REMOTE: A Multiagent System to Select and Execute Remote Software Services in a Wireless Environment*

José M. Puyal[1], Eduardo Mena[1] and Arantza Illarramendi[2]

[1] IIS Depart., Univ. of Zaragoza, Maria de Luna 1, 50018 Zaragoza, Spain
{chema,mena}@prometeo.cps.unizar.es
[2] LSI Depart., Univ. of the Basque Country, Apdo. 649, 20080 San Sebastián, Spain
jipileca@si.ehu.es
WWW home page: *http://siul02.si.ehu.es/*

Abstract. The actual great demand of wireless devices is generating a special interest on developing new services that provide users of those devices with new software facilities. In this context we have developed REMOTE, a system that helps users of wireless devices to select, through the Web, software services of their interest and then to request their execution.
The available software services can be executed in one of the following three ways: 1) on the wireless device, without having to install in the device new software to execute the service, 2) on other computers, automatically selected by the REMOTE system, and that provide a higher performance than the user device, and 3) following a hybrid approach, some part is executed on the user device and another part on a remote computer. Thus, the REMOTE system proposed in this paper, provides users of wireless devices with new possibilities that go beyond the capabilities of their devices. REMOTE has been developed using agent technology (Tryllian), DAML-S (to express the description of the offered web services), and an ontology (managed by a system based on Description Logic) that semantically describes the different services offered. Therefore, one main contribution of REMOTE is the open nature of its architecture that allows upgrading it with many other services without changing its implementation.

Keywords: service descriptions and ontologies, mobile agents, wireless environments

1 Introduction

Nowadays, wireless devices like Personal Data Assistants (PDAs) are in great demand and their performance is being improved very quickly (the last generation of PDAs includes a 400 Mhz CPU). Nevertheless, several aspects of this kind

* Supported by the CICYT project TIC2001-0660 and the DGA project P084/2001.

of devices (such as the small size, limited CPU speed and storage space, and slow network access) make their performance be lower than the performance of desktop computers. The goal of this paper is to introduce the REMOTE[1] system which allows users of wireless devices to use a wide variety of software applications, (available as web services[2]), in order to go beyond the limitations of their mobile devices. The main contribution of REMOTE is its independence with respect to the concrete services that it is able to execute: for each service it dynamically generates the needed GUIs and performs the corresponding tasks on the most appropriate computer. The services presented in this paper are just samples of the kind of services that REMOTE might offer.

REMOTE allows users of wireless devices to browse available services and to request the execution of the selected one following three main steps:

1. *Service selection*: First, the user expresses his requirements and preferences by providing a list of keywords, then the system shows a graphical taxonomy of the available services that match her/his request (for this task, the system makes use of a software service ontology described using DAML+OIL [12, 14]); and finally the user can navigate the service ontology, see the different features of the services (description, cost, etc) and select the most appropriate one for her/his needs.
2. *Service initialization*: after the user selects a particular service, the system analyzes the semantic description of such a service, available in DAML-S format [8, 1, 2], and generates a GUI to allow the user to enter the input parameters of the service (for example, the file to compress); notice that different services have a different set of input parameters.
3. *Service execution*: taking into account the service execution requirements, the system selects the best node to execute it. Depending on the goal of the service it could be executed locally (on the user device), remotely, or following a hybrid approach. For remotely executed services, once the execution ends, the system brings the results to the wireless device of the user.

The use of DAML+OIL and DAML-S specifications makes possible that our system works with any set of software services described in such a manner. Moreover, adding and removing services from the system is a simple process that does not require source code modification and recompilation.

We have designed REMOTE using an agent-based approach [18] (we later show the benefits of this technology for our goal). After analyzing some available autonomous agent platforms we chose Tryllian [21] because it includes some important features like persistency and habitat security.

REMOTE takes part of ANTARCTICA [9], a framework that provides users of wireless devices with a new environment that fulfills some of their data management needs. ANTARCTICA follows the Client/Intercept/Server (CIS) model for mobile computing [20] and so it deals with users of wireless devices and proxies in the fixed network that provide wireless users with different functionalities.

[1] REquest Management fOr Task Execution.

[2] Nowadays, web services for a wireless framework are called *M-services*.

Thus the REMOTE system is available at the proxies that provide wireless user devices with coverage.

The rest of the paper is structured as follows. The different agents that take part of the architecture of REMOTE, and how they interact, are described in Section 2. The features of the software service ontology, that stores all the information about the web services available and that plays the role of a service catalog, are presented in Section 3. In Section 4 we present the implemented prototype. Some related work appear in Section 5 and, finally, Section 6 includes some conclusions and future work.

2 Architecture of REMOTE

The different services that REMOTE can deal with can be executed in three different ways. In this section we first describe the main features of the different execution modes, and then we present the agent architecture of REMOTE, including the goal of the main agents and how they cooperate to achieve the requested tasks.

2.1 Generic Types of Services

As mentioned before, REMOTE classifies the services into three groups, from the point of view of how they must be executed:

- *Locally executed services*: some services must be executed on the user's wireless device due to their nature, e.g., an MP3 player (which needs to use the soundcard of the user device) but does not require the installation of new software because the service itself will "travel" to the user device and execute once there.
- *Remotely executed services*: other services must be executed completely on high performance computers connected to the fixed network (on the proxy that provides coverage to the user device or on third party computers), e.g., a differential equation system solver (in general, non-interactive and CPU/disk/memory consuming services). In this way the user device is relieved of the execution of resource-consuming tasks. Once the execution ends, REMOTE returns the results to the user's wireless device. Thus, these services receive an input and return an output.
- *Hybrid services*: for interactive services that require resource consuming tasks, a part of the service is executed remotely and the other part is executed on the user's wireless device, e.g., net gaming where a client program is executed on the user device and a server program is executed on a remote computer.

It is very important to stress that the architecture of REMOTE only depends on these three kinds of service execution but not on the particular services that are available, as we show later.

2.2 Agents in REMOTE

REMOTE has been implemented using mobile agent technology [18]. Five main agents participate in the system (see Figure 1). In the following we describe them and how they interact:

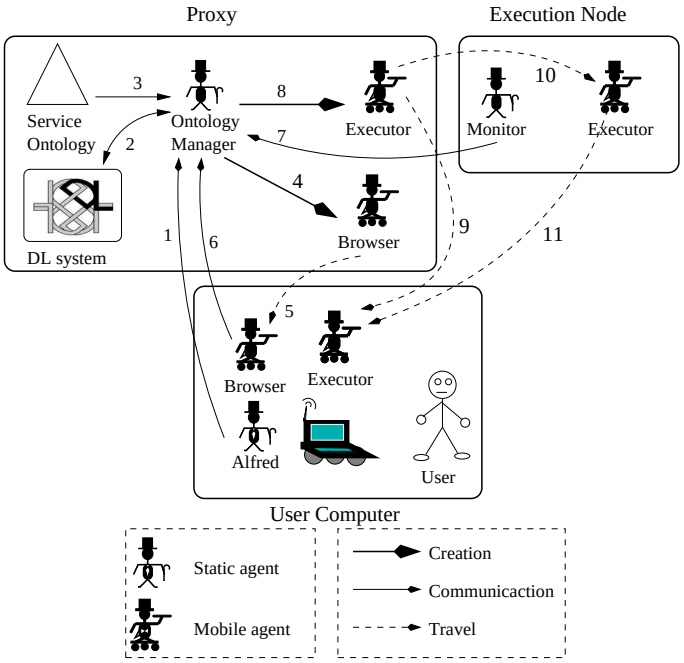


Fig. 1. Agents in REMOTE

- *Alfred, the user agent*: it is an is an "intelligent" static agent that resides at the user's wireless device and plays the role of an efficient majordomo that serves the user. Alfred stores information about the wireless device and about the user and knows how to invoke all the functionalities provided by the ANTARCTICA system[3]. Once the user chooses one functionality (REMOTE, for instance), Alfred delegates its execution on the corresponding specialized agents at the closest proxy[4] (see step 1 in figure 1).
  Alfred is the only agent involved in all the ANTARCTICA functionalities. The rest of the agents that we mention in the following are related to the REMOTE functionality only.
- *The Ontology Manager agent*: it is at the proxy that provides coverage to the user device and is on charge of: 1) initializing the service discovering

[3] As we said before, REMOTE is a part of the ANTARCTICA system.
[4] ANTARCTICA functionalities are available on every proxy.

process, and 2) managing the remote service execution. Taking a service ontology as basis (see Section 3) and considering user's preferences specified by Alfred, the Ontology Manager agent obtains a catalog of services that fulfil the requirements expressed by the user (steps 2 and 3). This task is performed using a Description Logics (DL) reasoner [3]. Then, To help the user to find the service that he needs, the Ontology Manager creates the *Browser agent* (step 4) who is on charge of presenting the obtained service catalog to the user.

- *The Browser agent*: it is created at the proxy (step 4) and then it travels to the user's wireless device (step 5) to help her/him to navigate the service catalog. The user interacts with the Browser agent and, once the user has selected a service, this agent consults the semantic description of the service and creates the most appropriate graphical user interface (GUI). Then the user enters the service input parameters through the GUI, and the Browser agent sends a message to the Ontology Manager agent (step 6) describing the selected service and the corresponding input parameters.
- *The Executor agent*: it is created at the proxy by the Ontology Manager agent to attend a service execution request (step 8). It is initialized with information about what service must be executed and where it must be executed (called *execution node*). The behavior of the Executor depends on the kind of service to execute:
  1. If the service selected by the user must be executed locally, the Executor travels to the user's wireless device (step 9) and executes the service there.
  2. If the service selected must be executed remotely, first the Executor travels to the selected execution node (step 10), then it executes the service there, and finally it travels to the wireless device (step 11) to present the results to the user.
  3. If the service selected is a hybrid service, then the Executor agent clones itself, one clone travels to the remote execution node (step 10) to execute a part of the service and the other clone travels to the user's wireless device (step 9) to execute the rest of the service. Both clones will interact to execute the service.
- *The Monitor agent*: it resides at the service execution nodes, which are high performance computers, interconnected with proxies, that allow software execution by incoming agents. The Monitor agent measures some features (like processor speed, free disk space, remote communication speed and free memory) and sends this information to the Ontology Manager (step 7). In order to decide where a service must be executed, first the Ontology Manager selects the execution nodes that fulfil the service execution requirements, and then, from this set of nodes, the Ontology Manager chooses the best node taking into account the user preferences (who could prefer, for instance, paying more for the service but getting the results quickly) about the service execution.

We would like to stress that the functionality of the Software Manager and Browser agents are very similar to their counterparts in the ANTARCTICA

Software Retrieval Service (SRS), where the user is looking for installing new software on his wireless device rather than for executing services; see [17] for a more detailed description of the functionality of such agents.

We can observe that mobile agent technology perfectly fits the requirements of a system like REMOTE, performing the processing wherever needed. Alternative solutions would need extra infrastructure (servers waiting for remote invocations) on fixed computers and, what is worse, on users' wireless devices, in order to achieve our goal. However we exploit the features of mobile agents to bring the execution of tasks to the most appropriate computer or device, which also allows a dynamic managements of task executions: at any time, the execution of a service could resume and continue on a different and more suitable computer effortless.

3 The Service Ontology

REMOTE uses an ontology [10] that stores information about all the available services. This ontology is a rooted acyclic digraph that plays the role of the knowledge base of the system. It is composed by service categories, the semantic relationships among them, and the individual description of each service (see Figure 2 which shows a subset of the ontology used in our prototype; individual services are the leaves of the ontology[5]).

The ontology stores different information about services: 1) *Internal information*, like the execution type (local, remote, or hybrid); 2) *Descriptive information* that will be shown to users and can be used like search criteria[6]; 3) *Execution requirements* and preferences of the service, which are used by the Ontology Manager agent to select the best node to execute the service; notice that this information has no sense in locally executed services. Each service can include its execution preferences in terms of processor speed, free disk space, free memory and remote communication speed.

The ontology is described using DAML+OIL [12, 14], an ontology markup language that takes an object-oriented approach and that was designed to describe the structure of a domain in terms of classes and properties. DAML+OIL is equivalent to a very expressive Description Logics (DL) [3], more precisely, it is equivalent to *SHIQ* [11] with the addition of existentially defined classes and datatypes (often called concrete domains in DL).

The DAML+OIL ontology description is used integrated with a Description Logics (DL) reasoner that provides concept based reasoning services. When the Ontology Manager receives a service discovering request, it creates a new concept and fills its properties with the search preferences of the user, then it asserts this

[5] As we said before, the goal of this paper is not presenting particular services but the architecture of REMOTE; services available in the prototype are just examples to show the functionality of REMOTE.

[6] Our prototype deals with keywords and cost roles as search criteria but due to the use of the DL system it is very simple to add or remove search criteria (any ontology role can be used as search criteria).

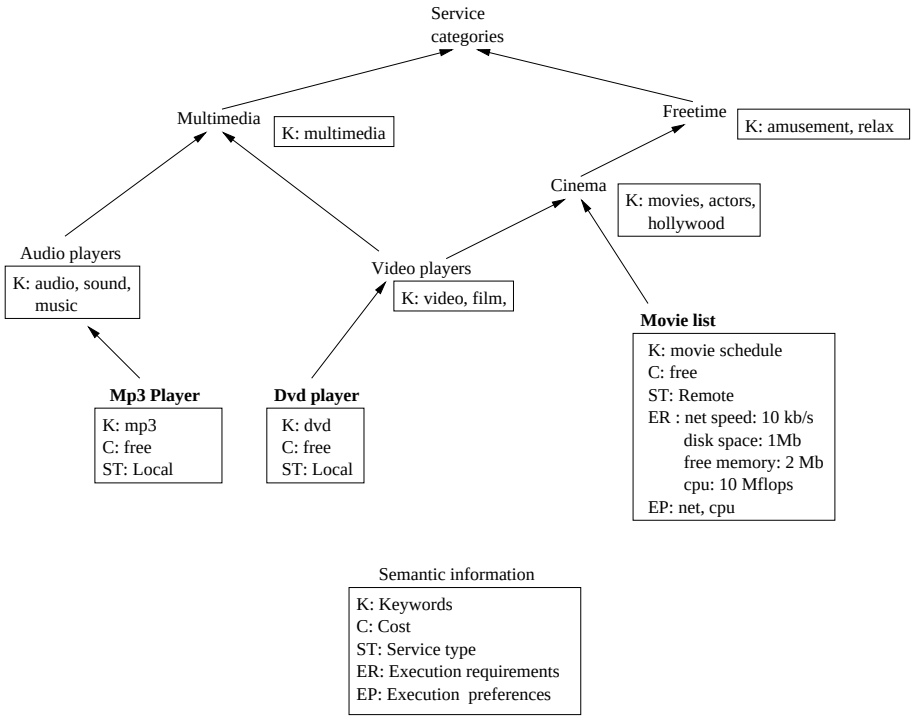


Fig. 2. Extract of the ontology used by REMOTE and terms definitions

new concept to the DL system which is on charge of classifying it in the ontology. The child nodes of this new concept are the services and services categories that must be sent to the user. For instance, if the keyword 'movie' and the cost 'free' are specified by the user, a new auxiliary concept with those values for role keywords and cost, respectively, is defined; then the DL reasoner will classify it within the service ontology and the resulting subtree under such a concept are the services that fulfil the user requirements (in the sample ontology in Figure 2, services ' DVD player' and 'Movie list').

An important part of the ontology consists of the DAML-S description [8, 1, 2] of each service. DAML-S is a markup language to provide a computer-interpretable description of web services and also the mean by which they are accessed. DAML-S allows the description of simple and composite web services. In web service composition, DAML-S provides a mechanism to describe how the subprocesses must be executed and synchronized to succeed in the global web service execution. The mean by which a web service is accessed is described using WSDL bindings [6] for its use with DAML-S and the SOAP binding [5], so the web service inputs and outputs are encoded and transmitted in XML messages.

4 Prototype

When the user of the wireless device starts REMOTE, the system presents him a graphical user interface (GUI) which is used to specify the requirements of the service that he is looking for. Let us suppose, for example, that the user is looking for locating an address on a map. In our prototype, the user expresses the main features of the service in terms of keywords and cost limit through this GUI; both roles are the search criteria in our prototype. In the example (see Figure 3), the user specifies the keywords 'map', 'address', 'guide', 'street', 'multimedia', and 'sightseeing' and he does not want to pay a lot for the service.

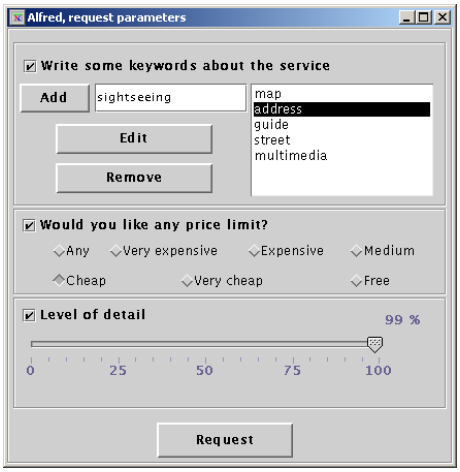


Fig. 3. Service initialization: Looking for a streetmap service

Alfred sends this information to the Ontology Manager which queries the DL system in order to match services categories with the user request. We use FaCT [13, 4] as DL system; FaCT provides a DL reasoner engine as a network resource via CORBA [19]. FaCT returns to the Ontology Manager the services that satisfy the constraints. After that, the Browser is created and travels to the user's wireless device. Once there it creates the GUI that allows the user to navigate the service catalog to consult the features of the different services (see Figure 4).

When the user selects a service, the Browser analyzes its DAML-S description and creates a GUI that allows the user to enter the input parameters of the service. In the example, after reading the features of the services in the catalog shown in Figure 4, the user selects the "streetmap" service from the catalog. Such a service requests a complete address as input parameter (see Figure 5.a) in order to obtain the map where that address is located.

According with its execution requirements, the streetmap service must be executed on a node with a high speed Internet access because it downloads the

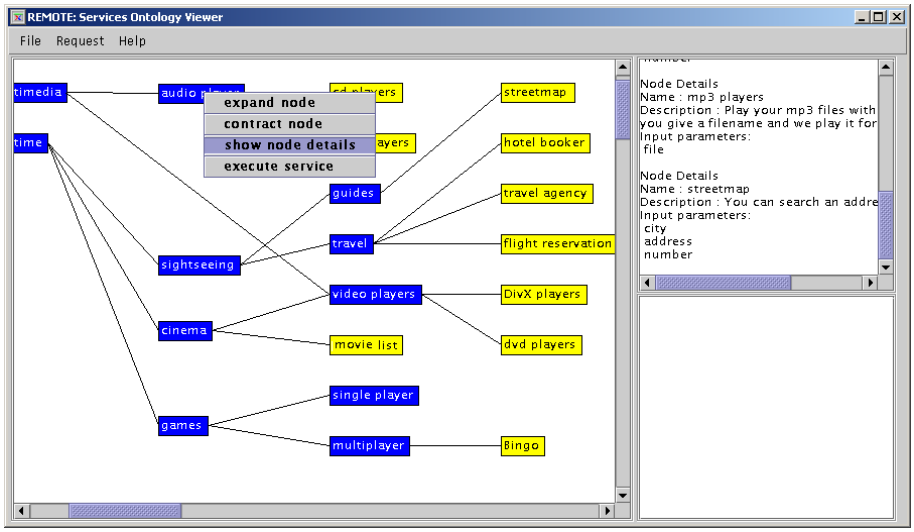
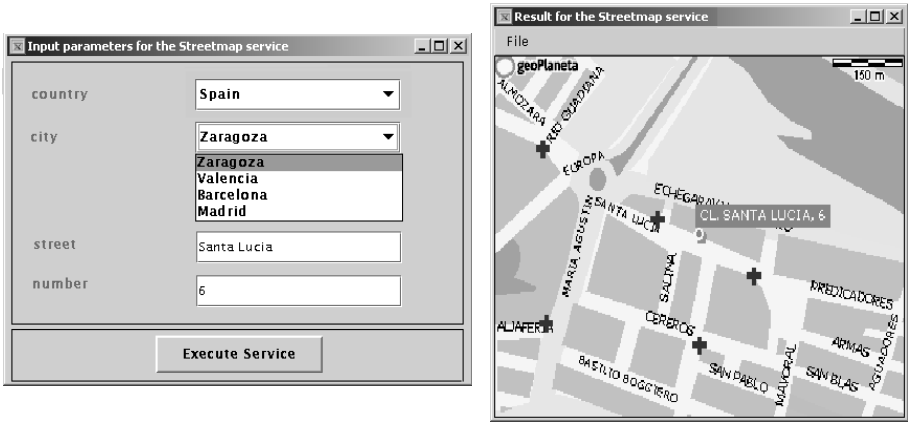


Fig. 4. Service catalog retrieved for the keywords specified in Figure 3



(a) (b)

Fig. 5. (a) Input parameter GUI for the streetmap service and (b) result obtained after remote execution

map from the Web: that is why this service should not be executed on the user wireless device but on a remote node. Therefore, the Executor agent travels to certain execution node selected by the Ontology Manager after analizing the different possibilities. Once there, the Executor retrieves the map corresponding to the specified address, and then it travels to the user's wireless device to present him the result (see Figure 5.b).

In order to test REMOTE we have implemented some concrete services that match the different types of generic services supported by our system (see Table 1); such services are organized in the sample ontology presented previously. However we would like to stress that the goal of this paper is not to show the benefits of those sample services but presenting the benefits of combining the technology of mobile agents with web services and ontologies to provide users of wireless devices with a very flexible system.

Service	**Type**	**Description**
Gzip compressor	Local	File compressor and decompressor based on the Lempel Ziv algorithm
MP3 player	Local	Audio player for MPEG3 encoded sound files
Movie list	Remote	Service that provides the list of movies that you can watch at the selected cinema
Streetmap	Remote	Given a complete address it provides a map locating it
Prime factorization	Remote	Service that factorizes a number
Bingo	Hybrid	Client/server game to play bingo
Flight reservation	Local	Flight reservation emulation service
Hotel Booker	Local	Room reservation service
Travel agency	Local	Composite service that allows to plan a travel: first it uses the flight reservation service (departure flight), then the hotel booker service, and finally the flight reservation service again (returning flight)

Table 1. Services implemented to test REMOTE

5 Related work

Our system can be viewed as a general purpose remote execution platform. Remote execution platforms are usually found in the area of resource management for distributed parallel systems where an optimal load balance is very important. The first generation of automatic load balancing systems assigned jobs to the least loaded computers. One of the representative systems in this group is the *Condor system [16]* whose main goal is to maximize the use of computers with the minimal interference of the owners. This system also supports process migration using a checkpointing mechanism. Another important feature of remote execution platforms is to hide the underlying networks and operative systems. In this sense we can mention BALANCE [15], a flexible parallel load balancing system for heterogeneous computing systems and networks. BALANCE is designed to support a wide range of software, including parallel and distributed applications as well as schedulers. In contrast with Condor and BALANCE, our approach not only offers an execution platform but it also offers a service

discovery process and a dynamic invocation mechanism. We could use their complementary approaches to select the best node to execute a service.

The system presented in [7] (namesake of our proposal) is a tool for automatic remote execution of CSIM18 simulation models. CSIM18 simulation models are built as programs that must be executed from the system prompt: they read inputs from text files and output into another text file. These model execution often take long time and the system distributes the programs among remote computers and collects the output files once programs have finished their execution. However our approach is a platform for remote execution of services of different nature (executed locally, remotely or in a hybrid way) and with different goals.

6 Conclusions

Taking into account the widespread use of wireless devices, we have presented in this paper a system that allows users of those devices to select and execute services wherever needed in an easy, guided and efficient way. Easy, because the system allows users to express their service requirements and preferences at a semantic level, i.e., they express what they need but not how to obtain it. Guided, because the service, using specialist knowledge-driven agents, only presents to the user those service categories related to her/his requirements and helps her/him to browse those categories until she/he finds the service. And efficient, because service features are considered to select the best execution node: some services are executed on the user device and other services are executed on computers that offer higher performance.

As future work, we plan to work on service composition which involves process modeling and workflow technology. Our research is oriented towards an implementation model based on Petri Nets and mobile agents, in order to achieve a flexible and expressive mechanism to perform service composition using different computers for the execution of subprocesses.

References

1. DAML Services Coalition (alphabetically A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, and H. Zeng). DAML-S: Semantic markup for web services. In *International Semantic Web Working Symposium (SWWS'01), California (USA)*, July–August 2001.
2. The DAML Services Coalition (alphabetically A. Ankolekar, M. Burstein, J.R. Hobbs, O. Lassila, D.L. Martin, D. McDermott, S.A. McIlraith, S. Narayanan, M. Paolucci, T.R. Payne, and K. Sycara). DAML-S: Web service description for the semantic web. In *The First International Semantic Web Conference (ISWC'02), Sardinia (Italy)*, June 2002.
3. Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Pastel-Scheneider. *The Description Logic Handbook. Theory, Implementation and Applications.* Cambridge University Press, ISBN 0-521-78176-0, 2003.

4. S. Bechhofer, I. Horrocks, and S. Tessaris. CORBA interface for a DL classifier. Technical report, March 1999.
5. Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple object access protocol (soap) version 1.1, 2002.
6. Roberto Chinnici, Martin Gudgin, Jean-Jacques Moreau, and Sanjiva Weerawarana. Web services description language (wsdl) version 1.2, 2002.
7. K. Christensen. Remote: A tool for automatic remote execution of csim18 simulation models. In *35th Annual Simulation Symposium, San Diego, California (USA)*, pages 134–142, April 2002.
8. DARPA, 2003. http://www.daml.org/services/daml-s/0.7/.
9. A. Goñi, A. Illarramendi, E. Mena, Y. Villate, and J. Rodriguez. ANTARCTICA: A multiagent system for internet data services in a wireless computing framework. In *NSF Workshop on an Infrastructure for Mobile and Wireless Systems, Scottsdale, Arizona (USA)*, pages 119–135. Lecture Notes in Computer Science (LNCS 2538) 2002, ISBN 3-540-00289-8, October 2001.
10. T. Gruber. http://www-ksl.stanford.edu/kst/what-is-an-ontology.html.
11. I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer-Verlag, 1999.
12. Ian Horrocks. DAML+OIL: a description logic for the semantic web. *IEEE Bull. of the Technical Committee on Data Engineering*, 25(1):4–9, MAR 2002.
13. Ian Horrocks. Reasoning with expressive description logics: Theory and practice. In Andrei Voronkov, editor, *Proc. of the 18th Int. Conf. on Automated Deduction (CADE-18)*, number 2392 in Lecture Notes in Artificial Intelligence, pages 1–15. Springer-Verlag, 2002.
14. Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. Reviewing the design of DAML+OIL: An ontology language for the semantic web. In *Proc. of the 18th Nat. Conf. on Artificial Intelligence (AAAI 2002)*, 2002. To appear.
15. Chi-Chung Hui, Samuel T. Chanson, Pui-Man Chui, and Ka-Ming Lau. BALANCE - a flexible parallel load balancing system for heterogeneous computing systems and networks. In *INFOCOM (2)*, pages 896–903, 1996.
16. M.J. Litzkow, M. Livny, and M. W. Mutka. Condor-a hunter of idle workstations. In *11th International Conference on Distributed Computing Systems*, June 1988.
17. E. Mena, J.A. Royo, A. Illarramendi, and A. Goñi. An agent-based approach for helping users of hand-held devices to browse software catalogs. In *Cooperative Information Agents VI, 6th International Workshop CIA 2002, Madrid (Spain)*, pages 51–65. Lecture Notes on Artificial Intelligence (LNAI), ISBN 3-540-44173-5, September 2002.
18. D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White. MASIF, the OMG mobile agent system interoperability facility. In *Proceedings of Mobile Agents '98*, September 1998.
19. OMG. The Common Object Request Broker : Architecture and Specification. Technical report, Object Management Group, Inc., dec 1993.
20. E. Pitoura and G. Samaras. *Data Management for Mobile Computing*. Kluwer Academic Publishers, 1998.
21. Tryllian. http://www.tryllian.com.