

Using semantic Web services for ad hoc collaboration in virtual teams

Matthias Flügge, Kay-Uwe Schmidt

Competence Center ELAN
Fraunhofer Institute FOKUS
Kaiserin-Augusta-Alle 31
10589 Berlin
matthias.fluegge@fokus.fraunhofer.de
kay-uwe.schmidt@fokus.fraunhofer.de

Abstract: The system presented in this paper shows an alternative solution for the communication problem in virtual teams. Although it does not address the full bandwidth of tasks of an groupware system it demonstrate a semantic solution for the appointment making process in virtual teams, overcoming the drawbacks of complex groupware systems since it is relying on existing applications the user has confidence in. The introduced system is a J2EE¹ application based on the Jena² framework and enables the manipulation of the calendar data of heterogeneous Personal Information Managers (PIMs)³ in order to make appointments for all the members of a virtual team at once. The communication with the PIMs occurs over Web services based on their semantic descriptions. Additional the system provides a matchmaking facility in order to find the appropriate services.

1 Introduction

Team collaboration has emerged as a core technology for large organisations. The emergence of information and communications technologies in general and of the World Wide Web in particular helped to overcome the constraints of time and distance and lead to the virtualization of organisations and the formation of virtual teams. Following a broadly used definition a virtual team is a team which works “across space, time and organisational boundaries with links strengthened by webs of communication technologies” [LS97].

Virtual teams respectively groups of people working together are supported by software tools and technology which are commonly referred to as groupware. While there is no default scale of functionalities typical web-based groupware tools include shared task

¹ Sun, Java 2 Platform, Enterprise Edition (J2EE), <http://java.sun.com/j2ee/index.jsp>

² HP Labs, Jena – A Semantic Web Framework for Java, <http://jena.sourceforge.net/>

³ PIMs are software applications that provide facilities to organize personal and business information such as contacts, events and appointments.

management, discussion forums, document repositories and shared calendars. More and more organizations are challenged with the integration of geographically dispersed teams and widely publicized industrial case studies by IBM and Boeing indicate that the deployment of groupware can result in cost reductions and increased productivity. In [Ha99] it is stated that “when groupware is successful, the benefits accrued can be dramatic”.

Although this sounds promising it is not the normal case - successful adoption of groupware is rare. When investigating the causes for this contradictorily seeming situation it becomes apparent that many groupware tools lack a seamless integration with existing applications and work processes. As a result the burden placed on users to effectively support the system is higher than the perceived benefit. In Grudins' famous paper “Why CSCW applications fail: Problems in the design and evaluation of organizational interfaces” [Gr88] the electronic calendar, respectively an automatic meeting scheduling feature is being examined as an illustrative example. Such a feature as it is provided by many groupware systems basically checks the calendar for each potential participant, finding a time convenient for all. The system then notifies all participants of the appointed date. Grudin discovered that in most cases automatic meeting scheduling fails because few of the team members are likely to maintain the calendars provided by the groupware. Thus, the scheduling program finds all times open and subsequent conflicts are preassigned. Although much time has passed and groupware tools have improved the basic problem is still present.

In the ideal case each partner of a virtual team would be able to make use of his own favourite tools to which he is accustomed to anyway. As mentioned above Personal Information Managers featuring calendars, address books and to-do lists are usually well-maintained by their owners and contain information that is of value for team collaboration and coordination. Data sharing and loose coupling of these tools would result in simple but efficient distributed groupware (team calendars, group address books, team to-do lists etc.) that might be able to deal successfully with the shortcomings of above-mentioned centralistic groupware approaches⁴. This however requires either that the various tools are able to collaborate with each other or that there exists a central intelligent server acting as an intermediary.

On the technical level Web services fulfil the need for a middleware that allows applications to collaborate over the internet irrespective of their concrete implementation. The coupling of heterogeneous PIM tools based on Web services is technically feasible. In fact it can be expected that in the near future most PIM tools will provide Web service interfaces. However the huge number of tools and applications makes it unlikely that interfaces between all of them will be defined. In order to be able to cope with a wide variety of tools in an ad hoc manner, i.e. without having to integrate them manually, a more generic approach is needed.

⁴ The same applies for chat clients or document repositories like certain sections in the file system of team members.

Recently efforts towards the creation of the Semantic Web gain momentum and the research community spawns several research activities in the direction of Semantic Web enabled Web services. Adding Semantic Web support to Web services will enhance the automated collaboration between heterogeneous applications.

In the following we take up the above discussed example and describe our implementation of an automatic appointment scheduling application called “SemApp” that is able to collaborate with arbitrary personal calendars (such as provided by Lotus Notes or Microsoft Outlook) that provide a semantic description of their Web service interfaces. Thus, we demonstrate how the evolving technologies in the area of Semantic Web services can be used in order to support the ad hoc collaboration in heterogeneous virtual teams based on existing PIM tools and their Web services.

The second chapter deals with the system architecture. Additionally it describes the need for semantics in the field of Web services. The third chapter elaborates the utilized ontologies. In the fourth, fifth and sixth chapter dedicated components of the system are explained. These components are the Matchmaking Engine, the Execution Engine and the PIM Proxy respectively. The article ends with a chapter concerning conclusions and future work.

2 System Architecture

As mentioned in the introduction the basic idea behind the *SemApp* system is to provide the user, preferable the coordinator of a virtual team, with a tool that facilitates the making of team appointments. In order to accomplish this task we used Web services in conjunction with Semantic Web technologies. By using our application the team coordinator is enabled to investigate the calendars of his team members. Furthermore the system provides a feature for making an appointment with certain team members based on the free times disclosed by their personal calendars. In this regard it is important to mention, that we assume a virtual team to be heterogeneous, i.e. the team members are using different products as personal calendars.

2.1 Why Semantics?

All Web service standards like SOAP [W3C03] and quasi standards like WSDL [W3C04b] are XML-based. XML however lacks a semantic background and aforementioned standards do not define any explicit semantics either. Thus an ad hoc collaboration between Web services that have not been designed to work together is hard to achieve. The WSDL descriptions do not carry enough information for the dynamic matchmaking and invocation of heterogenous Web services. Moreover a pre-agreed contract in terms of a standardized WSDL interface description is unlikely to exist in a world of concurrent products. Therefore the task is to semantically annotate already existing Web services in order to utilize them in the *SemApp* system [HMM03].

Suitable for this kind of service descriptions is the OWL-S ontology suite [Ma03]. With OWL-S the whole spectrum of information necessary to find and execute Web services can be semantically described. The OWL-S ontology suite is based on RDF [W3C04a] and OWL [W3C04b] and consists of four complementary ontologies: *Service*, *Profile*, *Process* and *Grounding*. The *Service* ontology as an upper ontology defines Meta concepts and ties together the other ontologies. By means of the *Profile* ontology parameters, preconditions, effects and properties of a Web service can be described in a rather “abstract” manner. “Abstract” in this regard refers to the separation of the service description from its concrete realization. The *Process* ontology provides constructs for a more detailed description of a Web service and contains (among others) concepts by which the control flow during Web service execution can be specified. Via the concepts defined in the *Grounding* ontology the *Profile* and *Process* descriptions are linked with a concrete Web service and its WSDL definition. Thus *Service*, *Profile* and *Process* can be considered as abstract definitions of a Web service and the *Grounding* as a concrete one. The rigorous separation between abstract and concrete description of a Web service makes OWL-S suitable for the definition of Web service functionalities and requirements without referring to a concrete service. By means of the *Profile* ontology in conjunction with the *Service* and *Process* ontologies a “Request” for certain Web service functionality can be expressed. The Request ontology for the *SemApp* system is explained in chapter three.

2.2 Overall System Architecture

The overall system architecture of *SemApp* and the interaction of the various components are illustrated in Figure 1. Being an internet-based application the *SemApp* system can be accessed through web pages generated by the *Controller* component (which corresponds to the controller in the MVC paradigm). The *Controller* steers the system, hence is responsible for processing the input, manipulating the data and generating the output. By using the web interface a user can create and modify team respectively contact data and manipulate the associated PIMs. Team and contact data is being stored persistently by the *Storage Engine*. In order to make a team appointment the *SemApp* system needs to access the PIMs of the individual team members. Following the afore presented approach the access is provided by the heterogeneous PIMs through their Web service interfaces. To access the “proprietary” Web services in a generic manner they are annotated with OWL-S descriptions. By means of the semantic service descriptions the *SemApp* system is being enabled to invoke the Web services and thus the wrapped PIMs independently of their concrete implementation and method signatures. It is not mandatory to provide the service descriptions on the same server that hosts the Web services; rather they may be distributed throughout the internet.

The various PIMs of the virtual team members are accessed by the *PIM Proxy* component activated by the *Controller*. Being the only component aware of the PIM domain semantics the *PIM Proxy* encapsulates the communication with the Web services. The proxy invokes the *Matchmaking Engine* in order to verify the Web service that has been referenced in the contact data of each team member. Subsequently the *Matchmaking Engine* examines whether the functionality provided by the Web service

service and extracts the information necessary for its invocation. The subsequent execution of the Web service results in the manipulation of the associated individual PIM. The return values are transformed by the *PIM Proxy* to appropriate Java classes and data types which are then passed to the controller. By generating an appropriate view in terms of dynamic web pages the results are presented to the user.

The *SemApp* system is a J2EE web application and has been realized based on J2SE 1.4⁵ and Java Web Service Developer Pack (JWS DP) 1.3⁶. For the processing of the OWL-S descriptions Jena 2 by HP Labs has been used. The Web-based user interface has been implemented by the help of the JavaServer Faces 1.0⁷ technology. To demonstrate the *SemApp* System we additionally created Web service wrapper for Microsoft Outlook⁸ and Zeno⁹. This also included the description of the Web services with OWL-S and WSDL. The Outlook Web service has been implemented on the basis of .NET Framework 1.1¹⁰ and the ZENO Web service is based on JWS DP particularly on JAX-RPC¹¹.

3 Ontologies

In order to satisfy the *SemApp* system requirements it is necessary to incorporate ontologies which describe the domain of interest, i.e. the PIM domain. Ontologies ensure the common understanding of a problem domain through the definition of concepts, properties and the relations among them. Two kinds of ontologies are used: a functional ontology and several domain ontologies.

3.1 Functional Ontology

The functional ontology describes the functionalities of Web services by defining concepts like *PIM*, *GetAppointments* and *MakeAppointment* (the latter are subclasses of PIM). Thereby the ontology states on a high level what the service does. A Web service described by the *MakeAppointment* concept is responsible for registering some kind of appointments to the calendar of its wrapped PIM. Our functional ontology for the categorization of Web services is the *Profile Hierarchy* ontology and defines each concept as a specialization of the generic *Profile* concept exposed by the OWL-S *Profile* ontology.

⁵ Sun, Java 2 Platform, Standard Edition (J2SE), <http://java.sun.com/j2se/index.jsp>

⁶ Sun, Java Web Services Developer Pack (Java WSDP), <http://java.sun.com/webservices/jwsdp/index.jsp>

⁷ Sun, JavaServer Faces, <http://java.sun.com/j2ee/javaserverfaces/index.jsp>

⁸ Microsoft, Outlook, www.microsoft.com/outlook/

⁹ Fraunhofer, Zeno, zeno.fraunhofer.de/

¹⁰ Microsoft, .NET Framework, <http://msdn.microsoft.com/netframework/>

¹¹ Sun, Java API for XML-Based RPC (JAX-RPC), <http://java.sun.com/xml/jaxrpc/index.jsp>

3.2 Domain Ontologies

The purpose of domain ontologies is to define a common understanding of the problem domains associated with the appointment making process. In order to describe appointments various concepts, such as *time*, *appointment*, *account* etc. are needed [He02]. In general the definition of common ontologies is not the task of system developers but lies in the responsibility of widely accepted standardisation bodies. For the description of appointments *RDF Calendar*¹², for time concepts *Time Zone Database*¹³ and for account data *Friend of a Friend*¹⁴ are suitable ontologies. In the context of the *SemApp* system the need for describing calendar data and time periods is obvious. An account ontology is necessary in order to describe the login data used by the Web services for authentication of the requestor. For performance, granularity and consistency reasons we decided to implement some of the ontologies needed by ourselves instead of the available ones. The reimplemented ontologies are defined using OWL Lite and contain only the set of concepts needed for the implementation of our system.

3.3 Request and Advertisement Ontologies

The *Request* ontology contains all the requirements a Web service has to satisfy in order to be recognized and invoked by the *SemApp* system. In the *Request* two OWL-S *Profiles* are defined as instances of the *GetAppointments* and *MakeAppointment* concepts of the *Profile Hierarchy*. These *Profiles* correspond to the needed functionality of the *SemApp* system. As stated in the previous chapters the *SemApp* system displays all appointments of the team members and registers a collective appointment to every PIM calendar involved. Therefore two operations of a Web service are defined as *Profiles* in the *Request*, namely a *GetAppointments* operation for getting the appointments a team member has in a certain time interval and a *MakeAppointment* operation for making an appointment with a team member. Furthermore the types of the in- and output parameters are defined by means of the domain ontologies. Additionally the *Request* could define properties, preconditions and effects which are omitted here for simplicity reasons.

Figure 2 shows a fragment of the *Profile* definition in XML serialization which describes the *GetAppointments* Web service operation. The service expects two inputs and returns one output. The inputs are stated to be of the types *Account* and *Interval* defined in the domain ontologies, i.e. the Web service operation needs account data for the authorization of the requestor and the time span of interest. The return value is defined as being an instance of *VCalendar*. Further the output is declared as unconditionally, i.e. it does not depend on anything.

¹² W3C, RDF Calendar Workspace, <http://www.w3.org/2002/12/cal/>

¹³ W3C, Time Zone Database, <http://www.w3.org/2002/12/cal/tzd/>

¹⁴ FOAF, The Friend of a Friend project, <http://www.foaf-project.org/>

```

...
<hierachy:GetAppointments rdf:ID="getAppointments">
  <profile:hasInput rdf:resource="#account"/>
  <profile:hasInput rdf:resource="#dateSpan"/>
  <profile:hasOutput rdf:resource="#vCalendarOut"/>
</hierachy:GetAppointments>
<process:Input rdf:ID="account">
  <process:parameterType rdf:resource="#acc;#Account"/>
</process:Input>
<process:Input rdf:ID="dateSpan">
  <process:parameterType rdf:resource="#time;#Interval"/>
</process:Input>
<process:UnConditionalOutput rdf:ID="vCalendarOut">
  <process:parameterType rdf:resource="#vcal;#VCalendar"/>
</process:UnConditionalOutput>
...

```

Figure 2: Request Profile of the *GetAppointments* Web service operation

The *Profile* description of a concrete Web service is called *Advertisement*. In addition to the abstract description of the Web service the *Advertisement* embraces also *Grounding* ontology, since a concrete executable Web service instance is being described. *Request* and the abstract part of the *Advertisements* are used for Matchmaking. For the execution of a Web service its whole *Advertisement* ontology is used.

4 Matchmaking Engine

The *Matchmaking Engine* is a generic and relatively independent component. It compares service requests with service *Advertisements* in order to find Web services that provide the functionalities needed by the *SemApp* system. Although the *Matchmaking Engine* is able to match concepts from various domain ontologies against each other, it has no idea about the semantics the concepts carry. For the engine all the domain concepts are plain OWL classes. This ensures its generic character. In conjunction with the *Execution Engine* the *Matchmaking Engine* represents the core of the *SemApp* system. As argued in [GTB01] and [Pa02] such an engine should not just be able of recognizing exact concept matches but also specializations and generalizations. In case of multiple matches the Web service with the highest correspondence can be chosen. The following matches are recognized: *exact*, *equivalent*, *sub class*, *super class*, *property*, and *no match*. Concepts match *exact* when they have the same URI. An *equivalent* match occurs when the concepts are stated as being equal. *Sub* and *super class* matches are recognized by the *Matchmaking Engine* when the concepts are in a generalization/specialization relationship. The *property* match can occur during the parameter matchmaking process, namely when the concept of a parameter is the aggregate of the concepts of the parameters it is matched against. By this means complex parameters can be matched against their simple parts.

The engine obtains the *Request* and one or many *Advertisements*. Subsequently each *Request Profile* is matched against each *Profile* of the *Advertisements*. The results are returned to the calling *PIM Proxy*. For each member of the virtual team the *Advertisement* of his PIM Web service is obtained from the contact data record.

Besides non-functional data, such as the name of the service provider, the *Profile* descriptions mainly contain functional data like parameter descriptions. The *Matchmaking Engine* considers only to the functional description of the service. The comparison of a *Request Profile* against an *Advertisement Profile* is accomplished in the following way: First the concepts of the *Profiles* are matched, followed by the concepts of the output parameters and the concepts of the input parameters. The matchmaking algorithm is based on a concept comparison instead of matching the method signatures by name. This guarantees the match of syntactic different but semantically equivalent method signatures.

Request Profile and *Advertisement Profile* match, as illustrated in Figure 3, when the concepts of the *Profile* match with the concepts of the *Advertisement*. If the Web service returns even one of the output parameters described in the *Request* the service might be of some use for the requestor. In the case that in the *Request* no result has been described the return parameters described in the *Advertisement* can be ignored. The output parameters of the *Request* are matched against the output parameters of the *Advertisement*. Matchmaking of the input parameters is performed vice versa, i.e. the input parameters of the *Advertisement* are matched against the input parameters of the *Request*. The reverse order is crucial because of the fact that the Web service needs every single input parameter to fulfil its task. On the other hand, because only the input parameters of the *Advertisement* are needed for the proper execution of the Web service, not every input parameter of the *Request* must conform to an input parameter of the *Advertisement*.

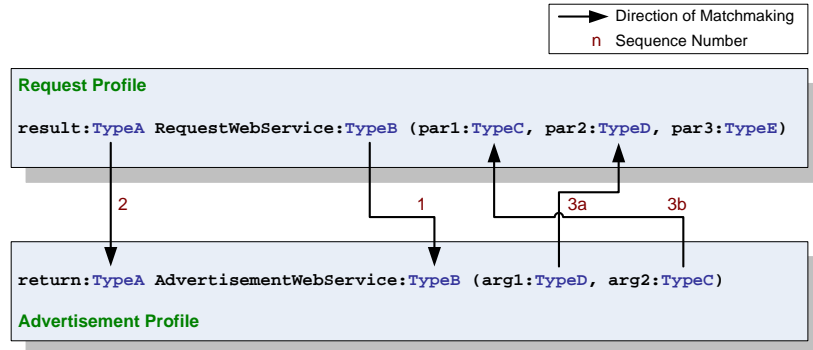


Figure 3: Matchmaking of *Request* and *Advertisement*

For the matchmaking process the reasoning facilities of the Jena framework are used. We employed the *OWL FB Rule Reasoner* with a rule set enhanced to meet our needs. Due to the fact that the *OWL FB Rule Reasoner* is still beta we faced its rather greedy memory consumption and the resulting performance and reliability problems. As a reaction we decided to redesign the domain ontologies and some parts of the OWL-S ontologies in order to have a minimum subset of concepts.

5 Execution Engine

The *Execution Engine* which is like the *Matchmaking Engine* a generic component is capable of executing arbitrary Web services in an ad hoc fashion. In order to do this the Web services must conform to the WS-I Basic Profile 1.0. Furthermore for each service a SOAP/HTTP binding and an OWL-S description has to be provided. The *Execution Engine* is realized as a standalone Java library and depends on SAAJ from the JWSDP, Apache's Xalan and Hp Labs' Jena 2 framework. SAAJ¹⁵ is used to communicate with the Web services via SOAP. With Xalan the transformation of OWL instances to XML Schema instances is accomplished. Finally Jena is (like in the *Matchmaking Engine*) used to process OWL ontologies.

The functionality of the *Execution Engine* is illustrated in Figure 4. As input the *Execution Engine* expects the OWL-S and WSDL descriptions of the Web service. The input parameters of the Web service are passed as OWL instances in terms of an RDF graph. The instances correspond to the concepts of the input and output parameters defined in the OWL-S description.

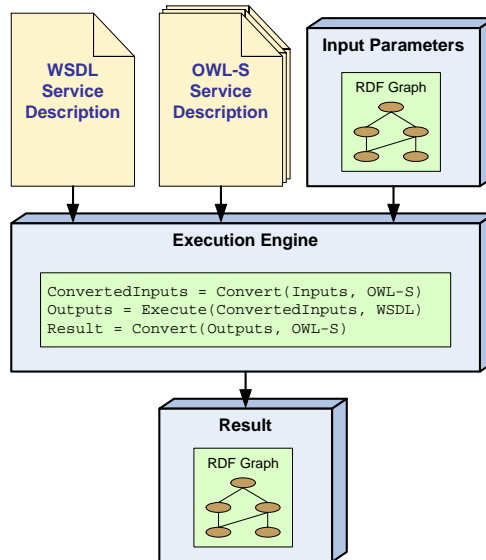


Figure 4: Architecture of the Execution Engine

Having processed the service descriptions the input parameters in form of OWL instances are converted to XML Schema instances via XSL transformations. We used the XALAN XSLT processor. The converted XML Schema instances comply to the XML schema definitions given in the WSDL definition of the Web service. The conversion is necessary because the Web service is only aware of XML Schema instances and cannot handle OWL instances. For each parameter the XSLT style sheet

¹⁵ Sun, SOAP with Attachments API for Java (SAAJ), <http://java.sun.com/xml/soap/index.jsp>

defining the transformation is referenced in the Grounding. Using the XML Schema input parameters and the information gained from the OWL-S and WSDL descriptions the Web service is executed dynamically. The return value again is an XML Schema instance which is converted to an OWL instance in the same manner as it has been done for the input parameters. Subsequently the transformed return value is passed to the calling component, i.e. the *PIM Proxy*.

6 PIM Proxy

The *PIM Proxy* component is the local representative of a PIM. On the functional level the *PIM Proxy* provides two functions to the *Controller* the same two functions described in the *Request*, one for getting the appointments of a team member and the other for inserting an appointment into the calendar of that member. It is manipulated by the *SemApp* system and controls the *Matchmaking* and *Execution Engine*. The *Controller* component manipulates the PIM over the *PIM Proxy* component. The *PIM Proxy* invokes the *Matchmaking Engine* in order to match the *Request* against the *Advertisement* of the Web services wrapping the PIM. The direct manipulation of the PIM is performed by the *Execution Engine*. The *PIM Proxy* is not a generic component since it needs to understand the concepts of the PIM ontologies, such as *RDF Calendar*. The functions take Java objects as inputs. Inside the *PIM Proxy* these objects are converted to OWL instances based on the input concepts defined in the *Request*. The same applies to the conversion of the outputs.

7 Conclusions und Future Work

In this paper we presented based on an example how Semantic Web technologies in conjunction with Web services can be used to develop tools that facilitate the ad hoc collaboration in virtual teams. The *SemApp* system exploits the semantic description of Web services in order to interact with heterogeneous Personal Information Managers without the need for a manual integration, i.e. without having to develop specific code. Based on this approach members of a virtual team are enabled to make use of their own favourite tools to which they are accustomed to anyway and which are usually well maintained. Thus acceptance and inconsistency problems are addressed increasing the acceptance and at last success of the groupware employed.

However during implementation of the *SemApp* system we also experienced some problems that may be ascribed to the immaturity of some of the technologies involved. Many of the OWL-reasoners available are not sufficiently stable and rather greedy regarding memory and processing resources. Efforts need to be undertaken in order to provide reasoners that are industrial suitable.

Another inadequately solved problem is the assignment of OWL-S concepts to WSDL parameters. The design of XSLT style sheets which are usually applied for the transformation of OWL to WSDL parameters and vice versa is not just labor-intensive

but also error-prone: The serialisation of a single OWL ontology may result in various (semantically equal but syntactically different) XML documents. A solution could be the utilisation of appropriate mapping ontologies by which the necessary transformations are described on a semantic rather than on a syntactic level.

We claim that following the presented concepts a simple but highly integrated groupware, featuring shared calendars, team to-do-lists, team chats and group address books may be designed and implemented.

Therefore as part of our future work we consider extending the system accordingly. Furthermore we intend to enhance the presented *Matchmaking Engine* towards the consideration of effects, preconditions and properties that have been specified in the service profiles. Another topic is the incorporation of security issues which for the sake of simplicity have been discounted so far.

Acknowledgements

This work partially was supported by the European Commission through the IST-1-002104-STP SATINE (Semantic-based Interoperability Infrastructure for Integrating Web Service Platforms to Peer-to-Peer-Networks) project.

References

- [Gr88] Grudin, J.: Why CSCW applications fail: Problems in the design and evaluation of organizational interfaces. Proceedings of the ACM, Conference on CSCW '88, pp. 85-92, 1988.
- [GTB01] González-Castillo, J.; Trastour, D.; Bartolini, C.: Description Logics for Matchmaking of Services. HP Labs Bristol, 2001: <http://www.hpl.hp.com/techreports/2001/HPL-2001-265.pdf>
- [Ha99] Hall, T.: Intelligence Community Collaboration, Baseline Study - Final Report. http://collaboration.mitre.org/prail/IC_Collaboration_Baseline_Study_Final_Report/to_c.htm. 1999
- [He02] Hendler, J.; Payne T.; Singh, R.; Sycara, K.: Calendar Agents on the Semantic Web. In: IEEE Intelligent Systems, 2002, May/June, pp. 84-86.
- [HMM03] Hendler, J; McIlraith, S.; Martin, D.: Bringing Semantics to Web Services. In: IEEE Intelligent Systems, 2003, January/February, pp. 90-93.
- [LS97] Lipnack, J.; Stamps, J.: Virtual Teams: Reaching Across Space, Time, and Organizations with Technology. John Wiley & Sons, New York, 1997.
- [Ma03] Martin, D.: OWL-S 1.0 Release. OWL-S Coalition, 2003, <http://www.w3.org/2004/OWL/>
- [Pa02] Paolucci, M.; Kawamura, T.; Payne, T.; Sycara, K.: Semantic Matching of Web services Capabilities. In: The Semantic Web - ISWC 2002: First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002. Proceedings; pp. 333-347.
- [W3C03] W3C (World Wide Web Consortium): SOAP Version 1.2. XML Protocol Working Group, W3C Recommendation, 2003, <http://www.w3.org/2000/xp/Group/>

- [W3C04a] W3C (World Wide Web Consortium): Resource Description Framework (RDF). Web Services Description Working Group, W3C Recommendation, 2004, <http://www.w3.org/RDF/>
- [W3C04b] W3C (World Wide Web Consortium): Web Ontology Language. W3C Recommendation, 2004, <http://www.w3.org/2004/OWL/>
- [W3C04c] W3C (World Wide Web Consortium): Web Services Description Language (WSDL) Version 2.0. Web Services Description Working Group, W3C Working Draft, 2004, <http://www.w3.org/TR/wsdl20/>