International Journal of Electrical, Electronic and Communication Sciences
ISSN: 2517-9438
Vol:1, No:3, 2007

The OpenKnowledge Kernel

Adrian Perreau de Pinninck, David Dupplaw, Spyros Kotoulas and Ronny Siebes

Abstract**—** Web services are pieces of software that can be invoked via a standardized protocol. They can be combined via formalized taskflow languages. The OpenKnowledge system is a fully distributed system using P2P technology, that allows users to publish these taskflows, and programmers to register their web services or publish implementations of them, for the roles described in these workflows. Besides this, the system offers the functionality to select a peer that could coordinate such an interaction model and inform web services when it is their 'turn'. In this paper we describe the architecture and implementation of the OpenKnowledge Kernel which provides the core functionality of the OpenKnowledge system.

Keywords**—** Architecture,P2P,Web Services,Semantic Web

I. Introduction

Web services are pieces of software which functionality can be invoked via standardized web interfaces. Combining these services leads to service interactions, which are specified as interaction models. In other words, interaction models contain roles and the execution flow between them. Web-services are not bound to specific interaction models; for example, a credit-card service can be used for a hotel-booking task or to buy music. From a user's perspective, the services are only a means to fullfill a task. The re-use of these interactions and decoupling them from the services that play the roles in the interaction is the novel contribution of the OpenKnowledge System (OK system). This 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* enabling the base functionality to find these interactions and the code or peers that enable to run the services. More precisely, the system is focused on efficiently sharing and finding these formally described *interaction models (IMs)* together with pointers to either the code for the services or peers that can execute the services. We call these services *OK components (OKCs)*. The IMs together with the OKCs are efficiently stored and retrieved in a P2P network. Besides this, due to the fact that the tasks are formally described, the OK system offers the functionality to coordinate a task by controlling the process flow between OKCs, (i.e., by executing the IM, selecting OKCs to fulfill a role, finding alternative OKCs in case of failure, and making sure the IM is followed by all OKCs). The users can *publish* IMs, write interfaces to services, and subscribe these interfaces to play roles in the IMs. The system helps these users by providing tools to ease re-use of existing IMs or by helping connect two services via mappings in case the output of one does not match the input of the other.

In [1] a detailed overview of the system requirements, assumptions and motivations together with a comparison to other systems and research is given. For a short summary of the related work, we refer to section VI. The OK system's most important requirements are ease of use, openness of sharing, and re-use of knowledge about interactions, which are the novel contribution of our approach. We assume that users are willing to provide knowledge about tasks. We think that this is a realistic assumption since community-based annotation systems like FlickR[1] and Youtube[2] have become quite popular. Also, we think it is reasonable to assume that a programmer of a web service wants it to be used and is willing to spend some time to write an IM with annotations if it would increase its probability of being used. Our goal of making our system user-friendly also holds for the programmers that contribute the IMs and OKCs by using past community knowledge. This knowledge is made up of interfaces to services, formal and semantically annotated task descriptions, mapping information, and feedback information for calculating reputations. Other approaches also pursue similar goals to Open Knowledge: sharing, searching and invoking services in a distributed and scalable way, for more information about them refer to section VI.

In this paper we define the OK Kernel. We first start with the architecture in the next section. Section III describes the services used by peers to interact with other peers. Section IV describes the protocols used by peers to interact with each other, how they are started and how they are executed. Section V gives an example to show how the system may be used. We conclude with related work in section VI and the summary and future work in section VII.

II. Architecture

The OpenKnowledge system has a P2P architecture where each autonomous peer shares a common piece of software that we call the *OK Kernel*. In this section we introduce the kernel's architecture, which will be described in more detail in the following subsections (see Figures 1 and 2). We follow a bottom up approach, starting from the basic building blocks, namely *OpenKnowledge Components* (OKCs) and *Interaction*

Manuscript received March 9, 2007. This work is supported by the OpenKnowledge Specific Targeted Research Project (STREP), which is funded by the European Commission under contract number FP6-027253. The OpenKnowledge STREP comprises the Universities of Edinburgh, Southampton, and Trento, the Open University, the Vrije University of Amsterdam, and the Spanish National Research Council, CSIC.

Adrian Perreau de Pinninck is with the Artificial Intelligence Research Institute, IIIA, CSIC, in Barcelona, Spain. He is supported by a CSIC predoctoral fellowship under the I3P program, which is partially funded by the European Social Fund (e-mail: adrianp@iiia.csic.es).

David Dupplaw is with the IAM Group of the University of Southampton, UK (e-mail: dpd@ecs.soton.ac.uk).

Spyros Kotoulas and Ronny Siebes are with the KR&R group at the section Artificial Intelligence from the Vrije Universiteit, Amsterdam, NL (e-mail: {kot,ronny}@few.vu.nl).

[1]http://www.flickr.com
[2]http://www.youtube.com

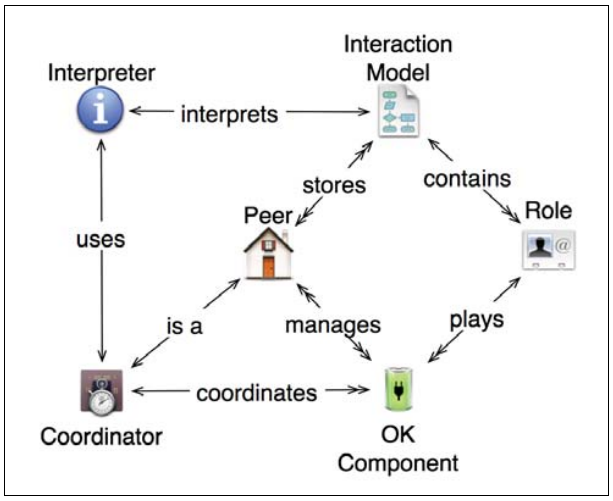

Fig. 1. Open Knowledge Architecture

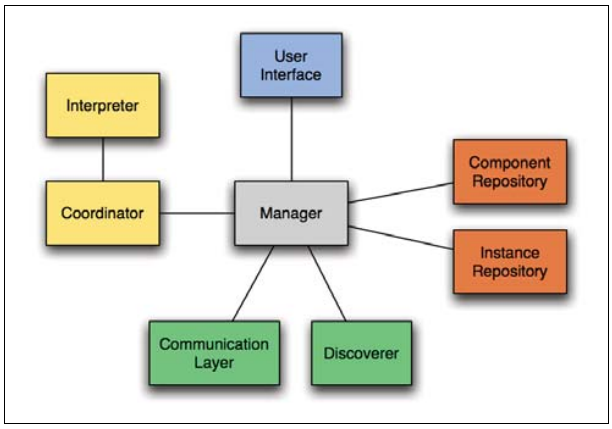

Fig. 2. Peer Modules

Models (IMs). An IM is a formal specification written in a language devised for this purpose (e.g., LCC [2] or BPEL[3]). IMs contain roles and define the interactions between them. Roles are implemented by OKCs. An OKC is comparable to a web service: it has an implementation and a standardized way to describe functionality. OKCs are mobile and are stored locally in *OKC repositories*, and distributedly via the DTS (see section III-A). Once they are being executed as part of an IM, the OKC instances are stored in *instance repositories*. Each peer manages the OKCs it has stored locally and can also act as a coordinator of an interaction between OKCs. OKCs communicate with the coordinator via the *communication layer*. A *user-interface* is provided to access the basic OK functionality by the user; search for IMs, download OKCs, and subscribe to roles. The *control manager* provides execution control over the peer's modules.

Besides these, there is a set of elements that are not essential for every peer but are essential for the system as a whole. Therefore, a subset of the OK peers need to execute them. Some peers, acting as *coordinators*, need to interpret the IMs and coordinate the communication between OKCs. Furthermore, some services are provided for peers to aid in the interaction process. The Discovery and Team formation Service (DTS) stores IMs, OKCs, and their subscriptions. The Trust and Reputation Service (TRS) is used to gather information about other peers in order to guide the user in choosing interaction partners. The Mapping Service (MS) is used by peers when interacting with each other to aid in mutual understanding.

A. Interaction Models (IMs)

OKCs on the system may be seen as passively providing functionality, just like a web service. When called, they process data and produce an output. If we were to write the specifications for the inputs and outputs we would end up with a semantic web service description that would allow OKCs to be executed on some data. However, any use of these OKCs in some larger problem remains undefined. An orchestration of two or more OKCs is useful in higher-level tasks and we define this through an interaction specification. However, from the types of inputs and outputs of a service, no inference can be made of how to sensibly orchestrate the values that pass between them; the coordination rules of the interaction provide such a formalisation of this orchestration. To fully define specific applications in which these OKCs may be activated we need to define both the interaction specification and coordination rules. Together, they create an IM by which we can orchestrate OKCs providing services that fulfil one of its roles.

IMs specify message passing interactions, with constraints on their execution. Messages are sent or received by peers if any constraints on the send or receive actions can be satisfied. Pragmatically, this means that all constraints are functional calls that will attempt to satisfy the constraint on the message operation. If constraints can be satisfied then the interaction may go ahead, otherwise we must backtrack to find an alternative path in the IM whose constraints are satisfiable.

B. OpenKnowledge Components (OKCs)

OKCs play a role in an IM. Many different OKCs with the ability to play the same role in the same IM can coexist. It is up to the user to decide which one suits his needs best.

OKC's are made up of the following parts:

- *IM reference* - the identifier of the IM that the OKC can play.
- *Role* - the name of the role that the OKC can play.
- *Annotation* - (optional) information describing the OKC implementation.
- *Code* - the algorithms that solve the role constraints in the referenced IM.
- *Facade* - the name of the main class that acts as an entry point to the OKC Code.

OKCs are packaged into one file so that they can be moved around the network easily. The file contains the compiled code (optionally the source code too), and an XML file which

[3]http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

contains the IM reference, role, annotation, and facade (i.e., the OKC Description).

More information about an OKC may be gathered from the network. Services can be used to find out how many peers host the OKC, its reputation ratings, or type mappings (see section III).

C. OKC Repository

Each peer can store OKCs locally in the OKC repository. A user is able to access the OKCs stored in it and use them to interact with other peers. OKCs downloaded from the OpenKnowledge network, and OKCs the user has implemented can be stored in the repository.

The simplest possible repository will contain the functionality to store new OKCs, retrieve an OKC previously stored using its ID, and list the currently stored OKCs. Advanced implementations may have querying mechanisms to search through the repository, and tag OKCs with labels according to a user-defined label hierarchy. Searching, storing, and labeling is achieved via the user interface.

D. Instance Repository

When the user starts interacting through one of its OKCs, a new instance of the OKC is created that will play its role in the IM. A new instance needs to be created, since an OKC may be taking part in more than one interaction. The peer needs to manage all these instances, and this is accomplished through the instance repository.

E. Communication Layer

The communication layer describes the set of network protocols used by the system objects (OKCs and Coordinators) to communicate with each other. We have designed the communication layer to send messages asynchronously between peers using TCP/IP as the transport protocol, which is widely used over the internet and guarantees message delivery.

F. Control Manager

The control manager is the glue that pulls the peer parts together. The user interacts with the peer through the UI. Its events are handled by the control manager which delegates the request to the modules that can satisfy them. The control manager follows the mediator pattern [3], whenever one module needs to use another module's functionality, it asks the control manager which delegates to the appropriate module.

The control manager drives the system protocols (see section IV), and accesses the different services offered by the network. It initializes all the modules and services when the peer is booted, and shuts them down when the peer is stopped.

G. Coordinators and Interpreters

Peers in the network may subscribe themselves to play the role of coordinator (see section IV). A coordinator is a peer in the network that will execute an IM locally, interpreting it, storing the state and simulating all message passing within itself. Since, it is unable to provide specific functionality that the IM requires (i.e., solve constraints), the coordinator will contact the OKCs playing the IM roles to provide this functionality. When the interpreter comes across a constraint, the coordinator intercepts it and delivers it to a peer who has subscribed to play the role that provides that functionality.

Delegating IM execution from each peer in the interaction to a single coordinator peer, provides us with a greater ability to execute potentially conflicting constraints in a parallel fashion, because the coordinator retains an overall vision of the interaction state. Passing the protocol from peer to peer during message delivery makes such parallel interactions very difficult to synchronise.

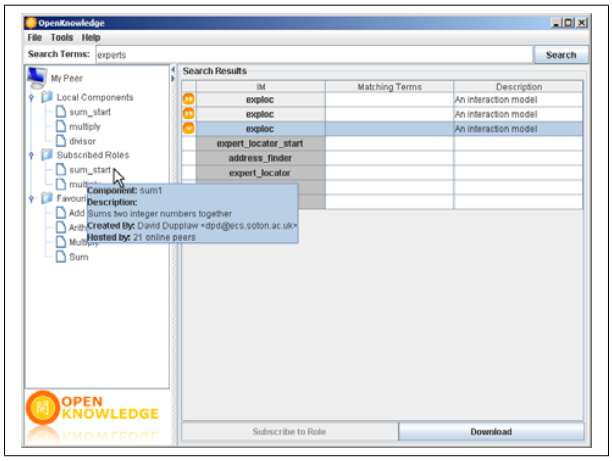

Fig. 3. Basic User Interface

H. Visualization - User interface

OK peers are provided with a basic user interface (see Figure 3), however it is not limited to such an interface. The interface is mainly used as an event generator, these events are handled by the system's control manager. The UI provides the means to search for IMs, download OKCs, and subscribe to roles.

A number of extra features are available when an IM is being executed; interaction state visualization, and constraint satisfaction by users. The use of coordinators makes visualising the current interaction state simple. Since the coordinator is in contact with all the interacting peers, it can send the interaction state to those that wish to visualize it. Also, when an OKC is unable to solve a constraint it can fire a GUI to procure a result directly from a user, using the visualization engine.

III. Services

Not everything happens via OKCs, a set of *services* is also part of the OK system. The *discovery and team formation service* is an essential service, meaning that the system cannot operate without it. Other services, namely the *mapping service* and the *trust and reputation service* are in the focus of our research, but are optional.

A. Discovery and Team formation Service (DTS)

Much of the functionality of the OK system relies on the DTS, its main responsibilities being the following:

- *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.

B. Mapping Service (MS)

An OKC needs to understand the other OKCs it is interacting with. Chances are that an OKC will not interact with all the other OKCs, therefore defining an a priori ontology seems unreasonable, given the complexity of the task. Furthermore, we want to achieve low entry cost, therefore, matching of one OKC's terms to another must be done at runtime. The MS's aim is to aid in this runtime process.

The MS is used in the searching and the interaction processes. When searching for IMs and OKCs it is used to map the text in their annotations to the query. When interacting, an OKC can use the MS to map those terms that another OKC is sending to its own terms. The MS taps into the information gathered from the system use, to provide community-supported mappings.

C. Trust and Reputation Service (TRS)

Since we are dealing with a completely decentralized system and flexible interactions, a service to maintain trust relationships greatly benefits the range of applications suitable for OpenKnowledge. To this end, a combination of personal preferences and community past experience needs to be taken into consideration when choosing whom to accept as interaction partner. Note, that although the policies governing this can be complex, our architecture is independent of the TRS implementation.

All that is needed from the user by the TRS is that it rates the interactions with other peers. This action is optional, but it is feasible to assume enough users will be willing to rate interactions seeing how other community-based rating systems such as FlickR and YouTube are being used. In return, users will be able to query the TRS in order to retrieve reputation data about other OKCs and peers with which (s)he might have to interact.

IV. OpenKnowledge Protocols

The OpenKnowledge peers interact with each other through different protocols; discovery, team formation, and interaction. In this section we will describe each one of them.

A. Discovery

The discovery protocol is used by peers to find information that other peers have published in the network. Therefore this protocol consists of the following parts: publishing, searching, and downloading.

A user, that has developed an IM or an OKC in his machine, can publish them to the OpenKnowledge network if he wants to share them with other users. OKCs that have not been published can interact with others. On the other hand an IM that is not published cannot be used. When publishing the user provides annotations associated with the IM or OKC so that other users can find them. The DTS is in charge of storing the published IMs and OKCs

Users can search for IMs and OKCs that other users have published by querying the DTS. The query is made up of a list of keywords. The DTS will search for IMs and OKCs whose annotation matches the user's query, the MS can be used in the process of matching the query to the annotations. The DTS will return a list of matching IMs or OKC descriptions[4] which are shown to the user through the GUI. The user may select one of the objects from the list and download it from the DTS, if the user chooses an OKC the whole package with description and code is retrieved.

B. Team Formation

Peers that want to interact with other peers must play a role in an IM. In order to achieve this the peer subscribes an OKC in its local OKC repository to the DTS. The user can add annotations associated to the subscription giving further information about the OKCs capabilities in the interaction, and partner restrictions. Furthermore, peers can also subscribe themselves to act as interaction coordinators to the DTS. Once enough peers have subscribed to play all the obligatory roles of an IM, a team can be formed. The DTS collects all the subscriptions and appoints a *coordinator* which is responsible for informing peers about the participation of their OKC in the team, carrying out **participant finalization**.

Participant finalization is an additional negotiation step in which OKCs are expected to commit themselves to participate. The coordinator sends a message to each peer with a list of the other peers that have subscribed. Each OKC then returns a message saying if it commits to play the subscribed role or not. If so, a new instance of the OKC is created and the interaction can start, the coordinator informs the DTS so that the used subscriptions are taken into account. If not enough peers commit, the coordinator must roll back the team formation process. Sends a cancel interaction message to each peer, which will then destroy the OKC instance.

[4]The code is not returned since it is only downloaded on demand with the whole OKC.

When the peer receives a message from a coordinator to commit to an interaction, it can check the other peers' reputation via the TRS. This information can be used to decide whether it want to commit to the current interaction or not. In some special cases (e.g., electronic marketplaces), deciding to commit can include step in which the user is asked to confirm.

C. Interaction

The protocol starts with the coordinator using the interpreter to parse the IM it has been handed by the DTS. When the interpreter reaches a constraint which must be solved by one of the interacting OKCs, the *constraint solving protocol* starts. The coordinator sends a message to the OKC that is to solve the constraint. This message contains the interaction state, the constraint to be solved, and the parameters associated to the constraint. When the OKC receives this message it tries to solve the constraint. If it does it returns a message with the constraint return values and the modified interaction state. When the coordinator receives the message with the solved constraint, it merges the new state with the global interaction state. If the OKC is not capable of solving the constraint (e.g., due to problems with ontology mapping, or to unforeseen problems with resource availability), it sends the coordinator a message indicating this, and the coordinator will search for another path in the IM that can be satisfied.

V. Example

We will now show through a simple example how the Open-Knowledge System works. We will show how a dictionary service can be created and used. The scenario that we describe in this section is made up of three peers, but many more could be added without changing the general idea. Each of the peers will also be part of the DTS and subscribe to act as coordinator. To simplify matters we will assume that these peers are always online, and the OK Kernel is being executed in each one of them.

Peer A develops an IM that describes an interaction between two roles, one being the *querier*, and the other the *dictionary*. The querier sends a word to the dictionary which returns its definition. It also creates an OKC that can play each of the roles. Peer A then publishes the IM and the querier OKC to the DTS giving annotations so that they can be easily found. The OKC Peer A has implemented to play the dictionary role can only give definitions to English words. Peer A subscribes this OKC and gives an annotation saying that it is an "English" dictionary.

Peer B wants to find a service that will allow it to find definitions to words in Spanish. It queries the DTS and finds an IM that suits its needs. It also finds an OKC which allows it to play the querier role, which it downloads to its local OKC repository. Peer B then subscribes the OKC to play with the restriction that it wants to find the definition for words in Spanish. At this point the DTS has enough subscriptions to start an interaction. But, since Peer B set a restriction that the subscription of Peer A does not satisfy, the interaction cannot start.

Peer C would like to offer a spanish dictionary service. It queries the DTS and finds an IM that suits its needs. Therefore, it creates an OKC that can play the dictionary role in the IM. Peer C then subscribes the OKC with an annotation saying that it is a Spanish dictionary. Now the DTS has enough subscriptions that (satisfy the restrictions) to play all the roles in the IM. It chooses one of the peers that has subscribed to play as coordinator and sends it the IM and the interacting OKC peers' addresses.

The coordinator sends a message to each peer asking them to commit to the interaction, with information about the other subscribed peers. Both Peer B and C receive the message and commit. Each of them creates an instance of the OKC for this interaction, and the coordinator notifies the DTS that the subscriptions have been used. The coordinator starts parsing the IM and it finds a constraint where the word to be defined should be returned. The coordinator sends a message to Peer B which solves the constraint by asking the user (using the visualizer). 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 using the visualizer to show 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.

VI. Related Work

Other approaches such as Web services, Grids, P2P, and MAS, share the goals of OK: writing, sharing, searching and executing task descriptions (IMs) and services (OKCs) in a distributed, user-friendly, and scalable way. Below we point out the key differences between OK and them:

- *Web services* are clearly distributed, but recruiting them for a work-flow is either static (i.e., fixed addresses in the work-flow description) or centrally organized like in a UDDI[4] registry for regular web-services and WSMX[5] for semantic Web services. Static pointers are a disadvantage because they are not flexible with regard to new peers that want to host the service or for those that do not want to host it anymore. A centralized repository, especially when it needs reasoning (e.g., Semantic Web services), is not scalable and is a single point of failure. In our system, we build a scalable distributed repository of IMs (DTS) together with pointers to (web) services or code. Although there are differences, we can use many of the technologies and methods like mappings, reasoning, versioning, and query relaxations in our system[6].
- *Grids* have the most overlap with our approach. The key difference is not the goal but the focus; which problems to solve, and the assumptions made. For example, the grid community focusses more on reliability, task scheduling, and distributed computing. We focus on making everything P2P, user-friendliness, community support and

knowledge re-use. The most related approaches to our work are ^{my}Grid [7] and OGSA[8]. Unfortunately, they assume a centralized infrastructure on which, for example, process flows are published and discovered which may cause scalability problems when applied in large scale. One other very interesting direction is distributing the control of the execution of a workflow to prevent single points of failure [9], [10]. Please recall that in our system, for each IM that is instantiated with peers that will play the roles, a coordinator is dynamically chosen from a pool that controls the IM. This means that if the coordinator fails, only the interactions the coordinator is orchestrating are lost, not the whole of the interactions being coordinated in the system.

- *P2P systems* are the most generic category. Many systems fall into this category, such as Seti@home (where peers act like drones), file-sharing systems (i.e., Kazaa and Gnutella), and messaging services (i.e., Jabber , and Skype). This field's limitation is its specific functionality (i.e., for each task there is a new application). We have designed a generic system where tasks are open and the peers in the system can execute it. For example, the Gnutella task could run on our system by writing the IM that describes the interactions between peers. Besides this, the P2P systems are mostly data oriented, where our system is service oriented.
- *Multi-agent systems* focus on complex behavior of a whole system or the agents within. Agents often have reasoning capabilities based on psychological models and show pro-active behavior. Individual agents or groups can perform tasks which are similar to our approach. The difference is that many systems still have centralized components, like matchmakers and brokers as InfoSleuth[11] or AMELI[12], and therefore could have scalability issues.

VII. Summary and Future Work

In this paper we have presented the motivation and the technical description the OpenKnowledge Kernel that offers the core functionality of the OK system. We defined the different protocols and services offered by it, how OKCs interact with each other, and the peer's local modules. We have left a lot of the implementation issues open to allow for multiple implementations. Our goal is to describe the different objects present in the Kernel and its core services and define the relation and communication between them. The novelty of the OK system lies in (1) the interaction centric approach, where interactions are published and efficiently stored in a P2P network, (2) decoupling interactions and roles from the services that execute these roles, (3) a distributed way of finding coordinators that coordinate IMs.

We have presented the first architecture version, and we are working on the next version that will improve it. For instance, the current version's coordinator is a bottleneck, and the single point of failure in a single interaction. We want to distribute the coordination task amongst different peers, so that if a coordinating peer goes offline, the interactions it is orchestrating are not suspended. We also want to offer all the current services and the coordinator as OKCs. Different implementations would be available, and users would choose which services to offer, and which to use.

In order to implement a pure OKC-based architecture, we must define how to create new IMs by using preexisting ones as sub-IMs. IMs would have references to other IMs, but this is just one part of the problem. We also need to design a mechanism through which sub-IMs can be coordinated, and how the coordinators at different levels of the IM keep whole interaction state consistent.

In the actual architecture there is a tight coupling between OKCs and IMs. To make the system more dynamic, we plan to allow OKCs to play roles in many IMs. This could be accomplished by allowing a list of IM references in an OKC, or by defining the OKC's capabilities and verifying real-time if the OKC is able to play a role in an IM. Also IMs are restricted to define reactive roles, because OKCs are only asked to solve constraints when the IM demands it. We also plan to allow OKCs to be pro-active bringing us closer to the multi-agent community.

References

[1] R. Siebes, D. Dupplaw, S. Kotoulas, A. P. de Pinninck, D. Roberston, and F. van Harmelen, "The functional description of the open-knowledge system," Open-knowledge consortium, Tech. Rep., 2006. [Online]. Available: http://www.cs.vu.nl/ ronny/work/okfunctional.pdf

[2] D. Robertson, "A lightweight coordination calculus for agent systems," *Lecture Notes in Computer Science - DALT*, vol. 3476, pp. 183–197, 2005.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of reusable object-oriented software*. Addison-Wesley, 1995.

[4] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the web services web: An introduction to soap, wsdl, and uddi," *IEEE Internet Computing*, no. 6(2), pp. 86–93, March 2002.

[5] P. Bouquet, J. Euzenat, E. Franconi, L. Serafini, G. Stamou, and S. Tessaris, "Overview and scope of wsmx," DERI, Ireland, Tech. Rep., 2005, http://www.wsmo.org/TR/d13/d13.0/v0.2/.

[6] G. Antoniou and F. van Harmelen, *A Semantic Web Primer*. MIT Press, 2004.

[7] R. Stevens, A. Robinson, and C. Goble, "^{my}Grid: personalised bioinformatics on the information grid," *Bioinformatics*, vol. 19, no. 1, pp. 302–304, 2003.

[8] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The physiology of the grid: An open grid services architecture for distributed systems integration," in *Open Grid Service Infrastructure WG, Global Grid Forum*, june 2002.

[9] D. Wodtke and G. Weikum, "A formal foundation for distributed workflow execution based on state charts," in *ICDT '97: Proceedings of the 6th International Conference on Database Theory*. London, UK: Springer-Verlag, 1997, pp. 230–246.

[10] W. Tan and Y. Fan, "Model fragmentation for distributed workflow execution: A petri net approach," in *Advanced Distributed Systems (LNCS 3563)*. Springer Berlin / Heidelberg: Springer-Verlag, 2005, pp. 207–214.

[11] M. Nodine, W. Bohrer, and A. H. H. Ngu, "Semantic brokering over dynamic heterogeneous data sources in infosleuth," in *Proceedings of the International Conference on Data Engineering*, 1999.

[12] M. Esteva, B. Rosell, J. A. Rodriguez-Aguilar, and J. L. Arcos, "Ameli: an agent-based middleware for electronic institutions," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, 2004.