3.2 Architecture

Our proposed solution to implement the *Web Service Engine* (WSE) as specified in this report, covers several individual components and a service management layer which can easily be adapted and plugged into the existing service management of the specific mobile agent platform.

**Stub Generator** SOAP messages transported through the network as result of a web service invocation, are typically exchanged between a client and a server stub. With respect to the WSE architecture, the server stub processes incoming messages and triggers the associated service object by means of direct method invocation. Vice versa, the client implements the service's interface and starts communication with the associated server stub, upon local method invocation. Thereby, the task of the *stub generator* is twofold. On the server side, it extracts the specific Java interface from a given service object, automatically generates a corresponding syntactic WSDL description and creates a new server stub, which is then associated with the service object. On the client side, it transforms a given WSDL description into the corresponding Java interface, and creates a client stub implementing this interface. To realize automated and transparent integration for Java-based systems, the stub generator must be able to dynamically generate new stub objects during runtime.

**Web Service Gateway** Server stubs created by the stub generator have to be exposed by means of web services endpoints accessible over the network. The *web service gateway* thereby implements the specific transport protocols and serves as both, as web server enabling access to server stubs (e.g. over HTTP and HTTPS) as well as web client used by client stubs as transport layer for the transmission of SOAP messages.

**Registry Service** To make agent services visible by means of web service discovery, the *registry service* transforms WSDL descriptions created by the stub generator from a given service object into appropriate UDDI business entities. These business entities can subsequently be registered at a UDDI-compliant registry. Furthermore, this service can be used to search for a web service which is syntactically compatible to a given Java interface. Thus, this service implements passive mapping as defined in Section 1.3.

**WSE Service** The *WSE Service* wraps the above described functionality and provides it via a simple interface which can *explicitly* be used by mobile agents to either search for web services, or to deploy and undeploy encapsulated service objects. In both cases, the agent does not need to know anything about the traditional web service stack: deployment is done by giving a Java object implementing a certain Java interface which is automatically exposed by means of a web service, then; a search request with a given Java interface directly returns the reference to a client stub implementing this interface, if successful.

**Service Management Layer** The *service management layer* transparently activates the above described processes by automatically forwarding appropriate requests (to register or lookup an agent service within the agent infrastructure) to the WSE service. Since web service deployment and undeployment is subsequently done implicitly, the administrator of the local agent server can configure this layer in advance, and select the types of services to automatically expose as web services.

In case of SeMoA (see 4.1), the only mean of interaction between mobile agents and/or services is based on direct method invocation through shared Java interfaces: Services are registered locally as entities within a hierarchical namespace (the service *environment*). Thus, a service requester searches for an appropriate service interface within a defined sub hierarchy of the environment. If successful, a Java object is returned which implements the given interface. Figures 4 and 5 show the corresponding extended processes for (web) service deployment and undeployment resp. (web) service discovery, in case WSE has been embedded into SeMoA.

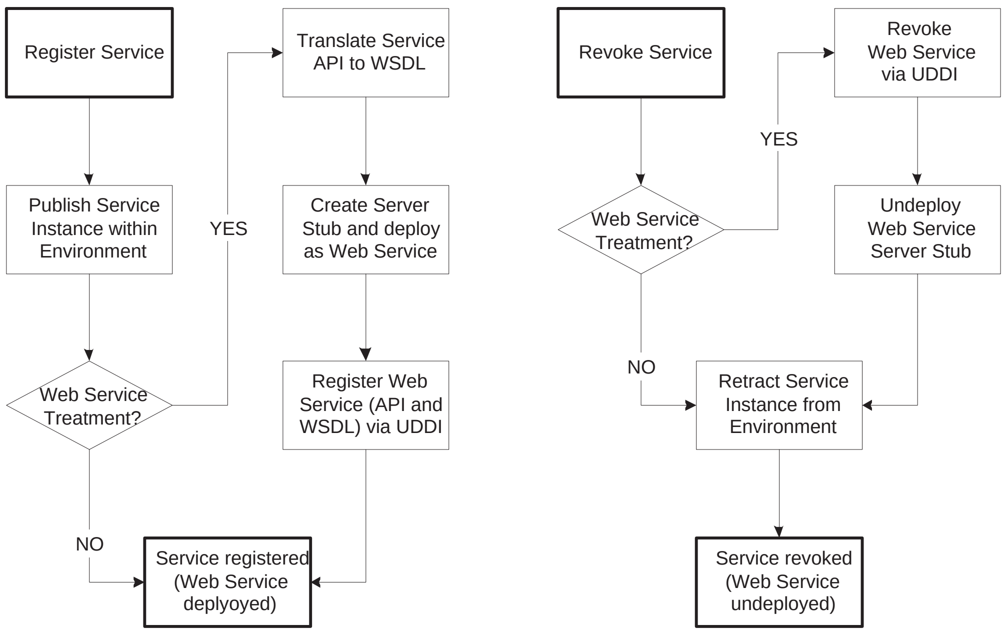

Register Service
Translate Service API to WSDL
Publish Service Instance within Environment
YES
Create Server Stub and deploy as Web Service
Web Service Treatment?
Register Web Service (API and WSDL) via UDDI
NO
Service registered (Web Service deplyoyed)
Revoke Service
Revoke Web Service via UDDI
YES
Web Service Treatment?
Undeploy Web Service Server Stub
NO
Retract Service Instance from Environment
Service revoked (Web Service undeployed)

Figure 4: Deployment/Undeployment of a (web) service

## 4 System Integration

In this section, we first introduce the core systems used for synthesis of mobile agents and web services, and then outline the required modifications to integrate both implementations.

### 4.1 SeMoA

*Secure Mobile Agents* (SeMoA) [40] is a Java-based open source framework for mobile agents with a special focus on all aspects of mobile agent security, including protection of mobile agents against malicious hosts as well as protection of hosts against malicious agents (cf. Section 5.1).

Within a SeMoA infrastructure mobile agents are able to migrate from one agent server to another, access locally published services, and communicate with other agents. *Mobile agents*

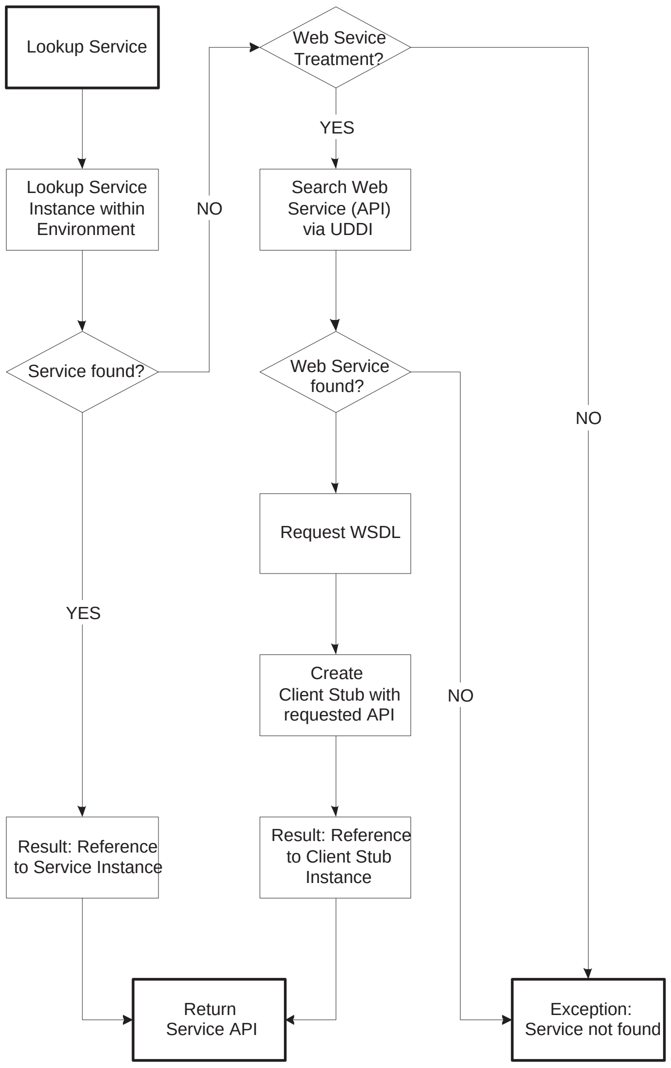

Lookup Service
Web Sevice Treatment?
YES
Lookup Service Instance within Environment
NO
Search Web Service (API) via UDDI
Service found?
Web Service found?
NO
Request WSDL
YES
Create Client Stub with requested API
NO
Result: Reference to Service Instance
Result: Reference to Client Stub Instance
Return Service API
Exception: Service not found

Figure 5: (Web) service discovery

consist of program code, data (initial properties and collected information), and meta information. They are launched within an agent server and subsequently process tasks on behalf of their owner. Basic and extended features of an agent server are provided by means of *services*, whereas a service is a Java module with a specific functionality, accessible through a well defined Java interface. Services can either be published by the local administrator of the agent server or by mobile agents as far as they have the proper security permissions. The *environment* is responsible for sevice publishment and discovery, and the only mean of local interaction between sevices and/or agents. Although agents and services encapsulated by agents are handled different than services published by the platform administrator with respect to access control and other security related issues, the environment as basic mean of interaction is shared by all agents and services on a local SeMoA platform.

The SeMoA platform (the agent server) has a layered structure (see Figure 6). The kernel layer contains a command shell for bootstrapping and administation purpose, inherent security mechanisms, and the above mentioned environment for service management. On top of the

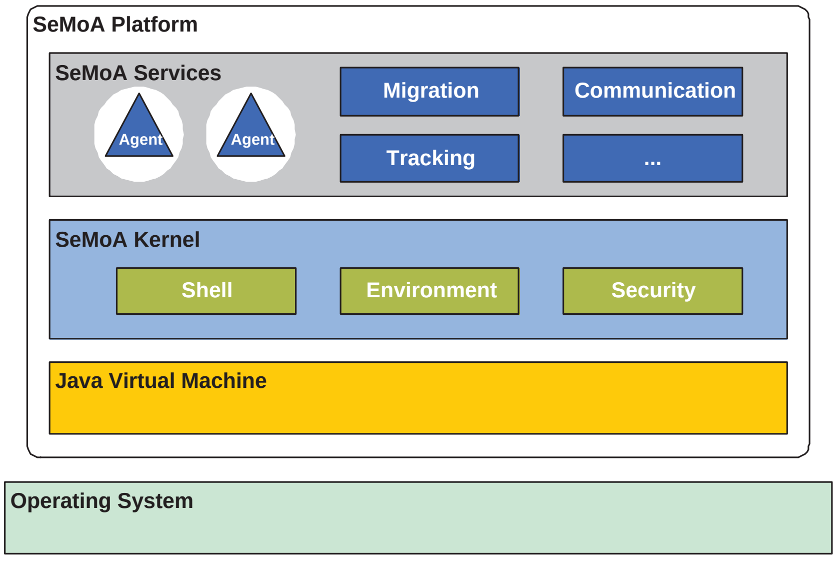


Figure 6: The SeMoA platform consists of a kernel and a service layer.

kernel, services and encapsulated mobile agents are executed within the runtime environment of the service layer. Some essential services, as there are services for agent migration, agent tracking and agent communication for example, are already installed and published during the bootstrapping phase of the agent server. Optionally, a variety of additional services included in the SeMoA framework can be loaded dependent on the desired characteristic of the local platform. In the context of web service integration, the available web server implementation which supports Java sevlets is very usefull.

Since the SeMoA environment (see Figure 7) is the main component for local service management, this module is described more detailed in the following: The environment is a hierarchically structured namespace wherein service instances can be published under a given path. This is analog to a file system, where paths lead to files - with the exception that paths are virtual in the SeMoA environment, i.e. the paths of all published services build the hierarchy and not vice versa. Subsequently, this anomaly allows the coexistence of a path and an equal service name within the same "directory". Again analog to a file system, the access to sub hierarchies or service objects can be restricted by means of access rights, as there is a specific permission for read access (*lookup*), and two permissions for write access (*publish*, *retract*). Since the environment is the only way of interaction between service and service, kernel and service, or application and service, this is an essential location for policy enforcement.

Besides being a kind of service directory, another important feature of the environment is the encapsulation resp. separation of services. When publishing a service the administrator can specify a so-called *dynamic proxy*, which is used for service interaction, subsequently. The following two types of proxies are specified for the SeMoA platform:

**Plain proxy** This type forwards method invokations to the encapsulated service object within the same thread. Initiated by a privileged signal, the proxy truncates the reference to the encapsulated object and releases it for garbage collection, as there are no more strong references to the object. Further method invokation leads to a corresponding exception thrown by the proxy. Thereby, access to the service can be prevented, even if another

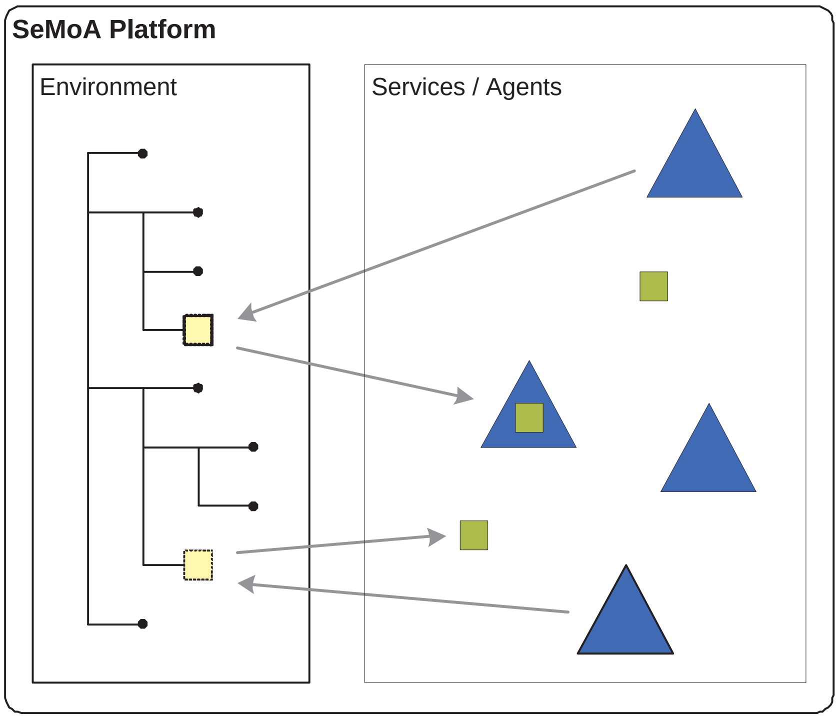


Figure 7: Local services are published and accessed through the *Environment* in SeMoA.

service has requested access to the object and still keeps the according reference, since the reference refers to the proxy not the encapsulated service object. Nevertheless, the proxy is not able to appropriately treat references returned by methods of the encapsulated object. Otherwise mechanisms were needed analog to active firewalling.

**Asynchronous proxy** This type initiates a distinct thread to serve method invokations. The caller thead is blocked till the result of the invoked method is available. The advantage is, that the caller thread may set a timeout, after whose expiry the caller thread can terminate, even if the method result is not available so far.

In both cases only public methods of all implemented interfaces of the published service object are wrapped by the proxy. These mechanisms reduce DoS attacks by possibly malicious or faulty applications and services. Further the encapsulation allows clean security context switches between caller and callee. Another security feature enables the return of a local view of the global environment to each single service. This local view of the environment automatically remembers all objects published by this service, and then retracts all those object, when the service is terminated.

## 4.2 AXIS

Axis[9] is the third generation of Apache SOAP (which began at IBM as "SOAP4J"), and thereby represents a framework for constructing SOAP processors such as clients, servers and gateways,

[9]AXIS - Apache Extensible Interaction System. http://ws.apache.org/axis/

independent from the underlying transport protocol. Furthermore, Axis supports the Web Service Description Language (WSDL), which allows you to easily build stubs to access remote services, and also to automatically export machine-readable descriptions of your deployed services from Axis.

The default deployment process of a web service in Axis requires manual steps: Emanating from a service, implementing a certain Java API, the XML-based web Service Deployment Descriptor (WSDD) has to be created. Together with the service implementation this deployment descriptor is used to subsequently deployed the service by the Axis engine, which generates the web service server stub as well as the web service description. Access to both, the server stub and the WSDL description is realized trough the Axis servlet. On the client side the WSDL description can be used to generate the corresponding client stub. Finally, the client and server stub are able to communicate through SOAP (e.g. Java-RPC) over a defined transport protocol, which subsequently enables the client to invoke functionality of the service.

## 4.3 Integration of SeMoA and AXIS

We make use of the Axis open source framework, because it can easily be plugged into SeMoA's servlet engine and because of its flexible configuration and extensibility (especially with respect to web service security) [1]. Conform to SeMoA's local service management and through an implemented Axis wrapper managing WSDD creation from a given Java-API, the Axis framework is used to automatically process the above described deployment steps. In other words, a service published in the SeMoA environment is then accessible as web service through SeMoA's web server by a remote client. This is done in a transparent way for the service or agent developer, presupposing that the developer has either set a certain *web service flag* when publishing the service, or published the service in a certain *web service region* of the environment. This extension of SeMoA's service management further includes a UDDI service enabling service discovery in a distributed infrastructure which is compatible to the web service specification (see Figure 8).

Axis supports scoping service objects (the actual Java objects which implement the service functionality) three ways: *Request* scope will create a new object each time a SOAP request comes in for the service, *application* scope will create a singleton shared object to serve all requests, and *session* scope will create a new object for each session-enabled client who accesses the service. Since our goal is to enable state-full mobile web services and because of an implementation flaw of AXIS (service object are internally cached within the Axis engine, when using the application scope, and thereby reused after an un-deployment/deployment process), we implemented our own deployment provider, which ensures that an external service request is forwarded to the service implementation, which is referenced within the SeMoA environment.

Furthermore, we improve the process of server and client stub generation which is supported by Axis only in a manual half-automated way (see regular processes in Figure 9). As consequence, a service can directly be deployed as web service on server side without the need of manually generating a deployment descriptor or accessing files in the local file system. On the client side, our framework contains dynamic and fully automated generation of Java instances out of a given web service description, again without the need of manual steps or the creation/access of files in the local filesystem (see our framework in Figure 9).

Another advantage of our framework is an improvement of the general development process, when handling web service: In a regular process, a service is first implemented and then deployed a web service on server side, before the description is requested and subsequently the

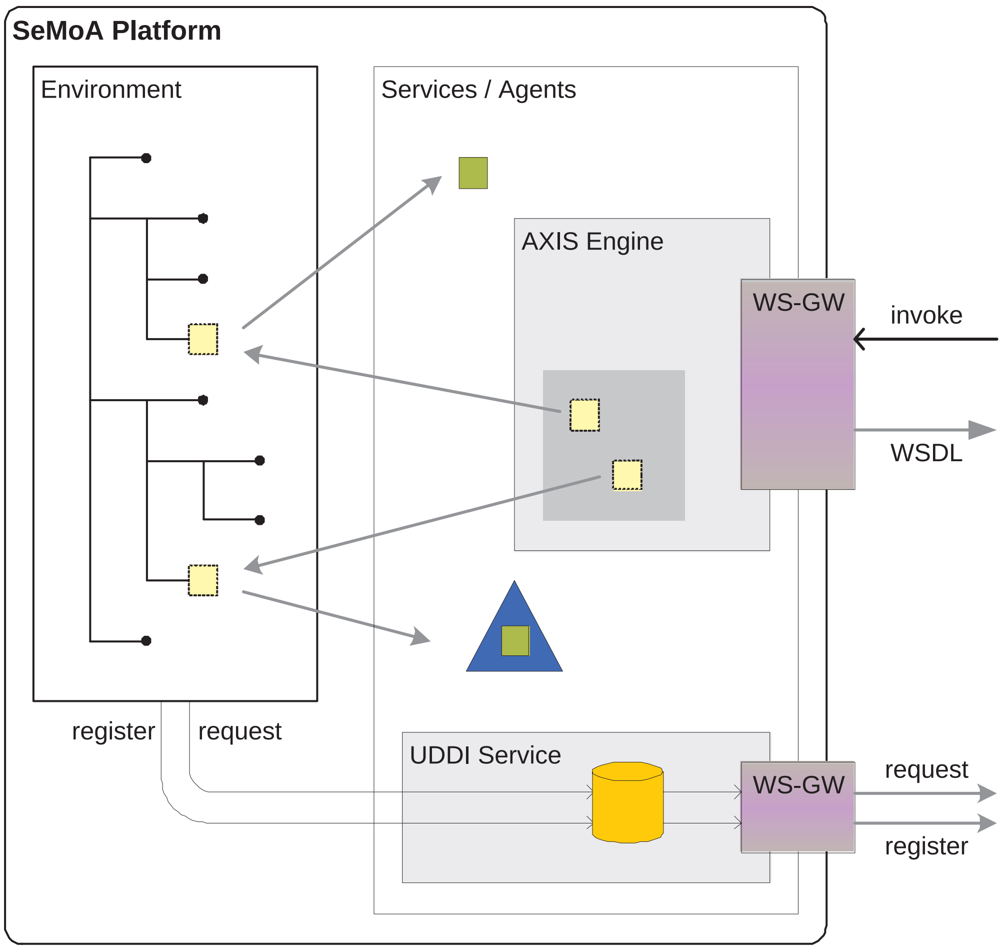


Figure 8: Service management extensions needed for web service interoperability in SeMoA.

corresponding stub is generated and compiled on the client side. Within our framework, only the Java interface has to be specified and shared between the developers. Subsequently, the service and a its client can be implemented in parallel (see Figure 10).

## 5 Security Aspects

Both web service and mobile agent technology have different attack scenarios and thus ask for different solutions. The main concern in web service security is to prohibit unauthorized access to resources from (malicious) clients. Mobile agent security also cares about malicious agents and malicious hosts. In our approach we combine security mechanism of SeMoA with security mechanisms from web service technology.

First the security features of SeMoA are described. After that, we introduce the current security mechanism offered by web services technology. In the third part of this section we present our approach on how to combine both security means.

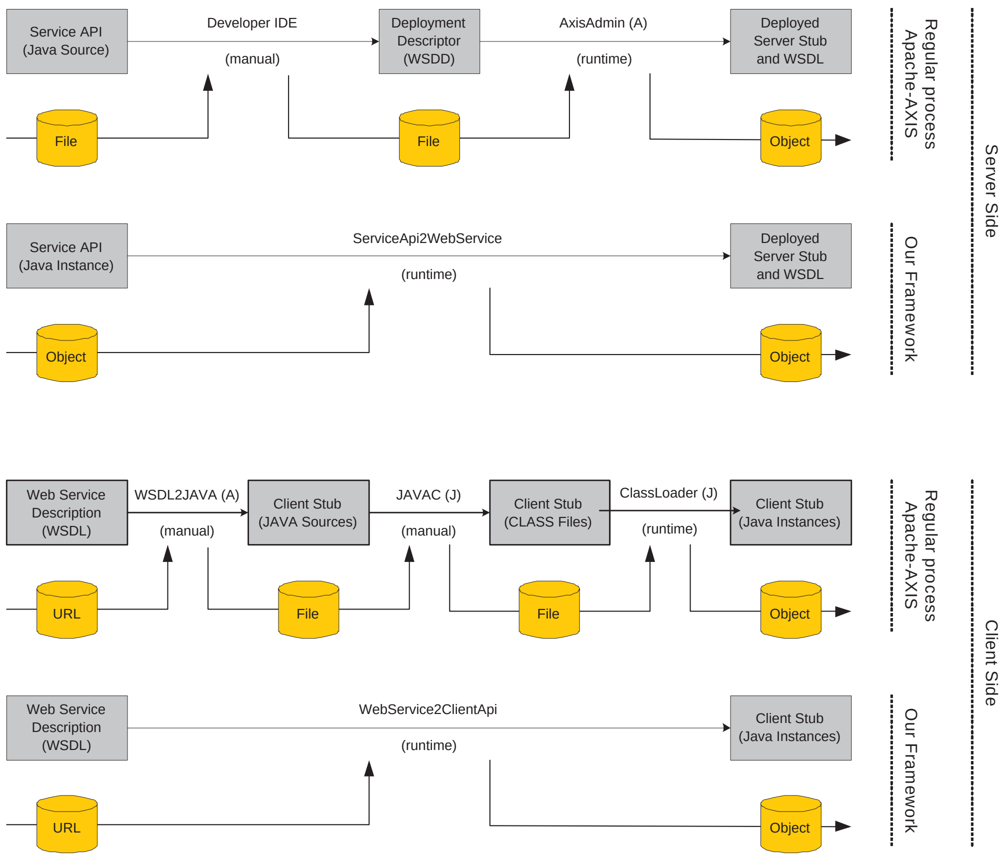

Service API (Java Source)
Developer IDE
(manual)
Deployment Descriptor (WSDD)
AxisAdmin (A)
(runtime)
Deployed Server Stub and WSDL
File
File
Object
Regular process Apache-AXIS
Service API (Java Instance)
ServiceApi2WebService
(runtime)
Deployed Server Stub and WSDL
Object
Object
Our Framework
Server Side
Web Service Description (WSDL)
WSDL2JAVA (A)
(manual)
Client Stub (JAVA Sources)
JAVAC (J)
(manual)
Client Stub (CLASS Files)
ClassLoader (J)
(runtime)
Client Stub (Java Instances)
URL
File
File
Object
Regular process Apache-AXIS
Web Service Description (WSDL)
WebService2ClientApi
(runtime)
Client Stub (Java Instances)
URL
Object
Our Framework
Client Side

Figure 9: This figure represent the regular processes of Apache-AXIS as well as the improved ones within our framework on client and server side, which are necessary to deploy and use a Java service as web service.

## 5.1 SeMoA Security Mechanisms

A mobile agent platform constitutes the runtime environment for the agents and further provides the middleware for access to local resources. It is quite clear that this platform should follow a security aware design concept. The SeMoA project developes an open server for mobile agents with a special focus on all aspects of mobile agent security, including protection of mobile agents against malicious hosts as well as protection of hosts against malicious agents [40].

The security architecture of the SeMoA server compares to an onion: agents have to pass all of several layers of protection before they are admitted to the runtime system (see Figure 11) and the first class of an agent is loaded into the server's Java Virtual Machine. The first (outer) security layer is a transport layer security protocol such as TLS or SSL. This layer provides mutual authentication of agent servers, transparent encryption and integrity protection. Connection requests of authenticated peers can be accepted or rejected as specified in a configurable policy. The second layer consists of a pipeline of security filters. Separate pipelines for incoming agents and outgoing agents are supported. Each filter inspects and processes incoming/outgoing agents, and either accepts or rejects them. Subsequent to passing all security filters, SeMoA sets up a sandbox for the accepted agent (which can be regarded as layers three and four). Each agent gets a separate thread group and class loader. This class loader supports loading classes that

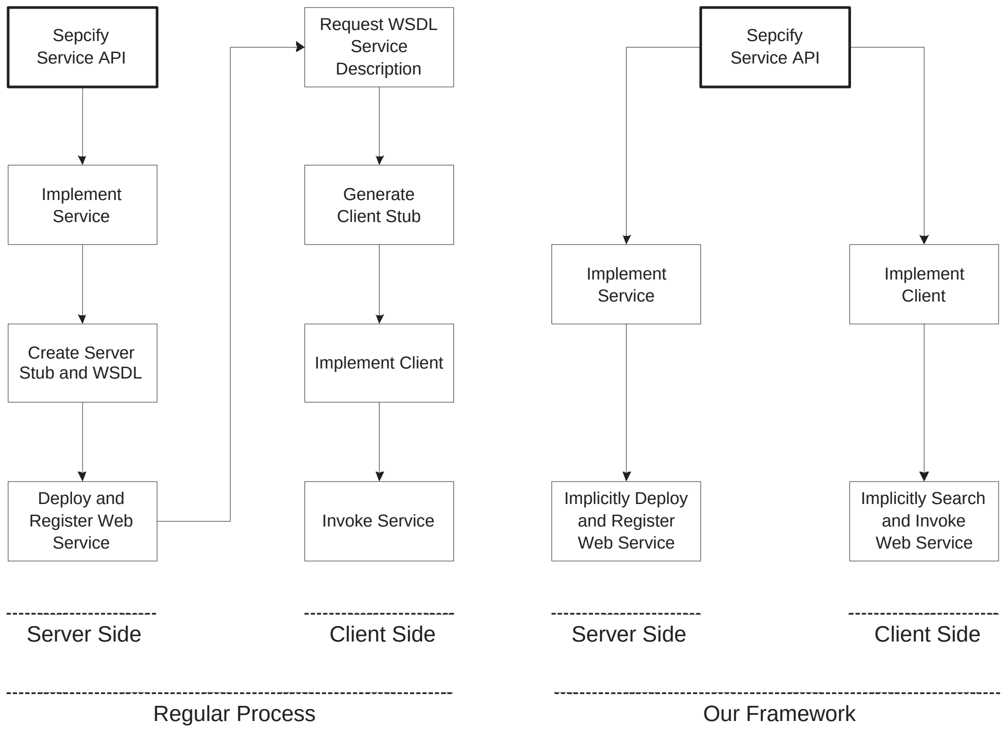


Figure 10: In contrast to the sequential development process when using the regular Apache-Axis framework, our framework allows parallel development on client and server side.

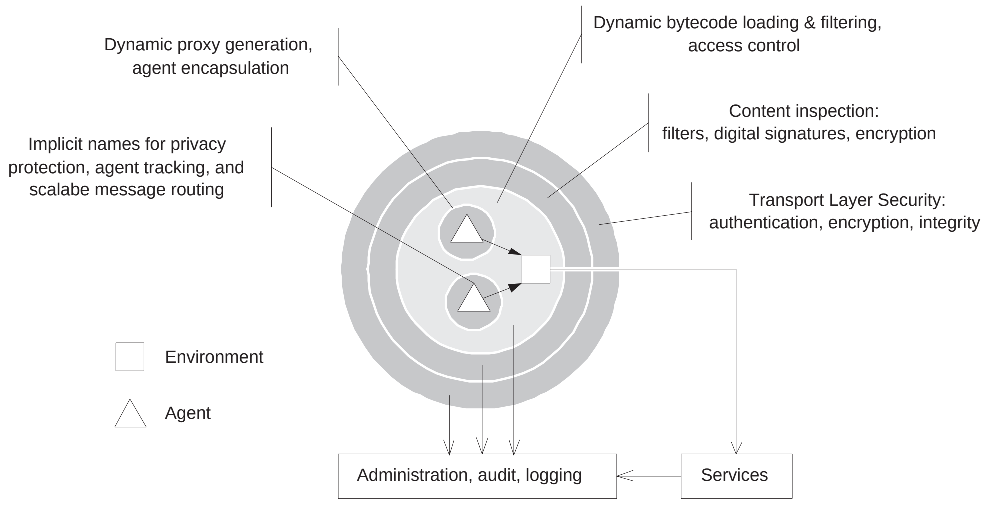

Dynamic proxy generation, agent encapsulation
Dynamic bytecode loading & filtering, access control
Content inspection: filters, digital signatures, encryption
Implicit names for privacy protection, agent tracking, and scalabe message routing
Transport Layer Security: authentication, encryption, integrity
Environment
Agent
Administration, audit, logging
Services

Figure 11: Incoming and outgoing agents have to pass several security layers.

came bundled with the agent, as well as loading classes from remoted code sources specified in the agent. Agents cannot share classes so one agent cannot not load a Trojan Horse class into the name space of any other agent. Agents are separated from all other agents in the system; no references to agent instances are published by default. The only means to share instances between agents is to publish them in a global environment. Each agent gets its own view on this global environment, which tracks the instances registered by that agent. All published objects are wrapped into proxys which are created dynamically. The agent identifier is generated from its signed static part (i.e. program code and initial parameters) by means of a cryptographical hash of the agent owner's digital signature. Thereby, this *implicit name* is globally unique with respect to cryptographical digests, and an agent cannot impersonate another identity. This identifier thus enables privacy protection, agent tracking for the owner, and scalable message routing [41] as features of layer five.

An agent which has passed all security layers, has successfully been inspected (program code and task context), and identified by means of digital signatures of the owner and the senders (constituting the itinerary of an agent). These informations are used to dynamically assign a specific role to the agent. In turn, a role corresponds to a certain set of permissions which enforce the local role-based security policy. This mean of access control is implicitly used for all resources as there are the services published within the platform's environment as well as basic Java interfaces to the network or other data sources located on the local host.

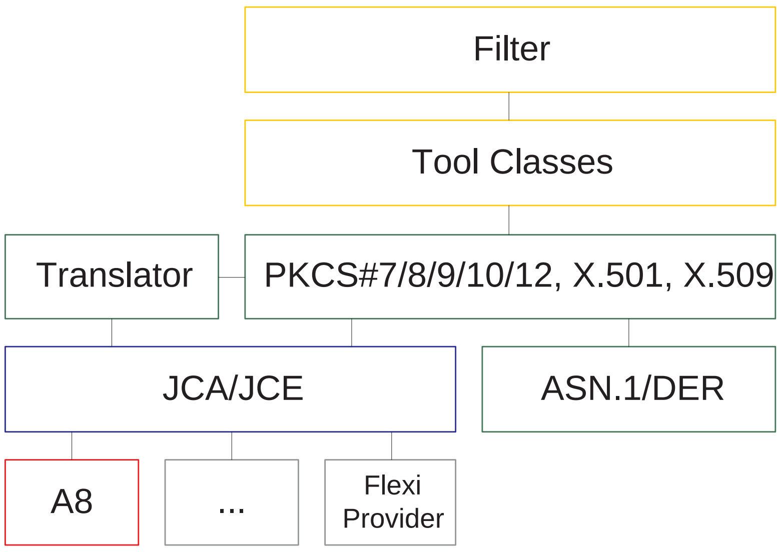


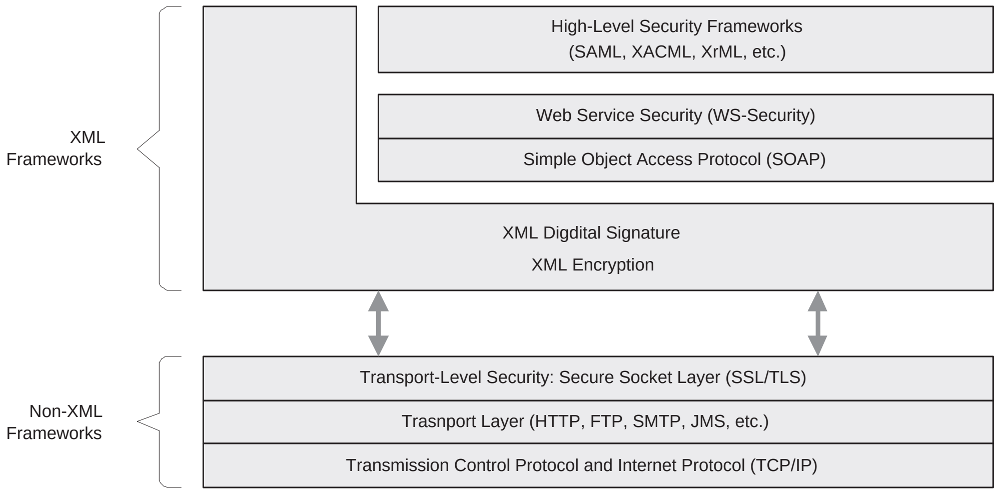
Figure 12: The crypto layers, SeMoA's security framework is based on.

Finally, administration during runtime allows the adjustment of current security filters and policies, and audit together with logging enables traceability of processes. To implement these security mechanisms, SeMoA supports a lot of (security) standards (see 12.

## 5.2 Security of Web Services

The current use of web services to share information and services across organisations make necessary a new mean of access control. Figure 13 gives an overview of existing web service security mechanism, which are described in the following:

**Transport Layer Security (SSL/TLS):** SSL/TLS [20] offer a secure channel on the transport layer. The server is authenticated based on certificates, the client is authenticated via



High-Level Security Frameworks (SAML, XACML, XrML, etc.)
Web Service Security (WS-Security)
Simple Object Access Protocol (SOAP)
XML Digdital Signature
XML Encryption
XML Frameworks
Transport-Level Security: Secure Socket Layer (SSL/TLS)
Trasnport Layer (HTTP, FTP, SMTP, JMS, etc.)
Transmission Control Protocol and Internet Protocol (TCP/IP)
Non-XML Frameworks

Figure 13: Web service security framework [7]

password or certificate. Nevertheless SSL/TLS neither offer application level security nor non-repudiation.

**Web Services Security (WS-Security):** If we assume malicious hosts, transport level security is possibly not enough. The transport level is controlled by the host, so the web service itself cannot verify the user credentials. The industry standard WS-Security [2] offers application level security as an extension to SOAP. It defines how to integrate various XML Security concepts as XML Signature [4], XML Encryption [22] or the Security Assertion Meta Language (SAML) [34] into SOAP.

The *XML Encryption norm* defines a syntax for selective encryption of XML documents.

The *XML Signature norm* defines a syntax for digital signatures that are embedded in XML documents. The signed data can be contained in the same XML structure or in an external document, even in an non-XML document. XML signature ensures that a message is not modified during transport. Furthermore the signer of the message cannot repudiate having sent the message.

**Security Assertion Meta Language (SAML):** SAML defines a XML-based framework for creating and exchanging authentication and authorization information. The standard purpose of using SAML is to realize Web Single-Sign-On. The user authenticates at the first site, retrieves an authentication and authorization token and subsequently uses this token to access further services without the the need of re-authentication.

**XML Access Control Specifications:** The XML Access Control Markup Language (XACML) [35] is an extension to SAML that focuses on access control rights. XACML defines how to express access policies. Furthermore it specifies a request/response protocol between a policy decision and a policy enforcement point.

The XML Rights Markup Language (XrML) [13] has a similar scope as XACML. The difference to XACML is its focus on digital rights management.

At the latest when combining basic web services to composite web services with added-value, the simple interaction paradigm between an explcit client and one service provider as server becomes obsolete. Rather, a dynamic tree models temporary bidirectional requestor/provider associations as part of a dependency hierarchy, established by a client at the root of the tree (compare [36]). Since a requestor then accesses a remote resource through a provider's web service on behalf of another entity (the client), authentication and authorization mechanisms have to be established, which support this kind of delegation principle, and concurrently minimize the overhead for the requesting client.

In this context, [27] compares current authentication and authorization infrastructures, while [10] discusses and proposes a new framework for web service access control based on the above presented standards which decouples authentication and authorization, and further takes dynamic aspects (as there is the request context by means of an access history) into account.

In our opinion, appropriate standard-based security mechanisms for web services have to be selected and combined according to specified security requirements. Especially in the context of integrating mobile agents and web services, the existing security frameworks for both technologies have to be thoroughly combined without enabling new side-channel attacks to the integrated system.

### 5.3 Security Architecture

We define our web service gateway as resource, so the host can decide who can use the web service interface for his agents.

The web server decides whether an agent may use the web service as a resource. This right given, the agent is responsible for deciding on the access to its services. An agent owner can either make an agent publicly available or restrict access to a dedicated user community. If the agent owner wants to restrict the access, an LDAP-based directory is used to check identities. The LDAP directory contains the user community of the host and is therefore chosen by the host.

To authenticate users, both SSL/TLS and WS-Security can be used, whereas in case of a malicious host, neither the use of transport layer security nor the use of web service layer security can assure correct results.

Figure 14 shows our approach to secure the agents. To realize authentication we chose SSL/TLS because of its easy integration into Axis. The credentials of a web service consumer are optionally checked against a LDAP user directory.

At the moment we don't have access control mechanisms integrated. We plan to integrate access control on an agent based granularity. We are planning to integrate XACML for this purpose.

## 6 Discussion and Future Work

Our approach of integrating web services and mobile agents can be compared with agent interoperability described in [38] (see Section 2.1). But instead of creating an appropriate environment for foreign components within the local agent system, we extend the local agent system with a *web service engine* which allows agents to interact with regular (in this case "foreign") web service components in a transparent way. The other way round, we rather follow the proposed design paradigms by automatically presenting agent services as Web services to outside

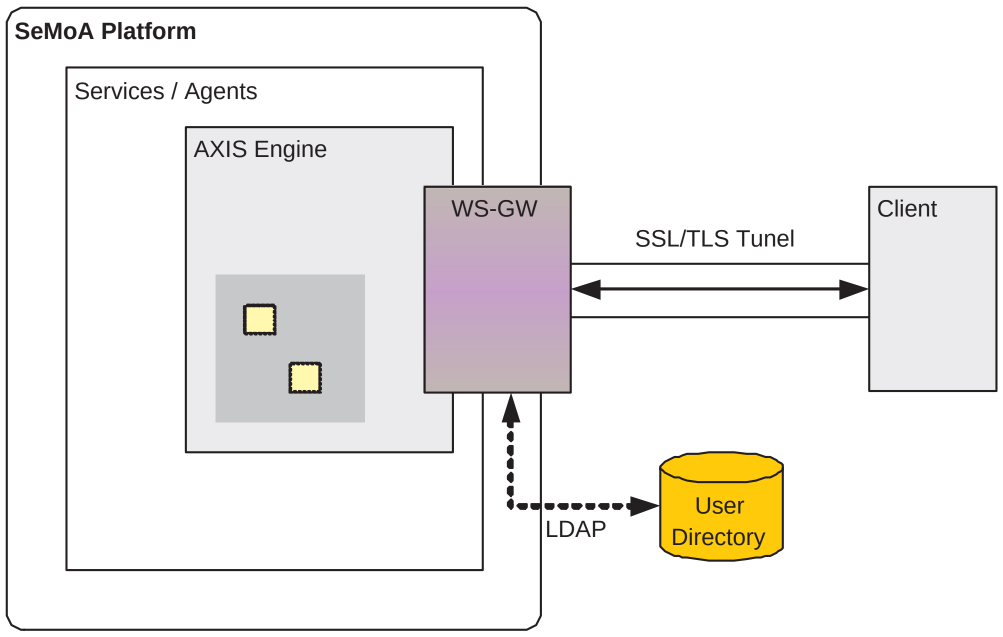


Figure 14: Web service security mechanisms

world, if needed. In this case we don't map between an agent and the agent system, but an agent and remote web service clients, which are generally located beyond the boundaries of the local agent platform.

If we want to compare our architecture with existing frameworks integrating agents and web services, we have to distinguish static agent and mobile agent systems, and additionally have to have a deeper look on the service interaction paradigm implemented by the corresponding agent system.

Most of the frameworks based on static agent systems [5, 28, 14, 43, 44, 21] (see Section 2.3) presuppose FIPA-compliant communication between agents, and a FIPA-compliant mechanism for service description and discovery (cf. [17, 16]). Since interaction is based on the exchange of messages, all the approaches lead to components (a generic gateway, or single wrapper agents) which map incoming FIPA-ACL messages and service register/deregister/search requests into messages of the corresponding web service protocol, and vice versa. These components further have to bridge the gap between asynchronous behavior of FIPA-ACL, and synchronous behavior of SOAP-based service invocation. Except the latest and most advanced framework described by Greenwood et al., which aims to implement all mechanisms for biderectional system integration in one single gateway component, the approaches need manual configuration steps for each integration process, which is partly automated but cannot be done during runtime. The framework described in [51] rather focuses on QoS monitoring for "web" services provided by client or server sided static J2EE-based agents, than on transparent integration of these two technologies.

In contrast, the existing contributions based on mobile agent systems have more diverse goals, and thereby do not completely follow transparent integration of agents and web services. This can certainly be reasoned by the lack of standards and the resulting diverseness of mobile agent systems, and further by the different application scenarios in which mobile agents are utilized in these approaches. While some approaches make use of specific advantages of mobile agents in the field of load balancing [6], or the efficient and concurrent task execution in distributed and mobile environments [30, 29], others focus on dynamic and efficient selection of web services through mobile agens [36] (compare Section 2.4). Instead of integrating web service support into an existent mobile agent system, Cooney et al. extended the .NET framework for web services with the ability of service migration [11] based on a rudimentary agent mobility paradigm. Since they mainly created a new service platform from the scratch, there was barely obligation to map existing mechanisms for service description and invocation to the web service world. The realization of mobile web services described in [23] represents a high level approach of system integration. Due to an interpreter agent encapsulating web service logic, this approach is mostly independent of the underlying mobile agent system and the given interaction paradigms. Thereby, this approach does not integrate web services into the underlying agent system, but extends the underlying system with web service support on an upper layer. As consequence, existing services of the underlying system cannot be reused transparently as web services, whereas the developer of mobile web services has a new workflow based programming model to implement composite services.

Although we have given concrete application scenario examples of mobile web services in Section 1.3.2, our system architecture is not restricted to these, nor is it exclusively bound to SeMoA. Nevertheless the architecture differs from most of the above mentioned ones dealing with static agent systems, since we build upon local service interaction transparently based on the simple paradigm of direct method invocation in Java. On the one hand, we don't have to map any asynchronous protocol behaviour to the synchronous behavior of SOAP. Since we provide automated runtime integration of web services, we have to cope with optimized and dynamic web service stub provisioning, on the other hand. The above summarized frameworks dealing with mobile agent systems, either enrich the web service world with specific advantages of mobile agent technology, or bind the mobile agent world to web service technology. In contrast, our architecture aims to provide bidirectional integration of both technologies in a seamless way. Fulfilling the assumptions (see Section 3.1), our web service engine can be integrated into existing mobile agent systems, transparently connecting these with web service compliant frameworks. Another aspect which has been neglected by these approaches is the overall security of the resulting service platform. Mobile agent technology itself already enables a couple of unusual attacks [8, 25, 26, 12, 39, 45]. When opening local agent services to the web by means of web service invocation as alternative to the interaction paradigms of the agent system, this might involve new side-channel attacks to the mobile agent system.

Up to now, services in SeMoA are simply described, registered, and searched by means of a path in the local service namespace together with the implemented Java interfaces. On this level of syntactic service description, the local service registry is automatically extended by the web service engine towards distributed (web) service management. This mean of service discovery has been integrated into the agent system in a transparent way for the agent developer. But in case, regular web service clients search for appropriate service implementations, this kind of service management might not be sufficient anymore, since the framework is not able to automatically generate a semantic service description from the agent service. Nevertheless, the support of semantic service descriptions can easily be integrated into the web service engine, when either the agent platform supports these, or in case of the current SeMoA implementation provides them together with the agent. Though, the second suggestion would not comply to transparent web service integration as defined in Section 1.3, anymore.

The current web service engine makes use of existing UDDI registries. Compared to the *Domain Name System* (DNS) as naming service for fairly static objects which cannot be used to track fast migrating mobile agents (compare [41]), the existing UDDI registries might not be suitable to track mobile web services in the context of load balancing as described in Section 1.3.2. which frequently change their location. Future work will evaluate these statements, and consider and analyze more scalable solutions as proposed in [50] with respect to performance issues.

Further implementation details and especially security aspects would exceed the scope of this report. Thus, they are presented and discusses within another contribution in the context of secure mobile services.

## References

[1] Apache. Axis Architecture Guide. Technical Report Version 1.1, The Apache Software Foundation, 2004. `http://ws.apache.org/axis/java/architecture-guide.pdf`.

[2] B. Atkinson, G. Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, J. Klein, B. LaMacchia, P. Leach, J. Manferdelli, H. Maruyama, A. Nadalin, N. Nagaratnam, H. Prafullchandra, J. Shewchuk, and D. Simon. Specification: Web Service Security (WS-Security). Technical Report Versoin 1.0, IBM developerWorks, April 2002. `ftp://www6.software.ibm.com/software/developer/library/ws-secure.pdf`.

[3] A. Avila-Rosas, L. Moreau, V. Dialani, S. Miles, and X. Liu. Agents for the Grid: A comparison with Web Services (part II: Service Discovery). In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'02)*, Bologna, Italy, July 2002.

[4] Mark Bartel, John Boyer, Barb Fox, Brian LaMacchia, and Ed Simon. XML-Signature Syntax and Processing. W3C recommendation, World Wide Web Consortium (W3C), February 2002. `http://www.w3.org/TR/xmldsig-core/`.

[5] Frank Büscher. How Multi-Agent Systems and Web Services can work together. In *Net.ObjectDays 2004: NODe Young Researchers Workshop '04*, Erfurt, Germany, September 2004.

[6] J. Cao, Y. Sun, Y. Wang, and S.K. Das. Scalable Load Balacning on Distributed Web Servers Using Mobile Agents. *Journal on Parallel and Distributed Computing*, 63(10):996–1005, October 2003. ISSN:0743-7315.

[7] Marc Chaniliau. Web Services-Sicherheit und die SAML. Online article, XML and Web Services Magazin, January 2004. `http://www.entwickler.com/itr/online_artikel/psecom,id,468,nodeid,69.ht%ml`.

[8] David M. Chess. Security issues in mobile code systems. In Vigna [45], pages 1–14.

[9] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL), Version 1.1. W3C working group note, World Wide Web Consortium (W3C), March 2001. `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`.

[10] M. Coetzee and J.H.P. Eloff. Towards Web Service access control. In *Computers & Security*, volume 23, pages 559–570. Elsevier, October 2004.

[11] Dominic Cooney and Paul Roe. Mobile Agents Make for Flexible Web Services. In *Proceedings of The Ninth Australian World Wide Web Conference*, Quensland, Australia, July 2003. `http://ausweb.scu.edu.au/aw03/papers/cooney/`.

[12] A. Corradi, R. Montanari, and C. Stefanelli. Mobile agents protection in the Internet environment. In *The 23rd Annual International Computer Software and Applications Conference (COMPSAC '99)*, pages 80–85, 1999.

[13] CoverPages. Extensible Rights Markup Language (XrML). Technology reports, Coverpages, March 2003. `http://xml.coverpages.org/xrml.html`.

[14] Jonathan Dale, Akos Hajnal, Martin Kernland, and Laszlo Zsolt Varga. Integrating Web Services into Agentcities Recommendation. Technical report, Agentcities Task Force, November 2003. `http://www.agentcities.org/rec/00006/actf-rec-00006a.pdf`.

[15] FIPA. ACL message structure specication. Technical Report FIPA document XC00061E, Foundation for Intelligent Physical Agents, August 2001. `http://www.fipa.org/specs/fipa00061/`.

[16] FIPA. FIPA Agent Discovery Service Specification. Preliminry FIPA document PC00095A, Version 1.2e, Foundation for Intelligent Physical Agents, October 2003. `http://www.fipa.org/specs/fipa00095`.

[17] FIPA. FIPA Agent Management Specification. Standard FIPA document SC00023K, Foundation for Intelligent Physical Agents, March 2004. `http://www.fipa.org/specs/fipa00023`.

[18] John Fou. Web Services and Mobile Intelligent Agents - Combining Intelligence with Mobility, 2001. `http://www.webservicesarchitect.com/content/articles/fou02.asp`.

[19] Stan Franklin and Art Graesser. Is it an Agent, or just a Program? In *Intelligent Agents III*, volume 1193 of *Lecture Notes in Artificial Intelligence*, pages 21–36, Berlin, 1997. Springer Verlag.

[20] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The SSL Protocol, Version 3.0. Internet draft, Netscape, November 1996. `http://wp.netscape.com/eng/ssl3/`.

[21] Dominic Greenwood and Monique Calisti. Engineering Web Service - Agent Integration. In *IEEE International Conference on Systems, Man and Cybernetics (SMC 2004)*, The Hague, The Netherlands, October 2004.

[22] Takeshi Imamura, Blair Dillaway, and Ed Simon. XML Encryption Syntax and Processing. W3C recommendation, World Wide Web Consortium (W3C), Dezember 2002. `http://www.w3.org/TR/xmlenc-core/`.

[23] Fuyuki Ishikawa, Nobukazu Yoshioka, Yasuyuki Tahara, and Shinichi Honiden. Toward Synthesis of Web Services and Mobile Agents. In *Proceedings of the AAMAS'2004 Workshop on Web Services and Agent-based Engineering (WSABE)*, New York, USA, July 2004. `http://honiden-lab.ex.nii.ac.jp/~f-ishikawa/docs/wsabe2004/camera-ready%.pdf`.

[24] Dag Johansen. Mobile Agents: Right Concepts, Wrong Approach. In *Proceedings of the IEEE International Conference on Mobile Data Management (MDM'04)*, Berkeley, California, USA, January 2004.

[25] G. Karjoth, N. Asokan, and C. Gülcü. Protecting the computation results of free–roaming agents. In *Proceedings of the Second International Workshop on Mobile Agents (MA '98)*, pages 195–207.

[26] Neeran M. Karnik and Anand R. Tripathi. Security in the Ajanta mobile agent system. Technical Report TR-5-99, University of Minnesota, Minneapolis, MN 55455, U. S. A., May 1999.

[27] Javier Lopez, Rolf Oppliger, and Günther Pernul. Authentication and authorization infrastructures (AAIs): a comparative survey. In *Computers & Security*, volume 23, pages 578–590. Elsevier, October 2004.

[28] M. Lyell, L. Rosen, M. Casagni-Simkins, and D. Norris. On Software Agents and Web Services: Usage and Design Concepts and Issues. In *The 1st International Workshop on Web Services and Agent-based Engineering*, Sydney, Australia, July 2003.

[29] Zakaria Maamar, Wathiq Mansoor, Hamdi Yahyaoui, and Arif Bhati. Towards an Environment of Mobile Servies: Architecture and Security. In *The 2003 International Conference on Information Systems and Engineering (ISE 2003)*, Quebec, Canada, July 2003.

[30] Zakaria Maamar, Quan Z. Sheng, and Boualem Benatallah. Interleaving Web Services Composition and Execution Using Software Agents and Delegation. In *The 1st International Workshop on Web Services and Agent-based Engineering*, Sydney, Australia, July 2003.

[31] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Omo, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White. MASIF - The OMG Mobile Agent System Interoperability Facility. In K. Rothermel and F. Hohl, editors, *Proceedings of the Second International Workshop on Mobile Agents (MA '98)*, volume 1477 of *Lecture Notes in Computer Science*, pages 50–67. Springer Verlag, Berlin Heidelberg, 1998.

[32] Luc Moreau. Agents for the Grid: A Comparison with Web Services (Part I: the transport layer). In *Proceedings of the Second IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2002)*, Berlin, Germany, May 2002.

[33] OASIS. Universal Description, Discovery and Integration (UDDI), Version 3. Technical committee specification, Organization for the Advancement of Structured Information Standards (OASIS), October 2003. `http://www.uddi.org/pubs/uddi_v3.htm`.

[34] OASIS. Security Assertions Markup Language (SAML), Version 2.0. Working draft, Organization for the Advancement of Structured Information Standards (OASIS), 2004. `http://www.oasis-open.org/committees/security`.

[35] OASIS. XML Access Control Markup Language (XACML), Version 2.0. Committee draft, Organization for the Advancement of Structured Information Standards (OASIS), September 2004. `http://docs.oasis-open.org/xacml/access_control-xacml-2_0-core-spec-cd-%02.pdf`.

[36] Amir Padovitz, Shonali Krishnaswamy, and Seng Wai Loke. Towards Efficient Selection of Web Services. In *The 1st International Workshop on Web Services and Agent-based Engineering*, Sydney, Australia, July 2003.

[37] Ulrich Pinsdorf. A Formal Approach for Interoperability between Mobile Agent Systems and Component Based Architectures. In *Proceedings of 11th IEEE International Conference on the Engineering of Computer-Based Systems (ECBS 2004)*, Brno, Czech Republic, May 2004. Institute of Electrical and Electronics Engineers, IEEE Computer Society Press.

[38] Ulrich Pinsdorf and Volker Roth. Mobile Agent Interoperability Patterns and Practice. In *Proceedings of Ninth IEEE International Conference and Workshop on the Engineering*

*of Computer-Based Systems (ECBS 2002)*, Computer Graphics Edition, pages 238–244, University of Lund, Lund, Sweden, April 2002. Institute of Electrical and Electronics Engineers, IEEE Computer Society Press. ISBN 0-7695-1549-5.

[39] Volker Roth. Mutual protection of co–operating agents. In Jan Vitek and Christian Jensen, editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, volume 1603 of *Lecture Notes in Computer Science*, pages 275–285. Springer-Verlag Inc., New York, NY, USA, 1999.

[40] Volker Roth and Mehrdad Jalali. Concepts and Architecture of a Security-centric Mobile Agent Server. In *Proc. Fifth International Symposium on Autonomous Decentralized Systems (ISADS 2001)*, Dallas, Texas, U.S.A., March 2001. IEEE Computer Society Press.

[41] Volker Roth and Jan Peters. A Scalable and Secure Global Tracking Service for Mobile Agents. In *Proc. Mobile Agents 2001*, Lecture Notes in Computer Science. 5th IEEE International Conference Mobile Agents (MA), Springer Verlag, December 2001.

[42] Volker Roth, Ulrich Pinsdorf, and Walter Binder. Mobile Agent Interoperability Revisited. In Keith Marzullo, Amy L. Murphy, and Gian Pietro Picco, editors, *Mobile Agents 2001. Poster Session*, pages 5–8, Atlanta, Georgia, USA, December 2001. 5th IEEE International Conference Mobile Agents (MA), IEEE Society Press.

[43] Laszlo Z. Varga. WSDL2Agent: Tool to Help the Integration of Existing Web Services into Agent Systems and Semantic Web Service Environments. `http://sas.ilab.sztaki.hu:8080/wsdl2agent/`.

[44] Laszlo Zsolt Varga, Ãkos Hajnal, and Zsolt Werner. An Agent Based Approach for Migrating Web Services to Semantic Web Services. In *11th International Conference on Artificial Intelligence: Methodology, Systems and Applications (AIMSA 2004)*, Lecture Notes in Computer Science, pages 371–380, Varna, Bulgaria, September 2004. Springer Verlag Heidelberg.

[45] Giovanni Vigna, editor. *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin Heidelberg, 1998.

[46] Alexander Hilliger von Thile, Ingo Melzer, and Hans-Peter Steiert. Managers Don’t Code: Making Web Services Middleware Applicable for End-Users. In *European Conference on Web Services (ECOWS 2004)*, Erfurt, Germany, September 2004.

[47] W3C. Simple Object Access Protocol (SOAP), Version 1.2. W3C recommendation, World Wide Web Consortium (W3C), June 2003. `http://www.w3.org/TR/soap/`.

[48] W3C. Extensible Markup Language (XML), Version 1.0. W3C working group note, World Wide Web Consortium (W3C), February 2004. `http://www.w3.org/XML/`.

[49] W3C. Web Services Architecture. W3C working group note, World Wide Web Consortium (W3C), February 2004. `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/`.

[50] Shoujian Yu, Jianwei Liu, and Jiajin Le. Decentralized Web Service Organization Combining Semantic Web and Peer to Peer Computing. In *European Conference on Web Services (ECOWS 2004)*, Erfurt, Germany, September 2004.

[51] Soe-Tsyr Yuan and Kwei-Jay Lin. WISE - Building Simple Intelligence into Web Services. In *The 1st International Workshop on Web Services and Agent-based Engineering*, Sydney, Australia, July 2003.