The OpenKnowledge System: An Interaction-Centered Approach to Knowledge Sharing

Ronny Siebes[1], Dave Dupplaw[2], Spyros Kotoulas[1], Adrian Perreau de Pinninck[3], Frank van Harmelen[1], and David Robertson[4]

[1] Vrije Universiteit Amsterdam
{ronny,kot,frankh}@few.vu.nl
[2] University of Southampton, UK
dpd@ecs.soton.ac.uk
[3] Artificial Intelligence Research Institute (IIIA - CSIC)
adrianp@iiia.csic.es
[4] The University of Edinburgh, Edinburgh, UK
dr@inf.ed.ac.uk

Abstract. The information that is made available through the semantic web will be accessed through complex programs (web-services, sensors, *etc.*) that may interact in sophisticated ways. Composition guided simply by the specifications of programs' inputs and outputs is insufficient to obtain reliable aggregate performance - hence the recognised need for process models to specify the interactions required between programs. These interaction models, however, are traditionally viewed as a *consequence* of service composition rather than as the focal point for *facilitating* composition. We describe an operational system that uses models of interaction as the focus for knowledge exchange. Our implementation adopts a peer to peer architecture, thus making minimal assumptions about centralisation of knowledge sources, discovery and interaction control.

1 Introduction

The pool of potentially available knowledge on the Internet is immeasurably large. It is fed by the traditional Web: by application programs feeding data onto the Web, by Web services accessed through various forms of application interface, by devices that sense the physical environment, and so on. It is consumed in a wide variety of ways and by diverse mechanisms (and of course consumers may also be suppliers). The aspiration of OpenKnowledge is to allow knowledge to be shared freely and reliably, regardless of the source or consumer. Reliability here is interpreted as a semantic issue. The Internet is in the fortunate situation that physical and syntactic reliability have been solved to satisfactory degrees, making semantic reliability the main challenge. Semantic reliability means that we want the meaning ascribed to knowledge that is fed into the pool, to be preserved adequately for the purposes of consumers.

Of course such "open knowledge sharing" is an aspiration that we know to be unattainable, in the strong sense where all knowledge supplied can be consumed with perfect freedom and reliability. Globally consistent common knowledge is impossible to guarantee in an asynchronous distributed system[1].

Interaction-specific knowledge sharing: The good news is that only a small proportion of the pool of available knowledge will be of use to any given consumer, since each must have an upper limit on how much knowledge it can process. A pragmatic aim of open knowledge sharing, then, is to obtain knowledge *appropriate to the activities* in which each consumer wants to engage, while maintaining free and (adequately) reliable connections between suppliers and consumers.

The standard way in which activities (and their sequencing) are described is via process languages like BPEL [2] or LCC [14], since no complex activity can be represented formally without modeling its temporal structure. In principle, we could use (models of) these activities to limit the scope of knowledge that we attempt to share. There is a problem however: activity models are themselves knowledge that must be shared. In other words, when an item of knowledge is openly shared in the context of some common activity it is necessary for the supplier and consumer to have knowledge of that context, otherwise there is no benefit (in terms of reliable knowledge sharing) from the activity focus.

For this reason the OpenKnowledge project has at its core a mechanism for sharing models of activities that require interaction across the Internet. We refer to such models as *interaction models* [14]. We expect that communities of practice will naturally form around collections of interaction models and that these communities can be stabilized by a mechanism for their rapid sharing across peer groups. Notice that this is explicitly an *interaction-centered* approach to knowledge sharing, as opposed to the traditional *data-centered* approach.

By building a system, we demonstrate that sharing interaction models at very low cost to consumers and suppliers is possible. The novelty of this system is that each interchange of knowledge is made in the context of the (shared) interaction model. The system is completely distributed using P2P technology. Each peer that participates in the OK system will at least run a piece of code that we call the *OpenKnowledge Kernel* [4] enabling the base functionality to find these interactions and the code or peers that enable to run the services.

2 Relevant Literature

Clearly, many others have previously identified the goals of reliably sharing knowledge freely and reliably, regardless of the source or consumer. In this paper, we will not discuss the plethora of work in the dominant data-oriented attempts at solving this problem, such as data-integration [10], schema and ontology mapping [15], data-mediators [7], etc. Instead, in this section we discuss some of the approaches that have also taken an interaction-oriented approach: web-services, grid-services and multi-agent systems. Although typically data-centric, we also include P2P systems in our comparison, because the OpenKnowledge architecture has strong P2P characteristics

We do not aim to provide a full-scale literature study here. Instead, we identify the key ideas behind each of these approaches, and argue why OpenKnowledge occupies a unique niche in this landscape.

[1] Even if it were a philosophically and culturally coherent notion.

Web Services. Perhaps the most closely related effort to OpenKnowledge is the work on web-services [3]. The aim of web-services is to enable invoking and executing of services in a distributed, scalable and interoperable manner. The work on *semantic* web-services [19] adds to this the goals to automatically locate and compose such services in an open and heterogeneous environment like the Web.

Both approaches (web-services and OpenKnowledge) use the principle that if the services are formulated into information objects (web-service *descriptions* either purely syntactic, such as WSDL [5] or semantic such as WSDL-S [17] and OWL-S [11]), then they can also be the subject of reasoning tasks for search and composition.

The OpenKnowledge approach is in some ways more flexible than the web-services approach, but in other ways more restricted. Semantic web-service work aims at automatic on-line composition of simple services into complex services, by means of intelligent algorithms (e.g. based on configuration [20] or planning [21]), whereas, OpenKnowledge restricts itself to executing predefined "work-flows" of services (the "interaction models" to be discussed later in this paper). The only decision that OpenKnowledge makes at run-time is which instance of a service is executed; that is, which agent providing the service will be used (i.e. "recruiting", not composition).

This recruiting aspect of OpenKnowledge is more general than the web-service architecture because it separates the advertising of a service from the execution of a service. In the web-service architecture, it is generally assumed that advertisements of service functionality are accompanied with the name of the executor of the service. In short: the matching goals of both approaches are the same (finding a service that matches a given functionality), while the composition goals of both approaches are different: OpenKnowledge aims to recruit peers to execute predefined work-flows, whereas semantic web-services aims to automatically compose complex work-flows out of atomic services.

Furthermore, OpenKnowledge explicitly acknowledges the need for approximate matching of service requests with advertisements, whereas this is only marginally the case in the semantic web-service world [1], and entirely absent in regular web-services.

Finally, OpenKnowledge aims explicitly for a distributed storage model for the work-flows and service descriptions, whereas all the dominant web-service architectures (UDDI [12] for regular web-services, WSMX [8] for semantic web-services) assume a centralised architecture.

Grid-Services. The general area of grid-services is even less well circumscribed than web-services, hence it is more difficult to make a crisp comparison. Literature on Grids [6] often align their approaches to the service-oriented architecture (SOA). In contrast to web-services, grid-services are typically organized in fixed work-flows. This makes them more similar to the OpenKnowledge approach, however, grid-services emphasise various aspects that are ignored in OpenKnowledge: long-term stability of services, provenance, quality of service and resource monitoring. Similar to web-services, grid-services differ from Open Knowledge by advertising a service functionality together with the identification of the service-provider; OpenKnowledge decouples these two and hence allows for a separate "recruiting" step. Finally, and perhaps most importantly, most grid-systems provide only a centralized mechanism for advertising services and work-flows, while OpenKnowledge aims for a fully distributed mechanism.

In particular, the myGrid project [18] is in many respects close to the goals of OpenKnowledge in its use of pre-configured work-flows and its approach to manual composition of such work-flows. However, it relies on centralized storage of such work-flow patterns, which is in sharp contrast with the fully distributed architecture of Open-Knowledge.

Peer-to-peer systems. Obviously, OpenKnowledge is close in spirit to the work on peer-to-peer (P2P) systems. The central P2P ideas of distributed storage, lack of centralized address registers and the symmetric roles of every peer as both provider and requester, are fully adopted by OpenKnowledge. Nevertheless, OpenKnowledge makes two important deviations from most P2P systems. First, most P2P systems aim at data sharing, whereas OpenKnowledge aims at *service sharing*. Of course, data sharing is simply a special case of service sharing (namely sharing a data-access service), making the OpenKnowledge system more generic. Secondly, OpenKnowledge is in the small, but rapidly growing, family of *semantic* P2P systems [16], which use rich descriptions of the content that each peer has to offer for purposes of routing queries through the network.

Agents. A final class of closely related systems is that of multi-agent systems. In general, there is a superficial similarity between multi-agent and P2P systems: distributed sets of autonomous processes exchanging information. However, on closer inspection, there are rather significant differences. In particular, agent systems often have highly structured architectures inside each agent often relying on cognitive metaphors for their architectural constructs (such as the Believes, Desires and Intentions (BDI) architecture [13]). P2P systems typically treat their peers as atomic. Finally, agent-systems emphasize their pro-active nature (autonomously reacting on their changing environment), while P2P systems, including OpenKnowledge, assume more classical reactive stance.

The differences and similarities described above are all summarized in Table 1. This table shows that OpenKnowledge inherits many aspects from other approaches but also occupies a particular niche, having features not fully explored by others.

3 An Extensive Example Describing the Functionality of the OpenKnowledge System

In this section we provide an extensive example how our system can be used. The architecture of the system is described in another paper [4]. From a user perspective, the OpenKnowledge system is a software bundle that allows a user to find, compose and execute tasks. Those tasks can be executed by users and/or software components. The tasks are described by *Interaction Models* (IM), where each IM is a formally described set of roles together with the process-flow between those roles. Users subscribe their peer to play roles within an interaction. For example, the task of buying an item requires at least the seller and buyer roles, and perhaps a payment service role. We call instances of these roles (e.g. a particular seller or a particular buyer) *OK-Components (OKCs)*. An OKC, for example a creditcard service, may play a role in many IMs. If the roles are constrained by some external functionality, then services provide that functionality. Much of the functionality of the OK system relies on the *Discovery and Team formation Service (DTS)*, which is a distributed storage and retrieval system over a P2P network. Its main responsibilities being the following:

Web-Services	**similarities:**	service-oriented,	
		distributed,	
		automated search	
		based on semantic descriptions	
	differences:	**Web-Services**	**OpenKnowledge**
		composition of atomic services	predefined workflows
		fixed link to executing party	dynamic recruiting
		centralised advertising	distributed
		equivalence matching	approximate matching
Grid-Services	**similarities:**	service-oriented,	
		fixed workflows	
		distributed	
	differences:	**Grid-Services**	**OpenKnowledge**
		provenance	absent
		QoS	reputation mechanisms
		resource monitoring	absent
		centralised advertising	distributed
		fixed link to executing party	dynamic recruiting
Peer-to-Peer Systems	**similarities:**	distributed,	
		scalable,	
		symmetric roles of each peer	
	differences:	**P2P Systems**	**OpenKnowledge**
		aimed at data-sharing	service sharing
		independent of content	exploit semantics
Multi-Agent Systems	**similarities:**	distributed,	
		symmetric roles of each peer	
	differences:	**Multi-Agent Systems**	**OpenKnowledge**
		cognitive architecture	none
		central brokers	scalable discovery
		pro-active behaviour	reactive

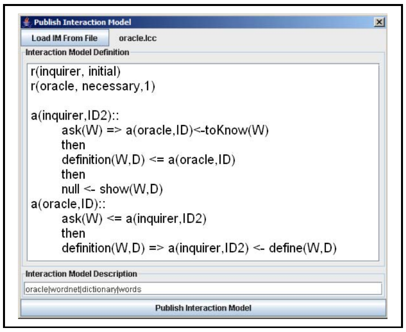
Fig. 1. OpenKnowledge compared to other approaches

– *IM Discovery* - the DTS is used to publish, discover and retrieve IMs.
– *OKC Discovery* - the DTS is also used to publish, discover and retrieve OKCs. This enables reusability thus providing scalable functionality. OKCs can be discovered either in the context of an already known IM or independently.
– *Role subscription* - peers can subscribe a locally stored OKC to play a role in an IM. Additional information such as annotations and restrictions concerning the other participants can be given along with the subscription.
– *Coordinator subscription* - peers may also subscribe to act as interaction coordinators.
– *Team formation and interaction initialization* - the DTS uses subscription information to form teams of OKCs, which will, potentially, participate in an interaction, and finds a subscribed coordinator to orchestrate them.

The system is based on previous work where the algorithms are simulated and implementations are emulated in order to see the performance of them [9]. More about the DTS can be read in the architecture paper [4]. Now we will explain the functionality of the first OpenKnowledge system by going through an example where we show how a dictionary service can be created and used.

3.1 Writing and Publishing an IM

In figure 2 user A uses the OpenKnowledge System to develop an IM for the dictionary service, by describing an interaction between two roles. One role is used to query the service, called the *inquirer*, and the *oracle* role provides the answer. In this example, the IM is written in the LCC language [14]. Current work in the project is to also have support to other languages like BPEL. The LCC model can be read as follows:


Publish Interaction Model
Load IM From File
oracle.lcc
Interaction Model Definition

```
r(inquirer, initial)
r(oracle, necessary,1)

a(inquirer,ID2)::
        ask(W) => a(oracle,ID)<-toKnow(W)
        then
        definition(W,D) <= a(oracle,ID)
        then
        null <- show(W,D)
a(oracle,ID)::
        ask(W) <= a(inquirer,ID2)
        then
        definition(W,D) => a(inquirer,ID2) <- define(W,D)
```

Interaction Model Description
oracle|wordnet|dictionary|words
Publish Interaction Model

Fig. 2. User interface showing an IM editor (LCC as the language in this example) and a button to publish the IM on the OpenKnowledge network

1. `r(inquirer,initial)`. This line states that the 'inquirer' role is the one that starts the interaction.
2. `r(oracle,necessary,1)`. Statement indicating that at least 1 peer needs to play the oracle role.
3. `a(inquirer,ID2)::`. A statement giving the 'inquirer' role an identifier 'ID2' and the '`::`' means that the definition of the role starts after it.
4. `ask(W) => a(oracle,ID) <- toknow(W)`. If the user wants to know a definition for a word 'W' it can start the interaction by fulfilling the constraint `toknow(W)`. In LCC the '`<-`' symbol is used to indicate that after it a constraint is defined. When the constraint is satisfied (i.e. the user provided 'W'), a message 'textttask(W)' is sent to the 'oracle' role identified by 'ID' (note that `a(oracle,ID)` relates the role to an identifier). In LCC the '`=>`' symbol is used to indicate that a message (in this case `ask(W)`) is sent from the current role to another role (in this case the 'oracle').
5. `definition(W,D) <= a(oracle,ID)`. In this line the 'inquirer' waits for the oracle role (`a(oracle,ID)`) to send a message with the definition as content (`definition (W,D)`). In LCC the '`<=`' symbol is used to indicate that a message (in this case `definition(W,D)`) should be expected from another role (in this case the 'oracle' role).
6. `null <- show(W,D)`. When the 'oracle' sent the message to this role, this statement shows the answer to the user. In this case `show` is a special constraint which is understood by the system to show a message (in this case with the query: `W` and the answer: `D`) in the user interface. `null` means that nothing happens after the constraint `show(W,D)` is fulfilled.
7. `a(oracle,ID)::`. Gives the 'oracle' role identifier 'ID' and starts to give its definition.
8. `ask(W) <= a(inquirer,ID2)`. This line makes the 'oracle' role wait for a message `ask(W)` from the 'inquirer'.
9. `definition(W,D) => a(inquirer,ID2) <- define(W,D)`. When the 'oracle' got the 'ask' message (previous line is executed), it will try to fulfill the 'define (W,D)' constraint, and if that is true, a message with the content `definition(W,D)` is sent to the 'inquirer'.

Now that a user `A` wrote down the IM, they should provide some keywords to describe the functionality of the IM. In the system we provide automated mapping and similarity algorithm to relate similar keywords during search.These keywords are needed by the DTS to index them in order to be retrieved by other peers. In this case, `A` decides to give the keywords '*oracle, wordnet, dictionary, words*'. Our current work tries to extend the ways to describe the functionality of an IM, for example by providing concepts from ontologies instead of keywords. Now that the IM is ready and the keywords are provided, the user can decide to publish it on the OpenKnowledge network by connecting to the network and pressing the 'Publish Interaction Model' button. The DTS will make sure it is scalably stored and indexed by the provided keywords.

3.2 Creating and Publishing OKC's

Besides writing the IM in the previous section, user `A` also writes the OKCs that implement both roles in the IM respectively. Currently, the user `A` has to implement their OKC by writing some code to a specific Java API. In simple terms, the methods in the Java source code should match the names and the arguments of the constraints in the roles, which are `toknow(W)` and `show(W,D)` for the 'inquirer' role and `define(W,D)` for the 'oracle' role. Note that here we assume `W` and `D` are of type `STRING`, where in the extended LCC language also types are supported, meaning that the definitions would be something like `show(W:STRING,D:STRING)`. After user `A` has implemented the interfaces, (s)he opens the window from the OpenKnowledge Kernel software where it can wrap the code into OKC's (the figure is not shown here due to space constraints).

The user loads its IM and attaches the java implementations of the role constraints via the user interface of the kernel. Also the OKCs may be described by a set of keywords, because they can be used as role implementations for other IMs and therefore need to be indexed so that they can be retrieved by the DTS. The intuition behind this is that an OKC implementing a credit-card payment service can be used in many IMs. Also these keywords can be used in the OKC selection process that allows a user to select their preferred OKCs after multiple matches have been found to an IM. For example, it can be that two OKCs exactly match the same 'oracle' role but one delivers results in English and the other in Spanish.

By clicking the '*Create OpenKnowledge Component*' button, the OKC is created and ready to be used. By sending a 'subscribe' message to the DTS (not shown in the figures), it tells the network that it is able to execute the role of 'oracle' for the given IM. Given that the user used Wordnet as the underlying implementation, it annotates the OKC with the keywords '*dictionary, english, wordnet,lookup*' (not shown in the figures). Besides this, `A` decides to publish the 'inquirer' OKC to the network, so that other users also may download it and run it on their own machines.

3.3 Searching for IMs and OKCs

Peer B wants to find a service that will allow it to find definitions of words in Spanish. It opens the search window from the OpenKnowledge Kernel (not shown due to space constraints). In this case, in the beginning (s)he searches for IMs matching to the word 'oracle'. The system starts searching and shows the found IMs together with their roles to the user. Assume that user `B` finds the IM together with the roles 'orcale' and

’inquirer’. The user wants to play the role of the inquirer written by user `A` and therefore decides to download it and tells the DTS that it is willing to play the role.

3.4 Team Formation and Execution

Given that in the previous steps `A` and `B` have both told the DTS that by subscribing their OKCs that they are willing to play the roles of ‘oracle’ and ‘inquirer‘ respectively, the DTS knows that all roles are instantiated meaning that there are enough peers to start the interaction. Now imagine that another user `C` also published an OKC that is able to fulfill the role of ‘oracle’, but has annotated its OKC with the keywords *dictionary, spanish*. So now there are three peers ready to play. The DTS selects a coordinator peer from the pool of peers. This is currently selected randomly (but current ongoing work is to make it reputation-based). This coordinator receives a message from the DTS with the three peers, their OKC descriptors and the IM. The coordinator now can start the team formation process.

The coordinator sends each peer the list of peers willing to play together with their OKC descriptions. Now the peers can select, automatically or with the user in the loop (depends on the OKC implementation), with whom to play. Assume that both the Spanish and English oracles have automatic selection process saying that they always like to play with whomever. However, the inquirer has user `B` in the loop, where the user selects the peer from user `C`, because its OKC description matches its wishes and sends its preferences back to the coordinating peer. Now that the coordinator has (within a certain time-out) received enough replies to start the interaction, its starts executing it. The coordinator sends a message to Peer B which solves the constraint by asking the user (using a visualizer showing the constraint to the user). The word is sent back to the coordinator which continues parsing the IM and reaches a constraint that must be satisfied by the dictionary role to give the word definition. The coordinator sends the constraint to Peer C which solved it and returns the definition in a message. The coordinator continues parsing and finds a constraint in which the querier role must show the user the word definition. It sends Peer B a message with this constraint and it is solved by showing the query results to the user. The IM is finished at this point, so the coordinator sends a message to each peer so they can stop the OKC instances.

As said, this example demonstrates the functionality of the system, but it is very simple. The interface presented is only one of the many possible interfaces, because we have designed the architecture to be as independent as possible from the user presentation system.

3.5 Other Examples

Some interesting examples can be made within the trade domain, like an interaction model for a transaction of goods. Somebody may publish an IM that contains the process-flow between a seller, a buyer and a payment service. Peers can subscribe themselves to these roles and when all roles are instantiated the interaction starts. The *Coordinator* initiates the interaction and coordinates it. Especially in this case, all role-players may want to have a trustworthy controller, and can specify the requirements for a coordinator when subscribing to an OKC.

Another example comes from a case study that we undertook in the bio-informatics domain [22]. In that paper we present a system that can be used to analyse real data of relevance to the structural bio-informatics community where comparative models of yeast protein structures from different resources are analysed for consistency between them. The interaction model described in that paper, written in the LCC language, describes the interaction between the roles of data collector, receiver and source, that together perform the task.

4 Summary

Much of the information that might be accessed in semantic webs is accessible through complex programs (web-services, sensors, *etc.*) that may interact in sophisticated ways. Composition guided simply by specifications of programs' input-output behaviours is insufficient to obtain reliable aggregate performance - hence the recognised need for process models to specify the interactions required between programs. These interaction models, however, are traditionally viewed as a consequence of service composition rather than as the focal point for facilitating composition. We have described an operational system that uses models of interaction as the focus for knowledge exchange. Our implementation adopts a peer to peer architecture, thus making minimal assumptions about centralisation of knowledge sources of interaction control. The direct contribution of this paper is to present the first operational system of this kind. The secondary contribution of this paper is to provide a new angle on service orchestration and ontology matching that re-interprets traditional methods for these tasks in a dynamic context.

Acknowledgements. This work has been supported by the FP6 OpenKnowledge project[2]. A. Perreau de Pinninck is supported by a CSIC predoctoral fellowship under the I3P program, which is partially funded by the European Social Fund.

References

1. Akahani, J., Hiramatsu, K., Kogure, K.: Coordinating Heterogeneous Information Services based On Approximate Ontology Translation. In: AA MAS 2002. First International Joint Conference on Autonomous Agents & Multiagent Systems (2002)
2. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business process execution language for web services, version 1.0. Technical report (2004)
3. Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., Weerawarana, S.: Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI. IEEE Internet Computing 6(2), 86–93 (2002)
4. de Pinninck, A.P., Dupplaw, D., Kotoulas, S., Siebes, R.: The openknowledge kernel. In: Proceedings of the IX CESSE conference, Vienna, Austria (2007)
5. Meredith, G., Weerawarana, S., Christensen, E., Curbera, F.: Web services description language (wsdl) 1.1. Technical report (2001)
6. Foster, I., Kesselman, C.: The grid: blueprint for a new computing infrastructure. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1998)
7. Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J., Vassalos, V., Widom, J.: The TSIMMIS Approach to Mediation: Data Models and Languages. Journal of Intelligent Information Systems 8(2), 117–132 (1997)

[2] http://www.openk.org/

8. Haller, A., Cimpian, E., Mocan, A., Oren, E., Bussler, C.: WSMX-a semantic service-oriented architecture. In: Proceedings IEEE International Conference on Web Services, 2005. ICWS 2005, pp. 321–328. IEEE Computer Society Press, Los Alamitos (2005)
9. Kotoulas, S., Siebes, R.: Adaptive routing in structured peer-to-peer overlays. In: 3rd Intl. IEEE workshop on Collaborative Service-oriented P2P Information Systems (COPS workshop at WETICE07), Paris, France, IEEE Computer Society Press, Los Alamitos (2007)
10. Lenzerini, M.: Data integration: a theoretical perspective. In: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 233–246. ACM Press, New York (2002)
11. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., et al.: OWL-S: Semantic Markup for Web Services. W3C Member Submission 22 (2004)
12. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.: Importing the Semantic Web in UDDI. Web Services, E-Business and Semantic Web Workshop (2002)
13. Rao, A.S., Georgeff, M.P.: Modeling rational agents with a BDI-architecture. Readings in agent, 317–328 (1997)
14. Robertson, D.: A lightweight coordination calculus for agent systems. In: Leite, J.A., Omicini, A., Torroni, P., Yolum, p. (eds.) DALT 2004. LNCS (LNAI), vol. 3476, pp. 183–197. Springer, Heidelberg (2005)
15. Shvaiko, P., Euzenat, J.: A survey of schema-based matching approaches. Journal on Data Semantics IV, 146–171 (2005)
16. Siebes, R., Kotoulas, S.: proute: Peer selection using shared term similarity matrices. Web Intelligence and Agent Systems 5(1), 89–107 (2007)
17. Sivashanmugam, K., Verma, K., Sheth, A., Miller, J.: Adding Semantics to Web Services Standards. In: Proceedings of the International Conference on Web Services, pp. 395–401 (2003)
18. Stevens, R., Robinson, A., Goble, C.A.: mygrid: Personalised bioinformatics on the information grid. In: proceedings of 11th International Conference on Intelligent Systems in Molecular Biology, Brisbane, Australia (2003)
19. Sycara, K.P., Paolucci, M., Ankolekar, A., Srinivasan, N.: Automated discovery, interaction and composition of semantic web services. J. Web Sem. 1(1), 27–46 (2003)
20. ten Teije, A., van Harmelen, F., Wielinga, B.: Configuration of web services as parametric design
21. Traverso, P., Pistore, M.: Automated Composition of Semantic Web Services into Executable Processes. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, Springer, Heidelberg (2004)
22. Gerloff, D., Sharman, J., Quang, X., Walton, C., Robertson, D.: Peer to Peer Experimentation in Protein Structure Prediction: an Architecture, Experiment and Initial Results. In: International Workshop on Distributed, High-performance and Grid Computing in Computational Biology, Eilat, Israel (2007)