THE UNIVERSITY *of* EDINBURGH

Peer-to-Peer, Multi-Agent Interaction Adapted to a Web Architecture

Xi Bai

Doctor of Philosophy

Centre for Intelligent Systems and their Applications

School of Informatics

University of Edinburgh

2013

Abstract

The Internet and Web have brought in a new era of information sharing and opened up countless opportunities for people to rethink and redefine communication. With the development of network-related technologies, a Client/Server architecture has become dominant in the application layer of the Internet. Nowadays network nodes are behind firewalls and Network Address Translations, and the centralised design of the Client/Server architecture limits communication between users on the client side. Achieving the conflicting goals of data privacy and data openness is difficult and in many cases the difficulty is compounded by the differing solutions adopted by different organisations and companies. Building a more decentralised or distributed environment for people to freely share their knowledge has become a pressing challenge and we need to understand how to adapt the pervasive Client/Server architecture to this more fluid environment.

This thesis describes a novel framework by which network nodes or humans can interact and share knowledge with each other through formal service-choreography specifications in a decentralised manner. The platform allows peers to publish, discover and (un)subscribe to those specifications in the form of Interaction Models (IMs). Peer groups can be dynamically formed and disbanded based on the interaction logs of peers. IMs are published in HTML documents as normal Web pages indexable by search engines and associated with lightweight annotations which semantically enhance the embedded IM elements and at the same time make IM publications comply with the Linked Data principles. The execution of IMs is decentralised on each peer via conventional Web browsers, potentially giving the system access to a very large user community. In this thesis, after developing a proof-of-concept implementation, we carry out case studies of the resulting functionality and evaluate the implementation across several metrics.

An increasing number of service providers have began to look for customers proactively, and we believe that in the near future we will not search for services but rather services will find us through our peer communities. Our approaches show how a peer-to-peer architecture for this purpose can be obtained on top of a conventional Client/Server Web infrastructure.

Acknowledgements

I have become really excited and joyful when I began to look over my whole PhD journey past and have been feeling so fortunate to have mentors, friends and family being extremely supportive of my work all along this rocky but wonderful road.

Words cannot say how sincere thankfulness I owe to my supervisor, Prof. Dave Robertson, who have guided me through the entire research of my PhD and the whole process of my thesis writing. Without his continual help, encouragement and inspiration, it would not have been possible for me to accomplish this work. I would like to thank Prof. Ewan Klein, who is more than a mentor to me but a real role model. Since I started my PhD, he has provided invaluable feedback and their insightfulness would not be found elsewhere. I would also like to thank my examiners, Dr. Paul Anderson and Dr. Leslie Carr, for providing insightful and inspiring feedback.

This thesis was co-funded by University of Edinburgh and Scottish Informatics and Computer Science Alliance (SICSA), and I would like to thank both organisations for their massive supports and generosities. I am so grateful to have the chance to visit Dr. Wamberto Vasconcelos from University of Aberdeen and also have him as my SICSA supervisor, who helped me to develop the original idea about OKBook. My sincere thanks also go to Romi Capdevila and people from MicroArt, Parc Científic de Barcelona, who welcomed me as a friend and helped me to finish project deliverables, and I would also like to thank FP7 Marie Curie Industry-Academia Partnerships and Pathways (IAPP) for their generosities during my visit. Special thanks go to Paul Adderley and Prof. Bob Fisher for their trusts and offers of project positions, which made me turn my hobby into the part-time work.

My PhD would not be this joyful without my colleagues and friends from School of Informatics, University of Edinburgh, and thank Dr. Jacques Fleuriot and all others for their help in my research and also encouraging me to outreach my university office life and banding together for fencing, bowling, pingpong, films, pizza, poker, warcraft and karaoke.

Lastly and most importantly, I would like to thank my parents, Baowu and Xiuyue, and my girlfriend, Jie for their heartfelt understanding, endless patience and continuous support whenever and wherever I need it.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Xi Bai*)

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In this chapter we introduce the concept of knowledge sharing in open, peer-to-peer systems and give an outline of the OpenKnowledge system[1] which is an EU-funded project, previously implemented and discussed in (Robertson et al., 2009), and also the reference architecture for this thesis. A key contribution of this thesis is to show how the approach taken in the reference architecture can be reconstructed in a Web architecture, so we also introduce this view along with its extension into the "Semantic Web" which is important in underpinning the approach of subsequent chapters.

Much of the research on knowledge sharing in the Semantic Web and Linked Data community has focused on the traditional view that the knowledge should be represented independently of the tasks in which it is used. The openness of the World Wide Web (WWW) succeeds in scaling to a global environment due to the network effect and low level of its participating cost while in an open knowledge sharing environment (anybody may join at any time at low individual cost), the traditional knowledge engineering does not scale due to the cost of providing absolute service semantics, which makes knowledge of services grow rapidly as more of them participate. However, our choice of task can strongly influence the way in which we represent knowledge, so recent research has studied the relationship between the way knowledge is expressed and the context of that knowledge in the tasks we wish to perform (Robertson et al., 2009). One reaction to this is to focus on the tasks themselves — we model these tasks in a formal specification language and share these tasks while at the same time sharing the knowledge associated with them (so that knowledge is always understood in the

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

context of the tasks in which it has been used). Systems such as OpenKnowledge have demonstrate how this can be done on peer-to-peer infrastructures, where peers[2] share executable specifications of tasks and a kernel system on each peer enables discovery, selection, enactment, ontology alignment and interactions with other peers. Although the OpenKnowledge system demonstrated how contextualisation by task could make this sort of task-based knowledge sharing possible, it relied on bespoke systems for peer-to-peer knowledge sharing (including the peer-to-peer infrastructure, the kernel system and associated discovery and ontology alignment systems). We demonstrate that a different (and arguably less limiting) architecture is possible in which task specifications are documents on the Web, marked up with metadata that allows their discovery using established Semantic Web techniques. Peer community formation can benefit from harnessing process-oriented programming with lightweight semantic enrichment. OpenKnowledge depends on a so-called Interaction Model (IM) expressed in the Lightweight Coordination Calculus (LCC), which is an executable specification based on π-calculus (Milner, 1999) and expressed in a Prolog-like syntax. In order to achieve more flexible peer interactions, we have developed eXtended Lightweight Coordination Calculus (XLCC) as an extended version of LCC in this thesis.

The resulting lightweight declarative language can be used for describing task specifications and has been adopted in this thesis to describe and serialise peers' interactive processes. A browser-focused application allows documents in XLCC also to be enacted as interaction processes between network nodes via the Extensible Messaging and Presence Protocol (XMPP) (Saint-Andre, 2004). This application removes the need for a peer-to-peer infrastructure since, only Transmission Control Protocol (TCP) and Hypertext Transfer Protocol (HTTP) are needed to support communication between peers. As a result, this architecture allows knowledge sharing to be viewed as an extension of the traditional Semantic Web infrastructure rather than as a radical departure from it. To enable this, however, requires a fundamental change in perspective for process-based knowledge sharing and a change in the basic mechanisms used to automate the system.

IMs describing interactive processes are *de facto* protocol specifications. In order to make these specifications adopted by all process participants, who have various requirements and offers, as widely as possible, a generic shared serialisation strategy is

[2]We follow the example of the OpenKnowledge system in using the term *peer* (rather than *agent*) to focus on reactive behaviours of participants within interaction.

needed and this is the place where declarative languages such as LCC come to play. On the other hand, sometimes the heterogeneity of process participants will be unfortunately ignored or treated blindly equivalently when a single declarative language is applied. In order to bridge the gap between the declarativity of the task description language and the knowledge heterogeneity of participating performers, and at the same time, balance the generality and the specificity of descriptions, methods and techniques derived from the Semantic Web and Linked Data community have been employed in this thesis to fertilise the semantic enhancement to the specifications which guide peers to interact with one another. The interpreter deployed on each peer is able to know the language semantics of LCC. The operational semantics (Scott, 1970) of LCC has been elaborated in (Robertson, 2004) and makes LCC a compact and lightweight language for both IM creators and IM stakeholders (Figure 3.8 in Chapter 3 gives an exemplary IM in LCC). However, when peers are subscribed to a specific IM which will be interpreted in a decentralised environment, not only operational semantics is needed to guarantee the participants' rational behaviours, but also context-specific annotations are necessary and can be applied within the process of discovering which IM can actually meet requirements of peers. Semantics (also known as metadata and ontologies) behind Web resources derived from the Semantic Web community can be harnessed as supplementary metadata and made use of along with operational semantics, in order to improve the knowledge representation of task-dedicated interactive processes. Therefore, inside an IM are LCC elements attached with annotations (in the form of Uniform Resource Identifiers (URIs)), which may be changed by IM publishers as time goes by. Based on situation calculus (a representation for first-order logic planning) (Reiter, 1991), each annotation (attached to a particular LCC element) here can be recognised as a *fluent* (a.k.a, properties of the world) since it describes part of the context in which the LCC element applies. The remainder of this chapter introduces the OpenKnowledge system as a peer-to-peer knowledge sharing platform, especially in the vision of the Semantic Web which raises the potential to evolve OpenKnowledge into a Web-oriented architecture in order to achieve peer communities.

1.1 Overview of OpenKnowledge

Interactions between different network nodes of service-oriented computational systems are usually dedicated to sharing services or more broadly speaking, functional-

ities. Many terms have been coined for these nodes, such as agents, peers and more recently, Web Services (WSs). Online agents play different roles during the interaction: for example, sometime they are customers, looking for particular resources from others and sometime they are providers, offering their resources to others insofar as some particular conditions are satisfied. Moreover, the role(s) played by a given network node during the process of fulfilling a task can be either static (sticked to a single role) or dynamic (switching between different roles), subject to the requirements of the node.

Generally speaking, interactions between nodes can be achieved in a centralised or decentralised (or even distributed manner). In the former case, which has been referred as *Orchestration* (Papazoglou, 2003), the interaction will be controlled centrally and different participants are gathered at design time and focus on their own needs and offers. In the latter case, which has been referred as *Choreography* (Peltz, 2003), the interaction will be constrained in terms of a piece of protocol, in which each participant is aware of the role(s) it will play during the interaction and since there is no central controller, the whole interaction process will proceed in a cooperative way. A notion of conformance between orchestration and choreography was proposed based on a so-called bisimulation and has been used to state when an orchestrated system conforms to a given choreography system but not all orchestrations have conformable choreographies (Busi et al., 2005). All in all, orchestration and choreography can be regarded as two different and sometimes alternative perspectives from which systems or services can be modelled and managed.

Targeting service choreography, the OpenKnowledge project has delivered an open platform which enables us to create a distributed peer-to-peer ecosystem based on shared interaction protocols encapsulated in lightweight IMs. Compared to an service implementation driven by orchestration, the OpenKnowledge platform can scale better with the growing complexity of systems (Robertson et al., 2009). Conventional service composition is carried out by connecting the output of a service to the input of another, which is built on an assumption that if the semantics of services could be precisely defined (e.g., in OWL-S (Martin et al., 2004) or WSDL-S (Akkiraju, 2005), etc.), the service composition can be done freely when the local semantics of each service is preserved. However, in a decentralised environment, even if this assumption were perfectly met, the traditional composition of services still cannot scale due to the heterogeneity of jointly employed semantics descriptions preserved by the partic-

ipants involved in the composition. The OpenKnowledge system addresses the above ill founded assumption and instead of requiring universally accepted semantics across all service interfaces, this system only requires a shared IM in which semantics is internally consistent, which is similar with the use in human affairs of protocols or contracts. Under this regime, only the semantics related to a specific IM will be concerned and standardised by service providers involved in the composition.

The OpenKnowledge system was designed and focused on problems including ontology alignment, coalition formation, outcome prediction, maintaining shared knowledge, respecting local constraints and relating interaction to process requirements. It turns the control of interaction into a declarative programming problem with LCC and those problems are hard to solve with traditional declarative or agent-oriented programming. Each user needs to download a piece of software and install it on his/her own machine. This software provides a platform User Interface (UI) for users to publish and test IMs and also has a file system to manage all the searchable data, as shown in *OpenKnowledge Manual*[3] and repeated in Figure 1.1. By typing keywords, users can look for IMs which may meet their requirements and perhaps more than one IM will be returned and ranked. Users choose the IM to subscribe to based on the recommendation and as soon as each role defined in the IM is filled by a particular peer, the OpenKnowledge kernel will randomly select a peer as the coordinator and forward all the subscription information to it. The coordinator is equipped with a LCC interpreter and behaves like a trusted middle-man. Therefore, the coordinator knows all the states of each peer during the interaction and can send or receive messages on behalf of each involved peer. If some peers can not behave based on their obligation (e.g., they can not satisfy the predefined constraints), the coordinator will terminate the interaction and recommend other substitute peers if then exist. After finishing the execution of the IM, the LCC interpreter informs the coordinator, which will forward the results, including statistics on participating peers, to the OpenKnowledge kernel. Finally, the kernel will update the local database with statistical information about executed IMs and participating peers.

[3] `http://groups.inf.ed.ac.uk/OK/download/manual.pdf`

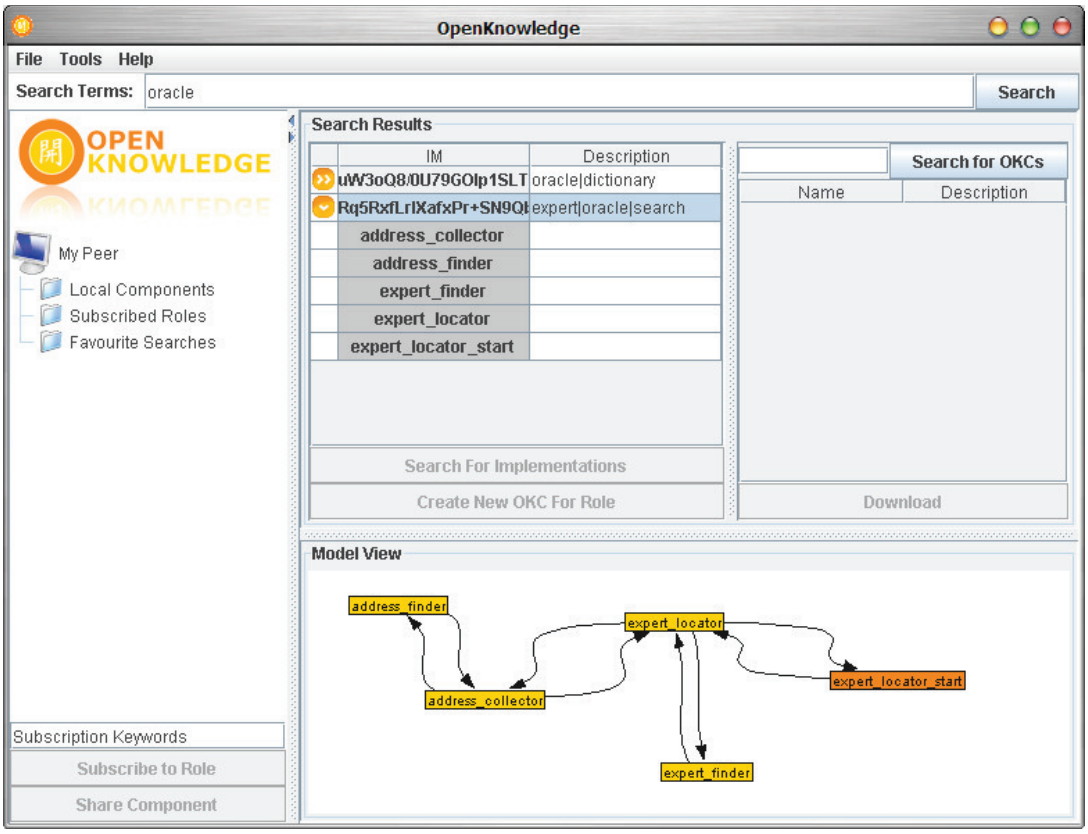


Figure 1.1: OpenKnowledge client-side UI

1.2 Web-Oriented Infrastructure for Knowledge Sharing

The WWW has become a significant infrastructure for knowledge sharing and interactive communication. Research in this area remains active and has spawned several sub-disciplines: communication protocols, data exchange formats, information retrieval and Web development. These categories are not independent of one another and so many research topics cross more than one of them. An increasing number of professional content publishers or amateur users have began to publish their content on the Web with various systems. Due to the exploration of Web information, search engines and recommendation systems have been designed and implemented to assist in people's daily life. With the development of Web searching technologies, crawling, indexing and searching have become the typical functionalities a modern search engine needs to have. The crawler is in charge of constantly gathering Web documents which are later indexed for filtering out and structuring useful information. Search engines provide users with their searching functionality via various UIs and, by matching

user's query against with their indices, they present the search results in a ranked list with the most relevant result at the top. The target resources that search engines aim to deal with are mainly Web documents with diverse formats and WS description can definitely be one kind of indexable documents. Motivated by this, this thesis will showcase potentials of generic search engines to function as hubs of service discovery.

In order to improve information acquisition and make them more precise and efficient, data exchange formats (bottom-up) and information extraction (top-down) have emerged as hot research topics in the last two decades. Meanwhile, an increasing number of recommended content publishing formats and information search techniques have been proposed for tackling the "information overload" problem. Following that, the concept of the Semantic Web was proposed by Tim Berners-Lee, James Hendler and Ora Lassila in *Scientific American* magazine in 2001 (Berners-Lee et al., 2001) and a use case scenario was also given in which a bundle of agents collaborate with each other in order to fulfil a complex task by exchanging understanding the information containing machine-readable semantics on the Web. The Semantic Web has been defined as an extension of the current Web (Document Web), rather than a competing invention. The original "layer cake" (Berners-Lee, 2000) to the Semantic Web technologies is illustrated in Figure 1.2, while its more recent version (Hendler, 2009) is illustrated in Figure 1.3. Several changes have been made, especially in those layers located between the URI layer and the Logic layer (e.g., SPARQL Protocol and RDF Query Language (SPARQL), Resource Description Framework in Attributes (RDFa) and Semantic Web Service (SWS)). This update also suggests that within the first decade since the Semantic Web was born, the progress on either the theoretical foundation has not been elaborated so intensively as other areas (e.g, Information Retrieval and Machine Learning). Another term used as a twin sister of "Semantic Web" is "Web of Data" (it defines "Semantic Web" also by Tim Berners-Lee as "a web of data that can be processed directly and indirectly by machine" (Berners-Lee et al., 1999)) has been used for emphasising the data-interweaving feature of the Semantic Web. "Linked Data" (Berners-Lee, 2006) is another term which is used most recently for describing the vision of the Web of data and many folks believe this is an attempt to rebrand the Semantic Web. A literal comparison given by Tim Berners-Lee himself is "Linked Data is the Semantic Web done right". A screenshot of the growing Linking Open Data (LOD) cloud diagram[4] which shows the datasets published ("openly accessible from a

[4]The Linking Open Data cloud diagram, available at `http://richard.cyganiak.de/2007/10/lod/`

network point of view") in Linked Data format, is illustrated in Figure 1.4. The Linked Data effort emphasises the technical aspects of linking data across the Web, including identifying resources using URI and employing the Resource Description Framework (RDF) data model, etc. in an attempt to focus on a practical subset of Semantic Web technology. All in all, the stack of semantic technology and related research plays an increasingly important role in the modern territory of knowledge representation and information sharing, and this thesis also further explores the power of the Semantic Web and harness it to utilise our approaches.

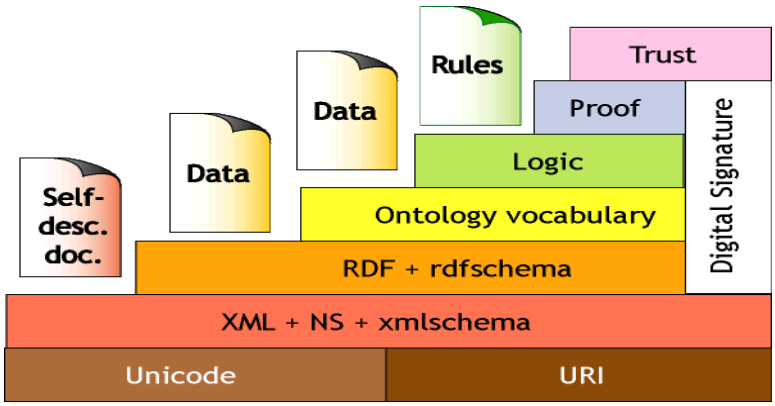


Figure 1.2: Semantic Web layer cake as of 2001

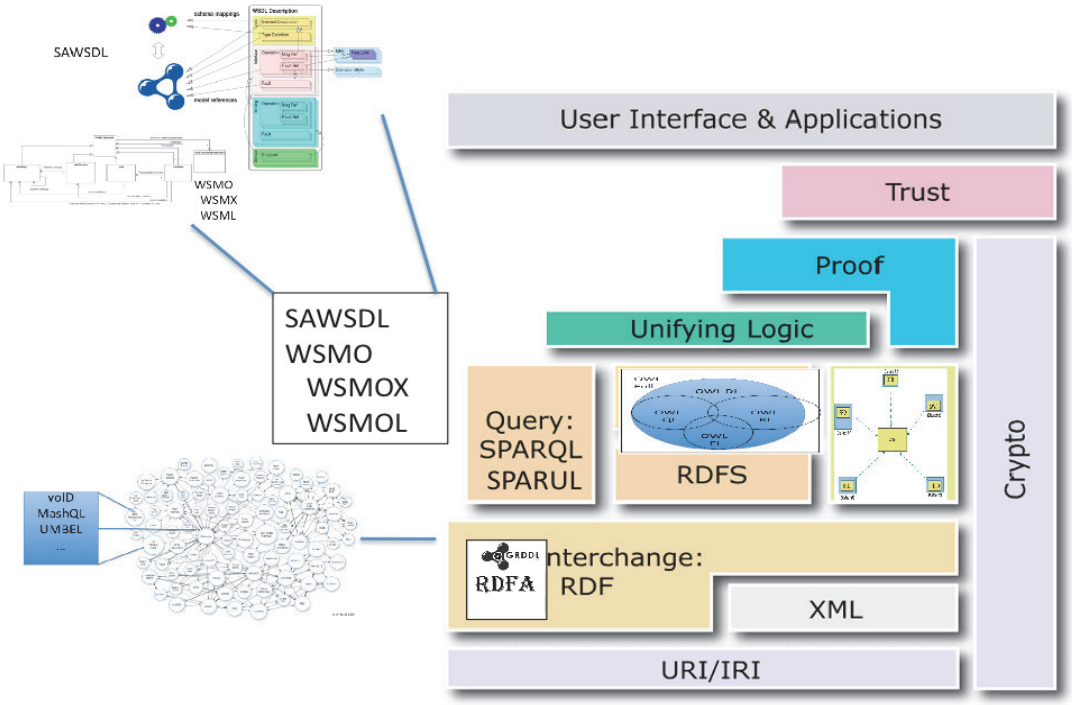


Figure 1.3: Semantic Web layer cake as of 2009

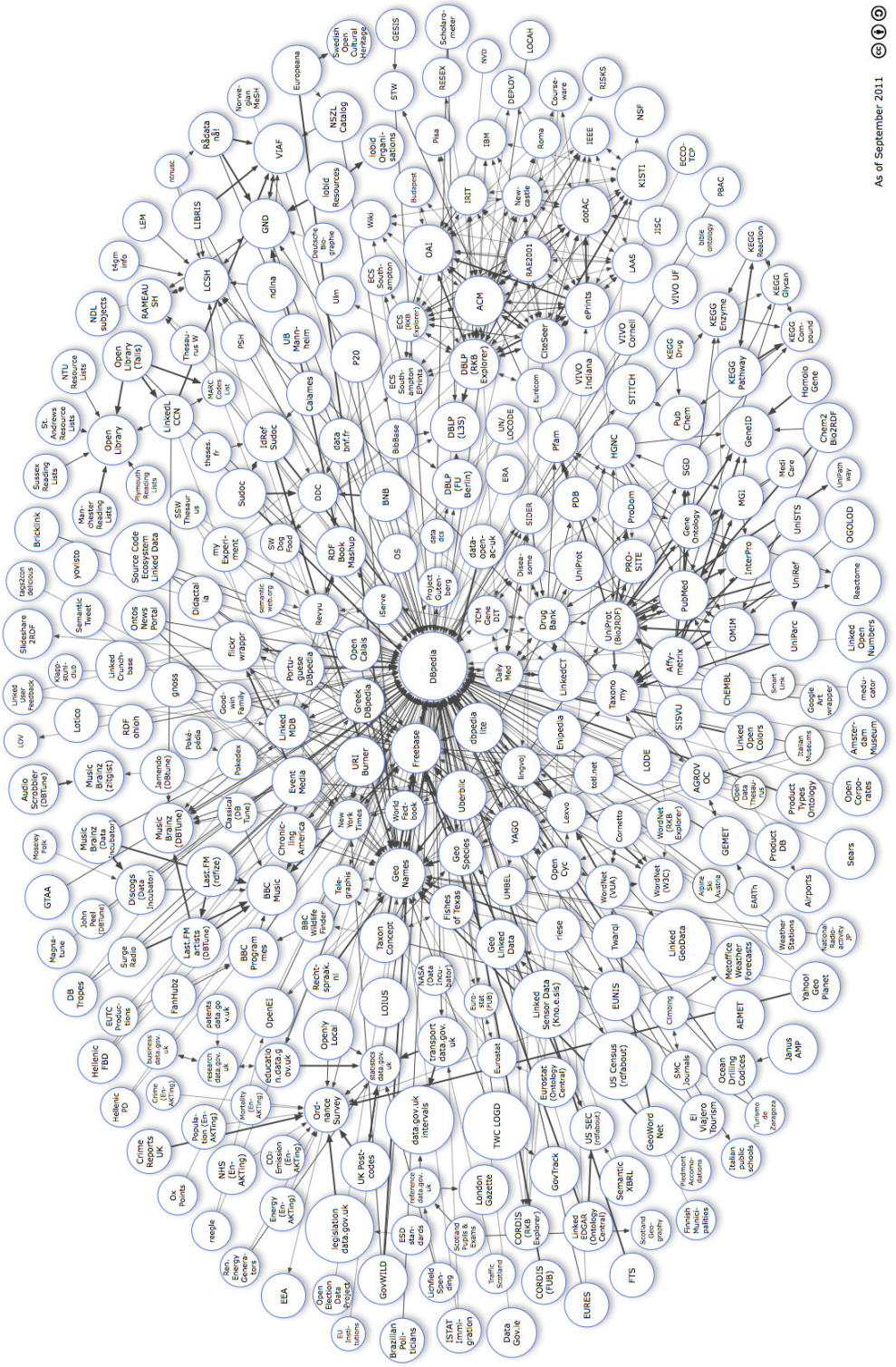


Figure 1.4: LOD datasets as of 2011

The read-write Web and Social Networking Sites (SNSs) have opened up more opportunities for people to share their data, compared with the Web which is read-only. The WWW has adapted since it began in around 1993 and online content publishing has began to move from the control held by a number of pioneer companies (which stimulated Web 1.0) to the decentralised interaction and collaboration between ordinary users (the heart of Web 2.0 (O'Reilly, 2007)). Before that, apart from Web content publishers, most users played the role of the consumer during the flow of Web content. The Web has changed the communication in our human society, and the research on social networks has also evolved as time goes by. Regarded as one of central groundbreaking products of Web 2.0, SNSs have emerged and changed our daily life bit by bit via adopting useful features from Usenet, Bulletin Board Service (BBS) and so on, and the analysis on them have benefited knowledge sharing in the (in)visible online community. A typical SNS normally has profiles of registered users, links between these users and the WSs provided according to profiles. Links are either bidirectional (e.g., on Facebook) or unidirectional (e.g., on Twitter). Compared to the online community services, SNSs are individual-centred rather than group-centred. As of 2011, 79% of American adults use the Internet and 59% of them use at least one SNS, which is close to double the percentage in 2008, as shown in a report (Hampton et al., 2011). Interestingly, this report also analysed the age distribution of SNS users and the result is illustrated in Figure 1.5 which shows the average population of SNSs are shifting to older users. Based on the above analysis, an increasing number of users join some SNS every day to share their statuses, events, audios, videos, and other interests and social activities. Undoubtedly, the SNS has become one of the most important medias for people to share knowledge on daily basis.

1.3 Bringing the (Semantic) Web into OpenKnowledge

The Internet has been one of the most effective networks for interconnecting digital devices. On top of it, the WWW has been built as a system for people or machines to publish interlinked hypertext documents. The client/server architecture makes the WWW a place where service requesters can interact with service providers via client-side WWW-aware applications such as Web browsers. Nowadays, users and network nodes on the WWW play increasingly diverse and dynamic roles within different kinds of interactions associated with different kinds of tasks. At the time of writing this the-

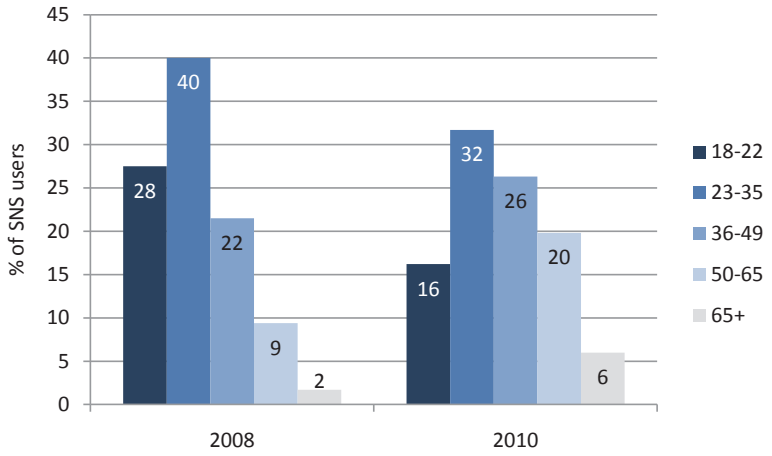


Figure 1.5: Age distribution of SNS users in 2008 and 2010

sis, RDF (McBride, 2004) is one of widely accepted knowledge representation models and focused on the knowledge sharing on the WWW. Along with RDF, a stack of methods and techniques have been proposed in order to transform the current Web into a so-called Semantic Web proposed to be the Web of data, an extension of the current Web of documents. Semantic Web-related researches and technologies have been blooming since 2001. As emphasised by Chris Welty in one of the keynote speeches, in the 21st international World Wide Web Conference in 2012, six components belonging to Watson (a super machine created by IBM which beat the best human players in the American quiz show *Jeopardy!* in 2011) have adopted semantic techniques.

On the other hand, service discovery plays an increasingly significant role as the information on the Web continues to explode. Based on the links between Web documents, search engine technologies have boosted in last two decades. With the development of the Semantic Web and Linked Data techniques, search engine vendors have realised that traditional searching is limited by the links between online documents. For example, in a situation where Bob is a retailer who buys a specific type of product from Carol and then sells it to Alice, it is difficult for Alice to make contact with Carol, relying on traditional search techniques. In another example, services (especially online services) may have availability issues such as how to find temporary off-line services or forthcoming new services and how to notify the service request as soon as they come online. This is still a challenge for existing service discovery techniques. The Web of data provides more fine-grained typed links, which can provide users with better searching experience by improving precisions/recalls (Joachims, 1998), compared with the document Web. For example, when a customer is browsing the Web page

of a restaurant from a Web site of reviews, by making use of his/her shared location and the restaurant location embedded as metadata, a browser plugin/addon can automatically plan the journey for this customer to get to the restaurant through his/her favourite means of transportation at a convenient time (since the search engines aware of the Web of data will target any data associated with particular types, semantically enhanced services have better chance to be discovered and used by them, compared with services with no annotations attached. Service discovery is an important functionality that the above Web-oriented infrastructure needs to provide and therefore, a meta-search engine has been built in this thesis to support users in discovering shared knowledge at the metadata-level. Services are encoded in a lightweight language derived from the process calculus and published as Web documents that are attached with tailored semantic annotations which can benefit the service discovery or even the interactions themselves during the runtime.

All in all, there are many prominent advantages of having the OpenKnowledge system fully integrated with the Web, including the network effect and demonstrably improved accessibility, etc. In this thesis, we have designed and built a Web-oriented infrastructure for nodes to share knowledge on top of the conventional WWW architecture. A system has been implemented as an online portal, which can be deployed by any network node owners in a decentralised environment, for peers to publish, discover, and (un)subscribe to IMs in order to collaborate with others and achieve common goals. On the other hand, a lightweight solution was also proposed and materialised for enabling each peer to run an IM interpreter inside a normal Web browser and interact with others in a peer-to-peer manner based on predefined specifications after subscriptions via the above online portal. More architectural details will be discussed in later chapters and initially illustrated in, for example, Figure 3.1 and Figure 4.1.

Many systems exist for community formation in extensions of traditional Web environments (e.g., SNSs, the Stack Exchange Network and BBSs, which normally run on centralised servers and for which users need to register and get an account to interact with others) but little work has been done in forming and maintaining community in the more dynamic environments emerging from *ad hoc* and peer-to-peer networks. On top of the above knowledge sharing infrastructure, this thesis also proposes an approach for forming and evolving peer communities based on the sharing of choreography specifications (in the form of IMs). Besides the meta-search engine, a dynamic peer grouping algorithm has been invented to asset peers in discovering service chore-

ographies and collaborators. An online platform has been implemented in accordance with this approach to help peers publish profiles and IMs (via a semantic enhancement strategy and an interactive annotator), discovering IMs and collaborative peers (via a meta-search-based method and a peer-group-based method plus ranking strategy in the case of multiple results returned) and (un)subscribing to IMs (via a user-friendly UI built on top of a well-modularised Content Management System (CMS)).

Chapter 2

Relevant Literature

This chapter discusses a selection of work related to this thesis, including the OpenKnowledge system, (Semantic) Web Service (WS) descriptions, peer-to-peer communities and agent-based peer-to-peer architectures, as well as the state of the art of social networks. Section 2.1 elaborates the underlying architecture of OpenKnowledge, which provides the theoretical basis of this thesis and has also inspired our work from the technical perspective. Section 2.2 revisits the languages designed for describing (Semantic) WS and compares them with the language employed by the OpenKnowledge system. *Ad hoc* and peer-to-peer networks are ideal environments for individuals or organisations to share knowledge in an open decentralised manner, and Section 2.3 and 2.4 present work on these topics. Section 2.5 describes several influential Social Networking Sites (SNSs) that are thriving at the time of writing this thesis.

2.1 Underlying Architecture of OpenKnowledge

With the Internet playing an increasingly important role in people's daily life, Web-based[1] knowledge sharing has become a challenge to almost every individual or organisation. Orchestration-based systems, where typically only the central network node has access to the workflow, do not scale well in an open decentralised or distributed environment (Barker et al., 2009; Besana et al., 2009). As overviewed in Chapter 1, the OpenKnowledge system offers an open platform for sharing choreographies among

[1] Our work has relied on the World Wide Web (WWW) in the application layer of the Internet. The knowledge sharing in diverse formats of data exchanging originated in other Internet layers is out of the scope of this thesis.

peers and based on those choreography descriptions, services can be discovered and executed in a peer-to-peer manner. Following that overview, this section further discusses the underlying architecture of OpenKnowledge.

The OpenKnowledge project explored an approach which allows peers to share knowledge via semantics of interactions instead of a pre-agreed universal standardised semantics. The protocols of interactions are encoded in so-called Interaction Models (IMs), each of which is a Lightweight Coordination Calculus (LCC) (Robertson, 2004) specification on the roles of peers and the communicative actions allowed within a particular interaction. Each instance of the OpenKnowledge system needs to run a so-called *OpenKnowledge Kernel*, a software providing a discovery service for peers to find services that may meet their requirements, curating subscription information (e.,g., peer-role pairs, etc.) and also executing choreographies encapsulated in IMs. Each peer provides services (a standardised way of describing functionalities) and attempts to satisfy the constraints referenced by subscribed IMs and these services are named as OpenKnowledge Components (OKCs). OKCs can be stored in the local OKC repository or published on distributed Discovery and Team Formation Service stores (de Pinninck Bas et al., 2007). Meanwhile, the OpenKnowledge system also enables peer to publish and validate new IMs. Moreover, a peer ranking module is under development in the OpenKnowledge kernel with the aim of recommending IMs, OKCs and collaborative peers. This module will be wrapped into the Trust and Reputation Service which is in charge of tracking peers and gathering information about them. Besides the above two types of services, another service called the Mapping Service is used for peers to reconcile their heterogeneity when sharing knowledge (de Pinninck Bas et al., 2007).

The architecture of the OpenKnowledge system is illustrated in Figure 2.1. As shown in this figure, peers are responsible for storing IMs and OKCs. An IM defines roles played by peers and can be interpreted by the LCC interpreter deployed on a so-called coordinator which itself is also a peer and coordinates interactions between peers by retrieving and executing OKCs managed by each of them in order to help peers to check if constraints predefined in the IM are satisfied or not.

The OpenKnowledge system can provide the functionalities described above as long as the kernel is downloaded and installed appropriately. A user can create his/her own OKCs by wrapping a piece of code via the provided Graphical User Interface (GUI) and can then publish these OKCs to the distributed discovery service which is

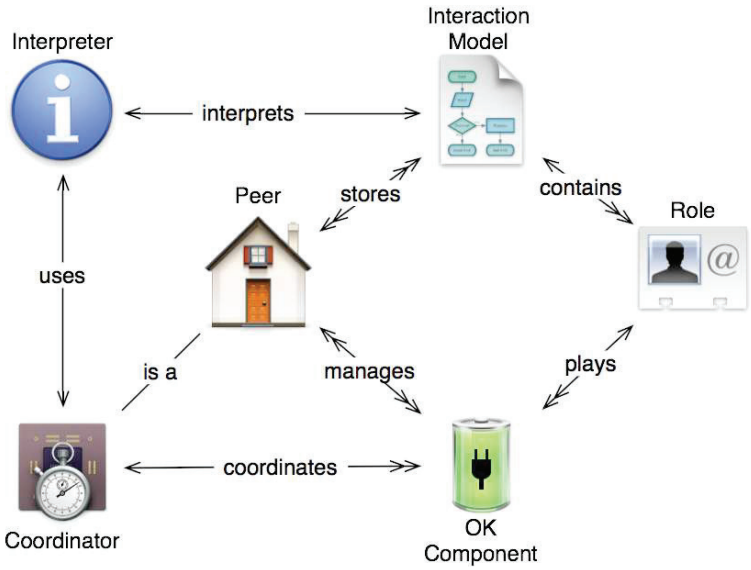


Figure 2.1: Architecture of the OpenKnowledge system

built on top of a directory storage. In order to find a useful IM, a user need to type into the GUI keywords, with which IMs (pre-annotated with keywords) stored in the storage will be queried over. Then, the user decides which of the discovered IMs to subscribe to and the kernel will randomly select a peer as a coordinator thereafter. Finally, subscription information is automatically forwarded to the coordinator and triggers the running of the IM. Our approach proposed in this thesis does not require any coordinator to play the middle-man role and delegate peers to run IMs. All the IMs will be run on the involved peers only and messages will be passed directly from senders to receivers in a peer-to-peer way, which effectively protects them from man-in-the-middle attacks (Kelsey et al., 1998). On the other hand, sometimes peers are lack of incentives for taking the coordinator responsibility originally required by the OpenKnowledge system.

2.2 Semantic Web Services and Their Descriptions

OpenKnowledge provides a platform for peers to share WS choreography descriptions. How to describe WSs and make them easy to share is a hot topic in both WS and Semantic Web (SW) communities. From the perspective of orchestration and also inspired by Web Services Description Language (WSDL) (Christensen et al., 2001), several approaches have been proposed for semantically enhancing WS descriptions

(in order to improve WS-oriented tasks including discovery and composition, etc.), such as OWL-S (Martin et al., 2004), WSDL-S (Akkiraju, 2005) and Semantic Annotations for WSDL and XML Schema (SAWSDL) (Farrell and Lausen, 2007). Accompanying these, several matchmakers (Paolucci et al., 2002) have been built, such as the OWL-S Matchmaker (OWLS-MX) (Klusch et al., 2006) and SAWSDL Matchmaker (SAWSDL-MX) (Klusch and Kapahnke, 2008). However, insufficient attention has been paid to semantically enhancing descriptions of WS choreography. Web Services Choreography Description Language (WS-CDL) (Kavantzas et al., 2005) has been proposed for describing WS choreography but it lacks appropriate support in a Uniform Resource Identifier (URI)-dedicated vocabulary which can be used for semantic annotations. On the other hand, existing Semantic Web Service (SWS) choreography description languages such as Web Service Modeling Ontology (WSMO) (Lara et al., 2004) are expressive and powerful but too heavyweight to run locally on autonomous peers such as mobile phones and personal digital assistants. For the purpose of improving WSMO, WSMO-Lite (Vitvar et al., 2008) defines a lightweight set of semantic service descriptions based on SAWSDL and has extended SAWSDL with conditions and effects. It enables the integration of REpresentational State Transfer (REST)-driven WSs and has been used for annotating WS descriptions.

WS annotating tools have been also developed and are continually evolving. ASSAM (He et al., 2004) was developed for helping publishers to semi-automatically annotate WS descriptions with existing ontology terms recommended by machine learning algorithms. METEOR-S Web Service Annotation Framework (MWSAF) (Patil et al., 2004) was designed (similarly with ASSAM) for semi-automatically annotating WS descriptions based on ontologies. Several REST-focused vocabularies were also devised due to the prosperity of RESTful Application Programming Interfaces (APIs) and services. HTML for RESTful Service (hRESTS) (Kopecky et al., 2009) is a Microformat (Suda, 2006) designed for describing Web APIs (wrapped in a simple service model) in a machine-readable manner. Its two extensions, Semantic Annotations for REST (SA-REST) [2] and MicroWSMO [3], were also proposed and have been focused on faceting public APIs and supporting semantic automation, respectively. Also targeting RESTful WSs, RESTdesc (Verborgh et al., 2011) was proposed as an approach to describing services in Notation3 (Berners-Lee, 1998) and service discovery will be achieved in terms of these descriptions by sending requests to service providers via

[2] http://www.w3.org/Submission/SA-REST/
[3] http://www.wsmo.org/TR/d38/v0.1/

the Hypertext Transfer Protocol (HTTP) OPTION method, which requires users to know the locations of the services they need up front. SmartLink (Dietze et al., 2011) provides a Web-based editor and search environment for Linked Services (conforms to the Linked Data principles). It encourages service publishers to employ a simplified schema with non-functional properties based on the Minimal Service Model (MSM) (Pedrinaci and Domingue, 2010) as well as use the aforementioned WSMO-Lite to annotate WSs.

Web-page-embedded metadata provides content publishers with a chance to attach semantics to Web content for the purpose of making their content both human-readable and machine-readable so client-side user agents can automatically glean the embedded data and improve the user experience in one way or another. Several solutions for embedding metadata into Web pages have been proposed. Microformat (Suda, 2006) makes use of existing HyperText Markup Language (HTML) and Extensible HyperText Markup Language (XHTML) tags to convey metadata and other attributes but new formats require new data models. Davis from Talis proposed Embedded RDF (eRDF)[4], which can be used with any version of HTML but it restricts itself to the existing HTML attributes and does not support full Resource Description Framework (RDF) (there is no data type and no blank node). Resource Description Framework in Attributes (RDFa) was proposed by Adida et al. (2008), which not only takes advantage of existing HTML attributes but also invents several new XHTML attributes for achieving flexibility and disambiguation within the process of Web pages are being marked-up. It reuses the existing RDF model and supports full RDF semantics. Microdata (Hickson, 2012) has been proposed and is still evolving as part of W3C editor's draft on HTML5. It has learnt lessons from both Microformats and RDFa and created new HTML5-only attributes to hook annotations but it is not backward compatible to HTML4. As of writing this thesis, Microdata does not support multi-vocabularies or co-typing. Schema.org [5] has been designed as a centralised vocabulary hosting site and the Microdata, which makes use of HTML5 attributes to carry annotations, is the only metadata-embedding syntax supported and recommended by the hosted vocabulary.

In this thesis, our approach to semantically enhancing WS choreography descriptions also employs an embedded-metadata-based strategy within the process of IM publication and publisers are encouraged to use URIs maintained by DBpedia (Auer et al.,

[4] http://en.wikipedia.org/wiki/Embedded_RDF
[5] http://schema.org/

2007) to annotate IMs. This strategy also works for any other that happen to be vocabularies in RDF due to its support in the full RDF data model. When peers face IMs annotated with diverse URIs indicating the identical individuals, they can use URIs comparators such as the sameAs service [6] or any appropriate ontology matchmakers to fulfil the alignment task. The metadata-embedding strategy is employed in the IM publishing process and RDFa was chosen as the serialisation format in this thesis due to its support on full RDF semantics. Note that, although users can create their own vocabularies to annotate and publish IMs, existing pervasive vocabularies are recommended due to their wide acceptance in the SW community. Nowadays, more and more search engines have began to crawl and index metadata-embedded Web pages and they actually provide us with a natural platform for building discovery services for the OpenKnowledge system.

2.3 Peer-to-Peer Communities

This thesis proposes an approach to forming the community of peers by automatically grouping them in terms of their common interests and aiming at this formation task, several related work have been done. Yao and Julita (2004) demonstrated a way of applying their peer-to-peer community formation mechanism to academic paper sharing based on the trust of each peer. Their mechanism and trust computation are based on a limited paper-sharing scenario and peers' actions are however more various and more complex in the real world. Kaykova et al. (2004) introduced an ontology-based community formation process depending heavily on domain ontologies and their hierarchical structures. Actually, peers described with different domain ontologies may have common interests and further form groups even though they do not have explicit structural connections in between. Khambatti et al. (2004) gave an approach to structuring a peer-to-peer network and a community-based search protocol was also presented to provide better search operations. Existing developments and research on social networks motivated their work but only query-based peer search was considered and possible peer interactions and service compositions were not taken into account within the peer community formation process. Palma et al. (2005) presented Oyster, which is a project for sharing and reusing ontologies in a peer-to-peer community. It gives a standard form of metadata for describing ontologies and their work is more con-

[6] http://sameas.org/

cerned with ontology sharing and less concerned with how peers can engage in community formation. Liu et al. (2006) proposed a statistics-based approach to building communities with hierarchical structure and quantifying the peers' similarities without disclosing their personal profiles. In our approach, peers do not achieve privacy by hiding their profiles and they can publish any content which they want others to know but still have a chance to hide sensitive information. Only the peer itself has the authority over its profile. Davoust and Esfandiari (2009) demonstrated a tool for publishing and navigating linked data over a peer-to-peer file-sharing network and this tool was implemented based on peer communities triggered by a so-called root community (Davoust and Esfandiari, 2008). However, it requires peers to use a common schema and does not support join queries. In this thesis, peer communities are built with the open environment based on our approach and they assist peers in discovering services and collaborative peers in distributed. We also maximised the extensibility of the community notion by employing vocabularies which have been widely used in the Semantic Web and Linked Data community.

The publish/subscribe model was originally investigated in (Birman and Joseph, 1987). A peer-to-peer publish/subscribe system supporting metadata and query language based on RDF was implemented in (Chirita et al., 2004). In this thesis, the discovery peer has implemented the publish/subscribe model in a different way: firstly, super peers are discovery peers rather than servers on the publisher side, which are more expensive to maintain for individual users and usually untrustworthy; secondly, subscription information is curated on the third party hub so neither discovery peers nor publisher peers have to store the data themselves.

Peer communities have been categorised based on different criteria from different perspectives (Porter, 2004; Akram and Allan, 2005). These criteria also apply to the peer community formed in this thesis and can assist discovery peers (introduced in Chapter 6) in discovering interesting community members.

2.4 Agent-based Peer-to-Peer Architecture

A number of peer-to-peer systems have been designed and implemented to address the issue of decentralised information sharing. However, among these systems, most are dedicated to sharing files, computing resources or bandwidth between peers. The agent

paradigm and the peer-to-peer paradigm are complementary to each other in spite of several overlapping concepts (e.g. autonomy and social abilities, etc.) and agents can be regarded as the "brain" of a peer, handling the logics of interaction with other peers. An agent model may provide peer-to-peer systems with high level abstract layers in which the features of real world situations can be better reflected (Moro et al., 2003). Little work on community-based peer collaborations (i.e., service sharing) has been done due to the lack of fundamental agreement on the interaction protocol(s). An agent-based peer-to-peer architecture is proposed in this thesis and on top of that, a peer community can be formed to help its members in sharing resources including not only files and computing power but also services which pave the way to social computing (Wang et al., 2007). In decentralised environments such as *ad hoc* and peer-to-peer networks, the collaborations require coordination to guarantee service quality since peers are autonomous and egocentric. The coordination itself has to be decentralised, which is difficult to achieve. Thanks to LCC, we now have a lightweight language for describing the peer coordination. In this thesis, the coordination in LCC will be published in the Web documents with embedded metadata. Therefore, the peer coordination in the community can be finally decentralised in the form of the Web document which will be shared by peer members together with their resources.

The above describes the multi-agent aspect of our solution and here, we also compare this solution with the Foundation for Intelligent Physical Agents (FIPA)-compliant multi-agent systems. FIPA is "an IEEE Computer Society standards organisation that promotes agent-based technology and the interoperability of its standards with other technologies".[7] A FIPA-compliant multi-agent system is an agent platform implemented based on the subset (25 specifications were selected in 2002) of FIPA specifications which represent a collection of standards widely accepted in the FIPA community. This type of system usually has following three components:

a. Agent Management System (AMS): It curates a white page of agents deployed on a specific platform and is used for agent registration and authentication.

b. Directory Facilitator (DF): It curates a yellow page of agents services and is used for discovering agents and their offered services.

c. Message Transport Service (MTS): It is in charge of supporting the intra-platform or inter-platform message passing between agents.

[7] `http://www.fipa.org/`

Moreover, FIPA-compliant agent systems use FIPA Agent Communication Language (ACL) as the format of messages. FIPA ACL is derived from Knowledge Query and Manipulation Language (KQML) (Finin et al., 1994) and has a more extensively explored formal semantics (Bellifemine et al., 1999). Compared with this type of agent systems, in our solution, Web browsers become the Agent Platform (AP) and there is no need to set up a separate AMS for agent management since agent registration and authentication will be done on Extensible Messaging and Presence Protocol (XMPP) servers. There is no need to set up DF either in our solution since agent and service discovery will be achieved via online peer communities which locally trace any participation and interaction activities. MTS is used for channeling messages in FIPA-compliant agent systems while in the Web browser, messages are channeled based on an interaction ID (a global session/conversation ID). FIPA ACL could be incorporated by our solution to substitute the currently employed JavaScript Object Notation (JSON) format to enable our system to interact with other FIPA-compliant agent systems so our system can behave like a fully functional multi-agent system to serve the similar communication/collaboration purpose. Nevertheless, it is difficult if not impossible to make our solution adapt and transform it into a FIPA-compliant agent platform due to the fundamental differentiation of their architecture designs as well as other requirements for FIPA agent applications and systems. For example, it is hard to do the browser-to-browser message passing in terms of the traditional client/server Web architecture and browsers are restricted to client-side applications which does not accept incoming connections. For security reason, Web documents from different domains are prevented from communicating with one another so it is not possible for browsers to share existing DFs. A variety of multi-agent modelling architectures have been proposed but little progress has been made on running intelligent agents in Web browsers prevalent on various operating systems because of the above constraints. Another challenge for this is preserving the states of agents during a complex interaction, which cannot be easily obtained without involving server-side codes and databases. Comparatively, our solution in this thesis provides a novel and browser-friendly design by which agents can interact with each other in a peer-to-peer manner via modern browsers (see in Section 7.3).

2.5 Social Networks: State of the Art

SNSs (e.g., MySpace, Facebook, Twitter and Google+ etc.) have flourished in the last decade and have made considerable social impact on people's daily life. An increasing number of users join this or that SNS with their real identity and some of them even share highly personal information with their online buddies. Also based on the report referred in Chapter 1, on average, only 7% of Facebook friends are people a registered user has never met in person and only 3% of them a registered user has met only one time (Hampton et al., 2011). According to this report, we can see that the SNSs tend to reflect social relationships of people in the real world and at the same time make maintaining existing relationships or building new ones extremely easy thanks to the power of Internet communication so it is not surprising that the average age of adult-SNSs users has shifted from 33 in 2008 to 38 in 2010. With the development of mobile devices, the social networks have began to migrate to small devices and become mobilised. Many applications invoke the APIs of Short Message Services (SMSs) and enrich existing social information with contextual data such as locations and temperatures. However, almost every SNS curates the data of registered users on its own server and there is an awkward fact that these users cannot 100% manipulate the data about themselves due to SNS companies' own interests. This architecture design for SNSs currently not only threatens people's privacy but also hampers the formation of decentralised or distributed social networks. Although several SNS companies provide the service of helping individuals or organisations to build self-organised sites instead of using the centralised services, many restrictive policies that may make profits for the SNS companies themselves are still out there so users, especially those who have conflict interests with those SNS companies, still hesitate to adopt that kind of services. By giving up extending the existing SNS architecture, several startups such as Diaspora[8] and OneSocialWeb[9] have been developing their own prototypes, and expect ordinary users to run their own personal social web servers so a decentralised social environment can be achieved to some extent when the servers representing/delegating different users begin to communicate with each other.

With the increasing amount of attention paid to the decentralised or distributed social networks, there has been a fierce war between a variety of communication protocols. It is not surprising that the protocols for email, the most widely adopted decentralised

[8] https://joindiaspora.com/
[9] http://onesocialweb.org/

tool, have been applied in the communication layer of several SNSs such as Mr. Privacy (based on Simple Mail Transfer Protocol (SMTP) and Internet Message Access Protocol (IMAP)) (Fischer et al., 2011) and psyced (based on SMTP).[10] These email protocols are simple and efficient and are already rooted in most people's daily life. However, since they are not designed as real-time solutions, it is difficult if not impossible to use them for fulfilling a real-time job such as status updating and instant message passing, etc. Therefore, lots of effort has been paid on creating diverse *ad hoc* protocols for tackling each aspect of the social network. Some of them are dedicated to authentication and authorisation, including OAuth (Hammer-Lahav, 2010), OAuth 2.0 (Recordon and Hardt, 2012), Activity Streams [11], OpenID (Recordon and Reed, 2006) and WebID (Sporny et al., 2011); some of them are focused on users' profiles, including Portable Contacts (Smarr, 2008) and Webfinger;[12] others are dedicated to updating user status in the real-time manner, including OpenMicroBlogging employed by OpenMicroBlogger[13] and later Ostatus (Prodromou et al., 2010). Last but not least, PubSubHubbub (Fitzpatrick et al., 2012) was designed as an open protocol for distributed communication based on the publish/subscribe model and has a demonstrative hub server on the Google App Engine.[14]

With respect to messaging protocols, XMPP (Saint-Andre, 2004) allows users to pass messages to others from different domains (servers). Several other protocols coexist and are comparable to XMPP. Simple (or Streaming) Text Orientated Messaging Protocol (STOMP)[15] is coming from the HTTP school of design and similar to HTTP. It is a lightweight message queuing protocol and has a limited number of commands to work over Transmission Control Protocol (TCP). Due to its simplicity, STOMP has been implemented in many contemporary programming languages but meanwhile, its functionalities are relatively limited compared with XMPP. Advanced Message Queuing Protocol (AMQP) is a middleware and wire-level protocol for building interoperable message-oriented systems (Kramer, 2009). This protocol has more functionalities/features than STOMP and has been implemented in several main-stream languages. Evaluation notes of AMQP and XMPP and other messaging protocols have done by the Linden Labs on the Second Life Wiki.[16] Compared with AMQP, XMPP

[10] http://www.psyced.org/
[11] http://activitystrea.ms/
[12] http://webfinger.org/
[13] http://openmicroblogger.org/
[14] https://appengine.google.com/
[15] stomp.github.com
[16] http://wiki.secondlife.com/wiki/Message_Queue_Evaluation_Notes

has several strengths and weaknesses including: supporting presence, quick-and-easy-to-fire, hard to manage roster and overhead of presence updates and so forth. Both AMQP and XMPP require intermediation (brokers in AMQP and servers in XMPP) and total peer-to-peer communication is difficult if not impossible to achieve accounting for the complexity of the current network architecture. At the time of writing this thesis, AMQP has not got broker federations standardised even though some of its implementations such as RabbitMQ (Videla and Williams, 2012) support this type of federation via *ad hoc* plug-ins. On that basis, XMPP is comparably easier and more straightforward to apply in distributed environments like peer-to-peer networks which contain decentralised brokers. That is also a reason why XMPP was chosen in this thesis to power the peer-to-peer messaging infrastructure. Note that even though the above protocols have claimed their goals to solve diverse aspects of social communication, they still share one or more features with others as described in their specifications.

As mentioned earlier, Diaspora is a free social network software by which each user can set up a SNS on his/her own server and all Diaspora servers are able to communicate with each other so a distributed social network can be achieved. It has employed the Salmon protocol [17] and also implemented the OStatus protocol. OneSocialWeb is a XMPP-based decentralised client-side software which can be installed and turn users' servers into an open social network platform. It was initiated by the Vodafone Group Research and Development, and aims to offer an open platform to which all other SNSs can be connected. OpenLink Data Spaces (ODSs) (Idehen and Erling, 2008) is a data space (distributed named structured data cluster) platform on which a wide spectrum of distributed collaborative applications are curated, including blogs, wikis, bookmarks and files, etc and each item of data on the ODS has a dereferenceable URI. The design of ODS is compliant with the Linked Data principles (Berners-Lee, 2006) so this platform can be regarded as a distributed social network built on top of the Web of data. Likewise, StatusNet [18] and Semantic MicroBlogging (SMOB) (Passant et al., 2010) offer a distributed microblogging system and its Linked Data version respectively. Last but not least, aiming at providing applications interoperating in the context of different SNSs, the OpenSocial Foundation was founded as a non-profit corporation dedicated to delivering open and free OpenSocial specifications, and several protocols and related APIs were born thanks to this community (Häsel, 2011).

[17] http://www.salmon-protocol.org/
[18] http://status.net/

To recap, the current SNSs or mobile social apps normally deal with community formation in the extensions of traditional Web environments but little is known about how communities might be formed and maintained in the more dynamic environments emerging from *ad hoc* and peer-to-peer networks. Compared with other social network solutions, our approach and implementations can dynamically form peer groups which assist peers in discovering more interaction protocols of interest as well as collaborative peers. In the distributed environment, there is no centralised node and each participant is more autonomous. Compared with the traditional SNS users, based on our approach, peers only have to search and subscribe to desired IMs and then leave the remaining work to our system.

Summary

In this chapter, we revisited existing methods and technologies behind the OpenKnowledge system, (Semantic) WSs, peer-to-peer communities, the agent-based peer-to-peer architecture and SNSs. With social networks thriving, many tools have been developed for forming online communities in extensions of traditional Web environments but little work has been done in forming and maintaining communities in the more dynamic environments emerging from *ad hoc* and peer-to-peer networks. OpenKnowledge actually has the potential to embrace the methodology of WWW and form an open knowledge sharing community which is easy for everyone to access. This potential will be explored and showcased through the remainder of this thesis.

Chapter 3

Decentralised Interaction-Driven Knowledge Sharing on OKBook

With the flourishing of Web 2.0, service providers may care more about how online communities (e.g., eBay, Yelp, etc.) review their products than the rankings provided by traditional search engine giants. On the other hand, service requesters trust recommendations by other users who have (in)direct connection with each other in social communities more than usually advertisements from service providers biased by their own interests. Therefore, the online community formed normally by grassroots users plays an increasingly important role in addressing issues related to service discovery. How to form online communities for users to share knowledge as story telling is still a hot research topic. Several systems exist for community formation in extensions of traditional Web environments but little is known about how communities might be formed and maintained in the more dynamic environments emerging from *ad hoc* and peer-to-peer networks. A difficulty with establishing communities in traditional peer-to-peer systems is that there is no structure that can be used as a basis for community formation; there is nothing analogous to their relations used behind the scenes in Web-based social networking systems in order to infer community information (such as friend-of-a-friend relationships). Some recent peer-to-peer knowledge sharing systems have, however, used languages for specifying choreography between peers that can be used to provide the relations to build social networks. The OpenKnowledge project has developed a peer-to-peer knowledge sharing system in which peer interactions are described as Interaction Models (IMs) coded in Lightweight Coordination Calculus (LCC). On that basis, in Section 3.3, an approach is proposed for forming

and evolving peer-to-peer communities in terms of their shared protocols with the assistance from the IM publishing system also designed and implemented in this thesis. Based on this approach we have developed OKBook — an open online platform for realising this new form of peer-to-peer communities.

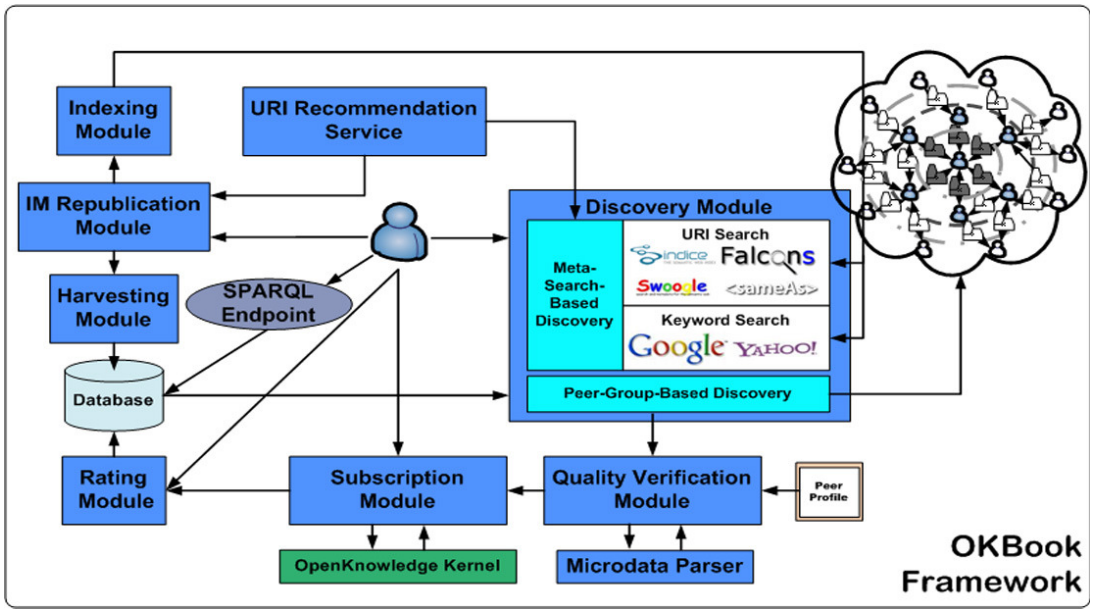


Figure 3.1: Overlook of OKBook modules

Figure 3.1 shows the main components which constitute the OKBook system. The *IM Republication Module* assists IM publishers in publishing IMs via Web pages using annotation embedding strategies such as Resource Description Framework in Attributes (RDFa) and URI lookup services (the detail on annotating IMs will be further discussed in Section 5.2). Then the *Indexing Module* gets these IM pages indexed by search engine giants such as Google and Semantic Web Search Engines such as Sindice (Tummarello et al., 2007) using tailored page submission calls. The *Discovery Module* offers a meta-search engine built on top of several Semantic Web Search Engines (SWSEs) and ranks search results by synthesising rankings given by these search engines. Indices from Google and Yahoo!Search are used for assisting users in refining and adjusting their keyword-based queries when few Uniform Resource Identifiers (URIs) related to these queries are returned. Moreover, the *Discovery Module* also allows users to discover IMs that meet their requirements in terms of peer groups which will be formed dynamically based on peers' interaction logs. Group members usually have particular interests in common and the above discovery mechanism is able to significantly cut down the IM search space compared with the meta-search-based mechanism (the performance on this mechanism will be shown in Section 7.1). The

URI Recommendation Service provides appropriate URIs of things for both the IM publisher within the annotation process and the IM requester within the querying process, respectively. The *Quality Verification Module* is in charge of discovering which role a peer is qualified to play in terms of IM triples harvested by adopting the embedded metadata parser as well as the triples from the profile shared by this peer up front. The *Harvesting Module* gleans triples serialised in RDFa from the published IM Web pages and later dumps them into a back-end triple store. The *Group Discovery Module* analyses peers' interaction logs and discovers peer groups based on the criteria which will be detailed in Section 3.1. If a peer is qualified to play a specific role in an IM, it can subscribe to this IM through the *Subscription Module* which will display all potential roles this peer is capable of playing and then the subscription information will be forwarded to each participants of the interaction. After the execution of the IM, the *Subscription Module* forwards the final results to the *Rating Module* through which peers can give some feedback about the IM to its original publisher based on their degrees of satisfaction with the interaction. The ratings further inform IM selection beyond the ranking provided by the *Discovery Module*. Meanwhile, the execution result will be stored as one interaction entry of every involved peer into the database of interaction logs. A SPARQL Protocol and RDF Query Language (SPARQL) endpoint is also exposed to peers so they can query over and reuse triples (e.g., creating mashups, etc.) stored in the back-end triple store so as to create and publish more linked data and (Semantic) Web Services (WSs).

Following Section 3.1, Section 3.2 describes the strategies used on OKBook for ranking IMs, peer communities and peer group members. Section 3.4 compares OKBook against with the OpenKnowledge system and analyses our system by giving its advantages. Section 3.5 takes peer-to-peer message passing one step further and proposes a solution to resource sharing via federating multiple OKBook servers.

3.1 How OKBook Architecture Meets the Requirements

So far, we have talked about how our goal is to realise a decentralised knowledge sharing environment, based on specifications of interactive activities, and the inside modularised functionalities. This section discusses how the above functionalities are achieved. On OKBook, peer profiles play an important role within IM discovery, and we explain how they are represented (Section 3.1.1), managed (Section 3.1.2) and used

for peer and IM discovery (Section 3.1.3).

3.1.1 Knowledge Representation for OKBook

OKBook is an online platform inspired by OpenKnowledge and has been built on top of a Content Management System (CMS). Through this platform, peers can import, publish, discover and subscribe/unsubscribe to IMs or join social groups with the assistance of which peers can find out desired interactive specification as well as related collaborators. The main modules of this platform are described in this subsection.

3.1.1.1 Peer Capability Description and Its Storage

In our peer-to-peer community with the spirit originating in the OpenKnowledge system, each peer has a profile (described in Resource Description Framework (RDF)), which will be uploaded and fed into the discovery service as soon as the owner of this profile signs up to OKBook. A peer profile accommodates its owner's public information such as which roles it can play and also proprietary information such as how it can solve particular constraints and the corresponding methods wrapped in OpenKnowledge Components (OKCs).

Since peers can be linked to others based on interaction logs, a profile can be also used for describing the relationships between its owner and his/her "friends", and these relationships can be represented using a specific vocabulary (e.g., Friend of a Friend (FOAF) (Brickley and Miller, 2007) or XHTML Friends Network (XFN) (Çelik and Meyer, 2004), etc.). With respect to the content inside each peer's profile, these relationships may vary over time. For instance, after rounds of running of IMs, a peer may want to make friends with its collaborative peers during those interactions or in the case that if a peer is willing to provide more OKCs, it is predictably capable of playing more new roles no matter in existing IMs or newly created ones. The events given above will trigger the update of the peer's profile. Profiles may be stored in RDF repositories or plain databases at peers' own will. However, since graph-based RDF repositories are competitively easier to merge because of the schema-less structure (triples) of data sources, compared with plain databases, in order to make those repositories comply with our semantic enhancement of existing IMs (Bai and Robertson, 2010), we store the profiles in the RDF data model instead of a plain table in the

relational database.

With the *Discovery Module* as described in Figure 3.1, OKBook itself can be regarded as a discovery server. In the peer-to-peer network, a discovery server behaves like a super peer and more than one peer of this kind may co-exist. When a peer registers on one of these super peers, others of them will also get copies of this peer's profile and the peer can log on to any of discovery servers using the same username. More details of this server federation will be discussed in Section 3.5. Currently, OKBook supports the OpenID (Recordon and Reed, 2006) standard which allows users to log on to different services having support on OpenID with the same digital identity.

Centralised storage of triples is inefficient and in-compliant with the peer-to-peer network for two reasons: firstly, it normally takes several hours to load in a billion RDF triples [1] so centralised storage will be slow and impractical; secondly, some data are proprietary and sensitive, which should be stored separately when copyright and security issues are taken into considerations. OKBook stores peers' profiles in a distributed manner. Peer profiles are more concerned with providing information that can benefit interactions between peers and comply with choreographies in the whole OpenKnowledge ecosystem. As aforementioned, each OKBook peer has copies of registered peers' profiles and each of them can be referred via a unique Named Graph (Carroll et al., 2005) when being queried either by OKBook itself or by other peers via a SPARQL endpoint. At present, there are two ways for OKBook to acquire registered peers' profiles and both of them are described as follows (expressed via the First Order Predicate Logic):

a. *Loading As Change*. In this case, the whole profile of each registered peer will be uploaded to the discovery service when this peer successfully registers on OKBook (at this stage, OKBook is unaware of which triples will be actually used for the queries in future). Therefore, every time a peer's local profile is revised, copies of this profile on other OKBook peers will be synchronised with this local changed one. In the case that an interaction involving this peer is performed, its local profile and corresponding copies may be updated and appended with new information about the peer (e.g., new friends). Thus, the update of this peer's local profile will also influence its copies on other OKBook peers it has logged on to (details of the updating mechanism will be discussed in Section 3.1.2). Since each peer is a server and client on the Web, this updating task can be ful-

[1] `http://esw.w3.org/topic/LargeTripleStores`

filled via Hypertext Transfer Protocol (HTTP) requests with the POST method. Among those OKBook peers on which a peer has registrations, there needs to be at least one in charge of bootstrapping the interconnection in the whole peer-to-peer network, and this super peer is reliable, trustworthy and deployed with the OKBook platform and named as the Bootstrapping peer (Bpeer). Any other peers which want to install the OKBook platform must know at least one Bpeer and inform it of the new installation. Bpeers can then rank themselves based on popularity criteria such as the capacity of the bandwidth. Thus a peer can select reliable OKBook servers to register on by referring to their Bpeer rankings. Peers are required to log into at least one OKBook server with their usernames and passwords before any knowledge sharing interactions. The burden on the management of public and private keys, employed in other authentication algorithms in the peer-to-peer network (Chirita et al., 2004), can be mitigated via this form of peer registration, the detail of which is however out of the scope of this thesis.

b. *Lazy Loading*. With the growth of the number of registered peers, much larger profile copies would need to be uploaded to the OKBook peers. Taking advantage of the profile as its "business card", a peer may enrich it with extra information such as the homepage and workplaces, which have not been detected and harnessed by any OKBook peers but could actually draw lots of attention from others for the profile owner. Storing and updating peer profiles may however cause a big burden on each OKBook platform. Therefore, we make use of lazy loading RDF in order to mitigate this burden. Note that peers publish their profiles through annotation-embedded Web pages (e.g., RDFa employed by OKBook). In this lazy loading style, there is no profile copy of each registered peer and OKBook will harvest triples embedded in peers' profiles by making use of RDFa-parsing services provided by third party softwares or an internal RDFa parser carried on by the OKBook peer itself. This lazy way of acquiring profiles allows peers to revise their profiles without notifying OKBook since triples inside profiles will be harvested in real time. Using the CONSTRUCT query, OKBook can create a simplified version for the registered peer's profile and this version only contains the information which is useful to OKBook. After an interaction finishes, OKBook, which triggered this interaction, will inform participants of their potential friend peers. Peers can enrich the descriptions on their

profile updates using vocabularies such as Triplify update (Auer et al., 2009) or Talis Changesets[2] afterwards.

The above two ways of acquiring peers' profiles both have pros and cons, respectively. In the former way, by caching or indexing parsed triples, OKBook does not have to retrieve a peer profile every time it tries to fulfil a task such as discovering IMs. However, once the profile owner changes its content, OKBook has to update the local cache or index of this profile. In the latter way, each task that needs to acquire the information inside a peer profile will inevitably cause some traffic on updating. However, OKBook does not have to be notified of a peer changing its profile since the return of real time queries can provide the latest information about this peer itself. On the other hand, in the former way of loading, only the latest revision will influence the discovery while previous revisions on the peer's profile are all blanketed by the latest one. In other words, according to the latter way, the acquisition of a peer profile is necessary only at the moment when a query starts being solved. Thus the former way will work better especially when queries occur more than profile updates do and the latter way will work better especially when profile updates occur more often than queries do. OKBook chose the former way to go since, with the latter way being chosen, peer profiles would be updated within the querying process and this could harm the user experience when his/her profile contains a considerable number of triples need to be updated, which would slow down the response to those queries.

3.1.2 Peer Profile Management

Peers publish profiles via profile feeds and only the peer itself has authorisation on its own published content. Each peer is allowed to publish its profile via more than one online document while the authoritative peer also has the responsibility for synchronising its profiles stored on distributed servers. It is possible that a peer has more than one account on multiple OKBook servers and as mentioned above, each OKBook server will hold a copy of the registered peer's profile but will however not be granted write access to that profile. On the other hand, a single peer will be allowed to apply for more than one account on an OKBook server, but each OKBook server will treat different

[2] http://n2.talis.com/wiki/Changesets

accounts as belonging to different peers by default unless an alignment is applied. A peer may publish more IMs and gain more capabilities as time goes by and develop more new OKCs. Under this circumstance, it may update its profile and inform all OKBook servers, on each of which there exists an account of this peer, of the profile change.

From the perspective of profile updating, peers are publishers and OKBooks are subscribers and hence we can use the Publish/Subscribe model here to fulfil this updating task. Here, PubSubHubbub (PuSH) (Fitzpatrick et al., 2012), which is an open protocol for distributed communication based on this Publish/Subscribe model, has been employed with the assistant of the hub server [3] provided on the Google App Engine. Note that this server from Google is only for demonstrating how the protocol works and not a requirement by OKBook in essence. This way of managing profiles conforms to the spirit of decentralisation and distribution, and many other hubs have been built based on the same protocol by a variety of third parties such as Superfeedr [4], WordPress [5] and WebGlue [6], etc., at the time of writing this thesis. These third-party hubs behave like super peers deployed in the peer-to-peer environment and every peer has the chance to be a super peer if it is capable of providing a reliable Publish/Subcribe service. Inspired by PuSH, the protocol used by the OKBook platform for updating peer profiles is depicted in Figure 3.2.

This figure shows peers can have multiple accounts (three accounts in this case) on more than one OKBook server. Each server actually forms a peer community and has one or more groups which will be discovered dynamically based on the algorithm discussed in Section 3.1.3.2. Peer communities with diverse degrees of maturity form a dynamic and decentralised environment *per se*. All the profiles will be managed by peers who published them and stored in distribute. No peer has any centralised control over anyone of others' profiles except the publishers themselves. The Uniform Resource Locator (URLs) of peers' profiles are published via feeds and every time a peer applies for a new account with its feed, the OKBook server (as a subscriber to the peer's profile) will automatically subscribe to this feed via a hub whose role, as discussed above, can be played by any peer capable of understanding and parsing documents in various formats (e.g., RDF Site Summary or Really Simple Syndication

[3] http://pubsubhubbub.appspot.com
[4] http://blog.superfeedr.com/api/http/pubsubhubbub/pubsubhubbub/
[5] http://wordpress.org/extend/plugins/pushpress/
[6] http://github.com/zh/webglue/tree/master

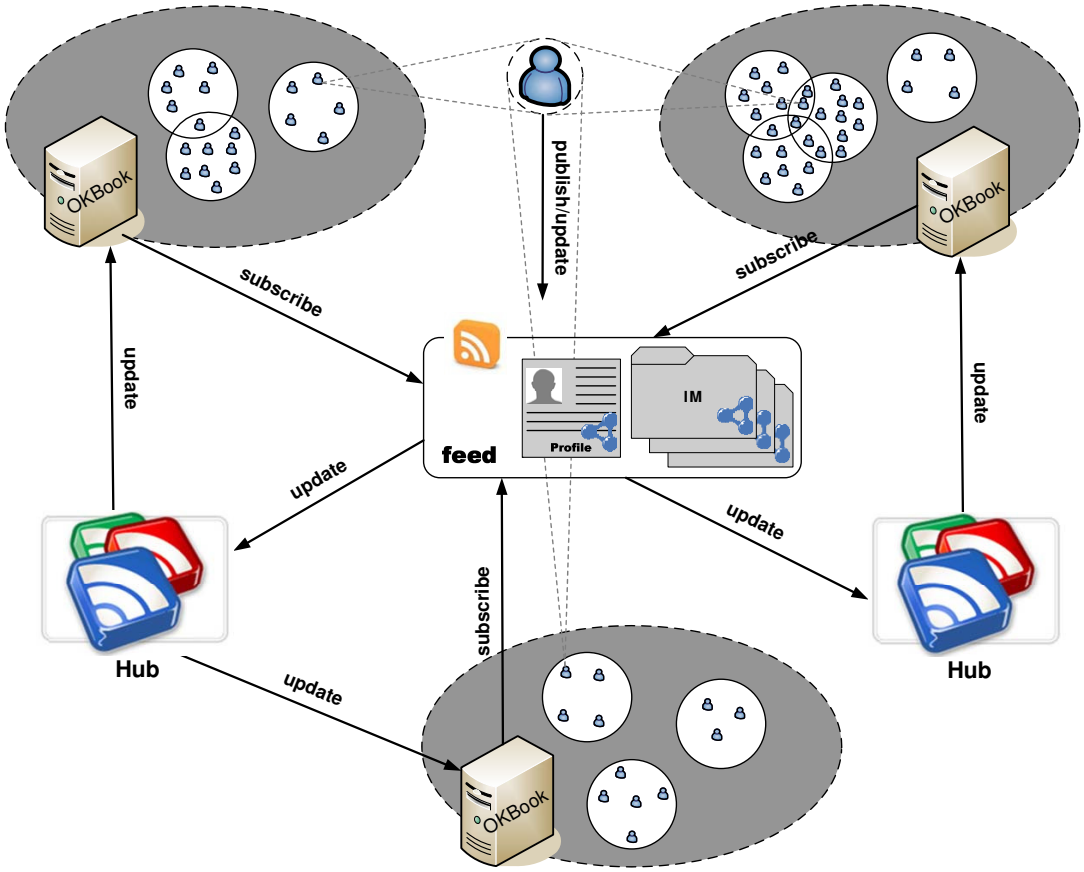


Figure 3.2: Peer profile updating protocol

(RSS) , or Atom, etc.). The profile updating will be automatically pushed to all the OKBook servers (by hubs) which have been holding its copies. After receiving an updating notification (a new feed entry) from a hub, an OKBook server will query its back-end database for the updater's ID and replace the profile content with the new one retrieved by dereferencing the URL parsed out from the OKBook seed. The Publish/Subscribe model was originally investigated in (Birman and Joseph, 1987) and its peer-to-peer version of a subscribe system supporting metadata and query language based on RDF was implemented in (Chirita et al., 2004). Compared with this system, OKBook has implemented the model in a different way: Firstly, super peers are OKBook servers themselves rather than servers on the publisher side, which are more expensive to maintain (because features such as always-online and changing less-frequently are usually required) for ordinary users so as to be untrustworthy in most cases; Secondly, subscription information is curated on a third party hub so neither OKBook servers nor servers behind publishers need to store the data, which would be otherwise too expensive to afford and too complicated to configure.

3.1.2.1 Linking Elements of IMs to the Web of Data

An embedded-metadata-based approach for republishing IMs in normal Web pages has been proposed and designed for our distributed knowledge sharing platform and its prototype has been implemented and will be further detailed in Chapter 5. OKBook employs and integrates this prototype, as shown in the *IM Republication Module* in Figure 3.1, aimed at linking the published IMs to the Web of Data. Thanks to this module, originally text-based IMs are republished using RDFa in HTML or XHTML ((X)HTML) Web pages.

Wikipedia[7] is a worldwide encyclopaedia which has 262 language editions and provides a knowledge base represented by natural languages *per se*. DBpedia (Auer et al., 2007) extracts structured information from Wikipedia and has been taken as a knowledge base represented by structured data in various formats such as Comma-Separated Values (CSVs) and RDF which links structured knowledge to the World Wide Web (WWW). DBpedia assigns `http://dbpedia.org/resource/Name`-like URIs to all entities that have been crawled from Wikipedia. A URI is used for identifying a specific entity and we here make use of URIs generated by DBpedia to annotate elements

[7] `http://www.wikipedia.org/`

inside IMs. Sometimes more than one URI indicates the same individual and they will appear differently within the annotation process in picking up a suitable URI and within the discovery process in matching annotations against the peer' profile. In order to allow IM publishers to efficiently find appropriate URIs to annotate IM elements, we take advantage of the DBpedia Lookup service which employs Lucene's string-similarity-based ranking and relevance metric having been discussed in (Kobilarov et al., 2009) to do the URI recommendation. As shown in Figure 3.1, the *URI Recommendation Service* wraps the above described functionality into an invokable service. This IM publication adheres to the four principles (Berners-Lee, 2006) of Linked Data because of the followings: URIs are used as annotations and they are dereferenceable HTTP URIs; employed URIs are curated by DBpedia and each of them comes with a RDF document and a human-readable Web page from Wikipedia; each IM becomes a RDF resource on the Web of data in the end and it will be also assigned with a URI linking to the URIs of other things on the Web. Note that the *Discovery Module* of OKBook also uses the above lookup service to help peers to refine their queries.

IM publishers are encouraged to use the property *rdfs:comment* to add details, which are more human-readable, about published IMs. Sometimes, users may want to use desired vocabularies or even their own. On that basis, three types of search services are offered to them for selecting diverse URIs to annotate IMs. The first service wraps in SWSEs such as Swoogle (Ding et al., 2004), Falcons (Cheng et al., 2008) and Sindice (Tummarello et al., 2007), which can crawl and index resources attached with URIs from the Web continuously. The second service wraps in search engines with generic purposes including Google and Yahoo!Search, which can assist users in refining and adjusting their keyword-based queries when too few relevant URIs are returned. The third service wraps in a co-references tool named as sameAs [5], which can help users find equivalent URIs of a specific one given by themselves. Figure 3.3 and Figure 3.4 illustrate screenshots on the above three services provided in the side block of OKBook plus an example search result for a query with the keyword "purchase" .

When users publish new IMs, the embedded triples are automatically and periodically harvested by the *Harvesting Module* as shown in Figure 3.1 and stored in the back-end database by OKBook with the help of the ARC2 library [6] afterwards. Obtained triples will be also exposed to users via a SPARQL endpoint based on HTTP bindings, which

[5] http://www.sameas.org/
[6] http://arc.semsol.org

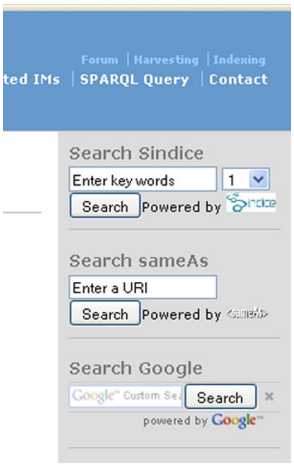


Figure 3.3: Services for selecting URIs for annotations

Figure 3.4: Screenshot on a search result

opens up an opportunity for other peers to reuse the RDF repository containing triples derived from republished IMs and establish their own applications of interest (e.g., IM mashups).

3.1.3 Discovery of IMs and Collaborative Peers

IMs describe choreographies between peers as protocols which guide peers to interact with one another and achieve cooperations among different service providers. Peers collaborate by subscribing to and running a specific IM but finding an appropriate IM for guiding themselves during interactions is a challenge. Keyword-based IM publication (Kotoulas and Siebes, 2007) limit IM discovery in the OpenKnowledge system due to the ambiguity aspect of human languages. Therefore, we connect to the broader Semantic Web discovery effort by using URIs to annotate IMs and based on this strategy, two mechanisms for discovering desired IMs and related collaborative peers are proposed in this section: meta-search-based discovery and peer-group-based discovery. The former mechanism is a generic solution for IM discovery and the latter one can discover other collaborative peers which have common interests shared with the service requester. The meta search works effectively especially when requesters' desired IMs are beyond the scope of common interests in their peer groups. The *Discovery Module* as shown in Figure 3.1 is in charge of fulfilling this IM discovery task on OKBook.

3.1.3.1 Meta-Search-Based Discovery

RDFa is one of serialisation types for RDF and RDF triples parsed out from the RDFa-embedded Web page can be indexed by SWSEs mostly targeting at RDF. OKBook provides a meta-search engine that allows the user to input queries and access several SWSEs with them. When a user submits a newly published IM to OKBook, the submission will also trigger a request message to be sent to SWSEs via submission mechanisms supported by them. For example, Sindice supports the Remote Procedure Call (RPC) ping Application Programming Interface (API) that was developed according to the specification of the Pingback (Langridge and Hickson, 2002) mechanism. So an IM submission or a submission of its revised version will ping Sindice for indexing or re-indexing this IM. On the OKBook query interface, users submit their queries to our meta-search engine by typing in related keywords. Unlike the original OpenKnowledge discovery service, instead of directly forwarding keywords to underlying SWSEs, the meta-search engine needs to preprocess them first since otherwise, documents related to these keywords that have nothing to do with interactions will be returned as well. OKBook expands users' queries about IMs by semantically attaching the type annotation such as *openk:InteractionModel*. Since one IM may be indexed by several SWSEs, the returned search results could contain overlapping IMs. By comparing URLs of the search results, we can group the overlapped IMs and just display one of them, with the indexing information from different provenances, to users. Moreover, their provenances will be also retained in case users want to further explore variant surrounding information (e.g., snippets of URIs) also indexed and provided by diverse SWSEs. The ratings given by other community members to displayed URIs are also displayed as references during URI selection. The reason for providing ratings is because as we mentioned before, we believe that users will trust recommendations from other peers (interest-driven) in the community more than advertisements from service providers (profit-driven).

On the search panel of OKBook, the URI recommendation service having been used in the *IM Republication Module* is applied as well within the query suggestion process, which helps users to refine their query phrases. When users type in keywords, the recommendation service will forward them to the DBpedia Lookup service which will calculate the similarities between these keywords and stored URIs and select out the most relevant URIs for users to refine their queries (a screenshot of this can be found later in Figure 3.11). OKBook will provide popup windows containing a human-

readable snippet for interpreting a focused URI without dereferencing this URI on the fly, which would otherwise cost more time and also damage the user experience. Since we use the same recommendation service in both the *IM Republication Module* and the *Discovery Module*, it is likely that both the IM publisher and the IM requester will choose the same URI for identifying the same item. Needless to say, this will benefit IM discovery because in this situation (a single URI is used for both annotating and querying), OKBook can just do a precise match between annotations and refined queries without employing any ontology matching or reasoning algorithms.

On the other hand, under the circumstance that the IM publisher and the IM requester refer to a single IM element using heterogenous URIs, we align them by employing the sameAs service which collects predicates of co-reference such as *owl:sameAs* from diverse vocabularies. Many other more sophisticated techniques have been proposed for ontology matching and these also could be used on the OKBook platform for assisting users in discovering more meaningful services. Discussion of this is however outside the scope of this thesis. Note that normally, different SWSEs use different ranking mechanisms so ranking results will be synthesised by the meta-search engine before being displayed finally to service requestors and the corresponding ranking algorithm will be discussed in Section 3.2.

3.1.3.2 Peer-Group-Based Discovery

Our peer-to-peer community will be established based on peers' interaction logs. As mentioned earlier, when enough peers fill the roles defined by an IM, the *Submitting Module* will forward the IM and relevant subscription information to each peer that will be involved in this interaction. After being executed, the IM along with the original subscription information will be sent back to OKBook which maintains a table to record the subscription log for each registered peer in the database. Peer groups will be discovered by analysing the interaction logs on the fly. For instance, an authorised peer will know peers with which it has been involved in an IM-driven interaction and what other IMs these peers have subscribed to. This will facilitate IM discovery so peers can find more IMs via which they may potentially interact with others. After letting OKBook analyse IMs in which its group members and itself have been involved, a peer can subscribe to any of these IMs by claiming to play a specific role inside. This method was actually inspired by the idea behind FOAF. If *PeerA* has interacted with

PeerB, it is possible that *PeerA* will be also interested in other interactions not having involved itself but have involved *PeerB* and for *PeerB*, the similar thing will happen in the reverse way. Therefore, we use this method to group peers and name it the Interactions From An Interaction (IFAI) method. Figure 3.5 depicts the discovery process as well as the discovered peer group.

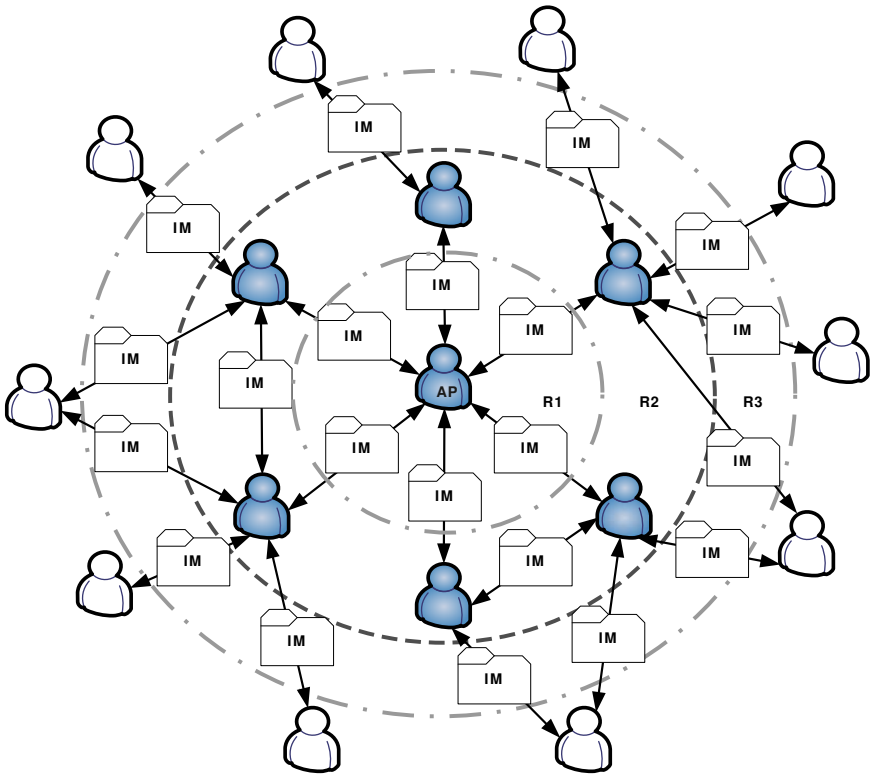


Figure 3.5: IFAI-based peer group discovery

In this figure, peers are linked to one another through historical interactions (stored as interaction logs) in which they were involved. The IMs in *RegionR*1 describe interactions in which authorised *PeerAP* previously took part. Then in terms of these IMs, *PeerAP*'s previous collaborative peers are discovered and form a group covered by *RegionR*2. The IMs in *RegionR*2 but not in *RegionR*1 describe previous interactions between *PeerAP*'s group members. On the other hand, the IMs in *RegionR*3 but not in *RegionR*2 describe previous interactions between *PeerAP*'s group members and external peers. Since IMs in *RegionR*3 but not in *RegionR*1 reflect interests of *PeerAP*'s group members as well, they are taken as a portion of results after IM discovery is finished. Actually, IFAI provides a peer with a screenshot of its dynamically generated group in terms of its historical interactions with other peers. Group members are updated when OKBook is fed in with new interaction records by each involved

peer. Displayed IMs are also followed by aforementioned rates given by the community members. The algorithm for the peer-group-based discovery is described in Algorithm 1. Note that Algorithm 1 describes a situation in which just peers directly known by *PeerAP*'s friends are taken into account. Actually, peers are allowed to do a deeper search depending on their preferences by invoking our IFAI method, which is helpful especially for who newly registered and have not had many friends connected to themselves. This can be also achieved by making use of newly discovered friend peers and running Algorithm 1 recursively.

Algorithm 1: IFAI Algorithm (single step)

Input: the URI of current authorised peer, $apeer\_uri$ and the interaction log, $record$.
Output: URIs of group members, $fpeer\_uris$ and URIs of IMs these members were involved in, im_uris.

```
begin
    IM_URIs = getInvolvingIMs (apeer_uri, record);
    for each im_uri ∈ IM_URIs do
        partner_peer_uris = getInvolvedPeerURIs(im_exec);
        for each peer_uri ∈ partner_peer_uris do
            if apeer_uri equals peer_uri then
                continue;
            else
                fpeers = fpeers ∪ {peer_uri};
                IM_URIs' = getInvolvingIMs(peer_uri, record);
                for each im_uri' ∈ IM_URIs' do
                    if im_uri' ∈ im_uris then
                        continue;
                    else
                        im_uris = im_uris ∪ {im_uri'};
```

3.1.3.3 Subscription Information Submissions and Feedback

After an appropriate IM is discovered, peers can subscribe to it via the *Subscription Module* as shown in Figure 3.1. In OKBook, the *Subscription Module* then send subscription-related data (such as which peer fills which role) along with the subscribed IM itself to all the involved peers, including the one that has been equipped with LCC interpreter in charge of bootstrapping the interaction and executing the IMs coded in LCC. In order to comply with this submission of subscription information and the LCC interpreter, when a peer decides to subscribe or unsubscribe to an IM, OKBook will record relevant information such as the peer's user account, the URI of the

target IM, the time when this subscription occurs as well as some auxiliary information such as the peer's contact details. Before the submission, OKBook also checks if each role in the being submitted IM is filled by at least one peer. Figure 3.6 shows how OKBook helps registered peers interact with one another based on IMs along with the OpenKnowledge architecture. From this figure, we can see that OKBook has replaced the Distributed Discovery Service (DDS) and provides peers with strong discovery power gained via resources shared on the Web of data.

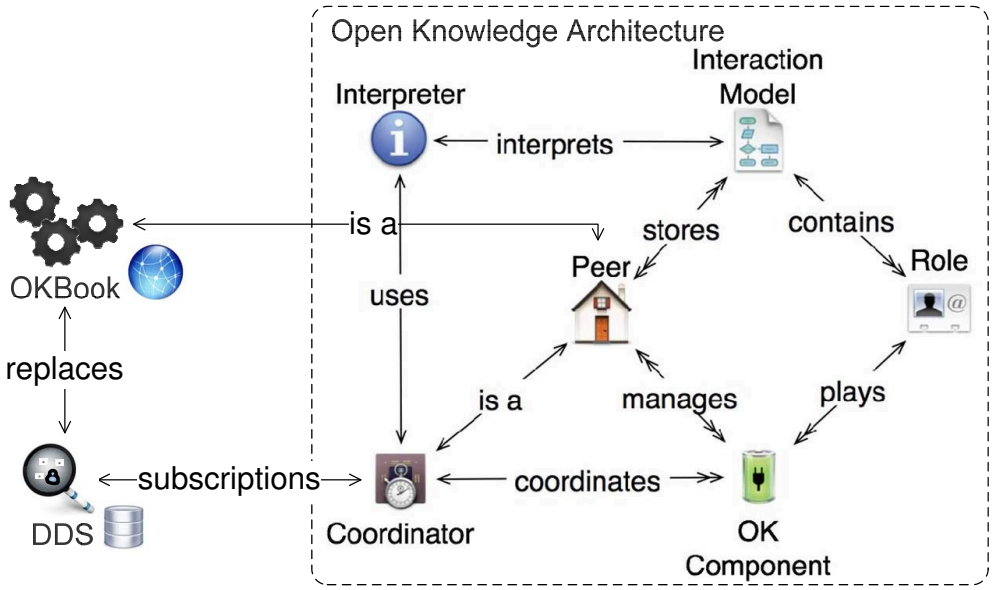


Figure 3.6: OKBook-equipped OpenKnowledge system

3.2 Ranking on OKBook

More than one IM may have annotations in common or describe similar tasks, so IM discovery is followed by ranking in order to assist users in filtering out search results of less interest. This section is focused on the ranking strategies so far applied on OKBook, including ranking IMs, ranking peer communities and ranking peer group members.

3.2.1 IM Ranking Criteria

For each input query phrase inside each discovered peer group, there may be more than one IM selected out and recommended to users by OKBook. Under this circumstance,

a ranking strategy is crucial for peers to select and subscribe to the appropriate IMs. In this subsection, different ranking criteria are discussed.

3.2.1.1 Ranking Discovered IM Over the Web of Linked Data

For the IMs discovered by our meta-search engine, assuming that different search engines may use different ranking mechanisms, the returned ranking results should be reconciled before being finally displayed to users because they usually expect only to see a single rank. Suppose Q denotes a querying phrase that a user types into our meta-search engine and U denotes the URL minted for a specific IM indexed by one or more SWSEs; S_i denotes the ith SWSE involved in the meta-search and the rank of this URL returned by S_i is denoted by $rank_{S_i}(U,Q)$. If S_k has not previously indexed U, then $rank_{S_k}(U,Q) = 0$. We take the weighted average of ranks of U returned by different search engines as the overall rank of U on our meta-search engine, which will be calculated by the following equation:

$$rank(U,Q) = \frac{\sum_{i=0}^{N} \chi_i \times rank_{S_i}(U,Q)}{N} \qquad (\sum_{i=0}^{N} \chi_i = N)$$

Here, N denotes the overall number of search engines on which our meta-search engine is built. χ_i denotes the weight on the rank returned by S_i and it can actually reflect the user's preference. And χ_i is equal to one by default. However, if a user prefers some search engines over others, he/she can inform our meta-search engine of this preference by modifying values of weights from the OKBook search panel. Using the above equation, ranks of all URLs can be synthesised and shown to users, in descending order of relevance. If more than one URL has the identical ranking value, they will be displayed together as a group of the same priority on the result page. Nonetheless, their provenances will be retained in case users want to further explore search-engine-specified information (e.g., snippets of search results) provided by diverse SWSEs.

3.2.1.2 Ranking Discovered IMs Via the Interaction Graph

For the IMs discovered via our group-based algorithm, they will be displayed along with their interaction participants on the result page. Apparently, the deeper the search depth, the more IMs may be discovered with Algorithm 1 described in Section 3.1.3.2. Thus an appropriate ranking strategy is also required by requesters for filtering out

appropriate IMs. In OKBook, IMs and their interaction participants are ranked in terms of search depth. The IMs discovered through the requester's intimate peers (include the requester peer itself) get higher ranks (with shallower search depths) and the IMs discovered through the requester's distant peers get lower ranks (with deeper search depths). For instance, in Figure 3.5, IMs in *RegionR*1 have a higher rank than IMs in either *RegionR*2 or *RegionR*3. This ranking strategy is based on the intuition that closer friend peers normally have more common interests in particular interactions and if a peer was involved in a running of a specific IM before, there will be more chance for this peer to be involved in the same IM again sometime in the future. There may be more than one IM discovered in the same depth and we need further rank them based on their occurrences which have been pre-stored in the requesters' profiles according to the architecture design.

3.2.2 Other Rankings

Currently, OKBook provides the above two IM ranking mechanisms. In addition to there, ranking other OKBook related objects or possibly ranking IMs interactively might be also interesting to users and will be further investigated in here.

3.2.2.1 Ranking IMs Based on the Interaction Feedback

Some feedback could be sent back to the OKBook server which triggered the interaction after all the involved peers have also sent back finishing signals which will be discussed in Section 4.2.2. Then IMs can be possibly ranked based on whether or not interactions were finished successfully. However, failure is not necessarily because the IM has been designed badly and it may be due to a peer which subscribed to this IM but did not satisfy all the constraints which it was supposed to satisfy. Under this circumstance, this peer might be blamed rather than the IM itself or perhaps even other peers might be blamed for putting the "failed" peer in an impossible position. Thus attribution of blame for IM failure is complex and imprecise although it may serve a purpose.

3.2.2.2 Ranking Peer Communities

Peer communities (hosted on OKBook servers) can be ranked based on the number of community members (i.e., how many peers have registered) or the number of peer activities (i.e., how many interactions have occurred) or both. Due to the various user preferences, it is difficult if not impossible to aggregate these two methods and provide users with a unified ranking result. So OKBook could allow users to see both above ranking results separately and choose the preferred one to harness.

3.2.2.3 Ranking Peer Group Members

After an IM is executed, some feedback will be sent back to the OKBook server which previously triggered the running of the interaction. The feedback indicates whether the interaction was finished successfully or not. If not, it also informs which peer should be responsible for the failure. This information is harnessed by OKBook servers statistically and will be used for peer ranking. Each OKBook server maintains a peer ranking list in which peers are ranked based on values indicating how many times they have been involved in successfully-finished interactions and how many times they have not. In the end, peers with higher ranks are displayed at the top of the final ranking list. The details of peer groups will be discussed later in Chapter 6.

3.2.3 Extended Open Graph Protocol

As also mentioned above, after being discovered, more than one IM may become candidates to which the logged-on user considers to subscribe. Beside referring to the ranking criteria supported by OKBook, users can also access the IM page and make further investigations there. But what extra information should be provided to users attempting at making right decisions? Here, an Extended Open Graph Protocol (EOGP) is proposed as an answer to this question.

In terms of successfulness of social network sites such as Facebook, Flickr and Twitter, users are usually not only simply connect to other users by hyperlinks but also establish connections via things *per se*. In other words, the social relationships are built on top of things which users are interested in and also want to share or talk about with their friends. OKBook has been built as a platform for sharing knowledge via

interactions among peers and on that basis, we put IMs into the social graph of peers. Open Graph Protocol (OGP) has been designed for Web pages representing profiles of real-world things such as movies, sports teams, celebrities and restaurants (Zuckerberg and Taylor, 2010). This protocol was originally created in the Facebook company and focused on the bidirectional relationships of users. In this thesis, EOGP is inspired by OGP and adopted in the peer-to-peer knowledge sharing environment on OKBook. Here, we refine the peer relationships by employing the subscribing (a.k.a., following) model. Figure 3.7 gives a general depiction on EOGP, based on which peers have been connected via IMs and notably, it is possible that a peer outreaches its community (an OKBook server) and links to an IM originating from another community.

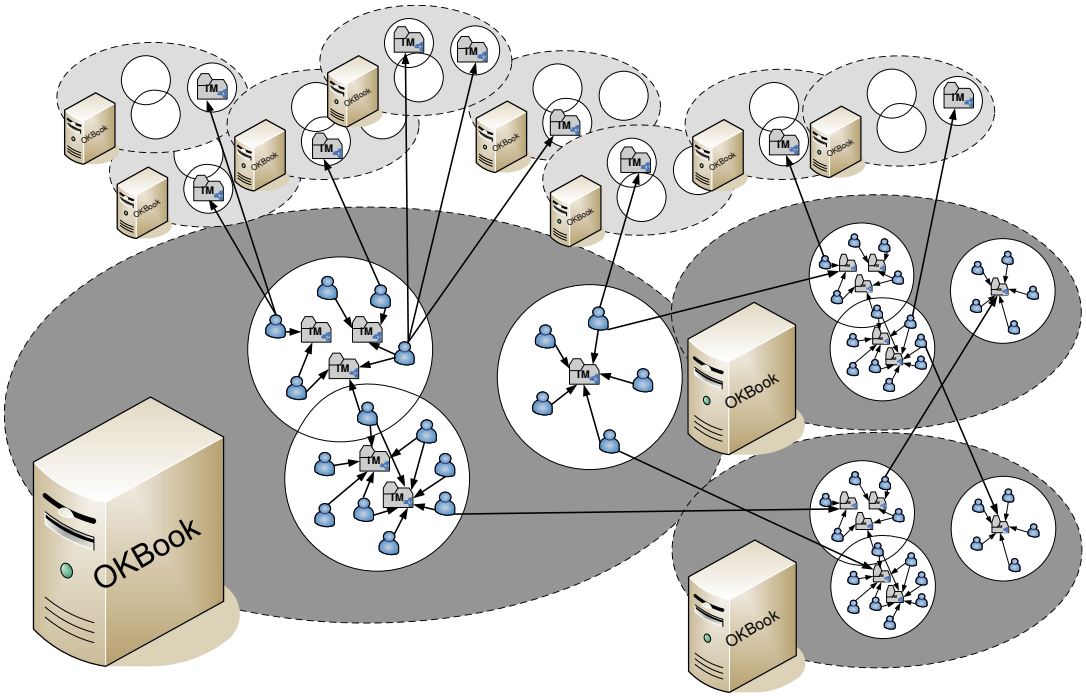


Figure 3.7: Extended Open Graph Protocol (EOGP)

When peer profile updating happens, as described in the Section 3.1.2, based on the PubSubHubbub protocol, the new profile will be propagated to all the OKBook servers on each of which the profile owner holds an account. Since IMs are published on normal Web pages (Bai and Robertson, 2010), we implement EOGP by forwarding the IM pages with RDFa annotations to the OKBook servers. Suppose a peer has been logged on to at least one OKBook server and is browsing a specific IM. The browser can identify this peer via cookies and forward the IM to known OKBook servers which query the backend subscription database with the URI of being browsed IM for infor-

mation about which peers were subscribed to the IM and among these peers, which of them are following that peer looking at the IM page and which of them are followed by this peer. Since different OKBook servers (as different peer communities) may have different peers register on themselves, even if each community has an identical copy of a peer's profile, the result of the above query may or may not be the same. Therefore, the query result will be sent back asynchronously and split in separate community sections on the IM page which the peer is currently browsing depending on the response time and policies of different OKBook servers. Each community section represents the peer's social groups given by a particular OKBook server. The first group contains peers which the browsing peer is following and the second group contains peers which are following the browsing peer and were once subscribed to the target IM. The third group contains peers which have no connection with the browsing peer but were also once subscribed subscribing to this IM. Note that the above groups play different roles when the browsing peer is considering whether or not to subscribe to the target IM. Peers in the first group are normally trusted by the browsing peer while peers in the second group are less trustworthy but they will contribute to the group's awareness of the browsing peer. The third group provides peers with which it is potentially worth interacting and, hence, the peer currently browsing the IM page has a chance to follow the members in this group by shifting them to the first group (note this will trigger the updating of its profile). In each group, the further details on group members in terms of the target IM are rendered as well, such as which role(s) they were playing during the previous running of this IM as well as individual comments on those interactions.

The above discussion shows that EOGP can assist peers in investigating IMs and subscribing to decent ones in a social and interactive way. It offers a page-to-page solution that addresses the problem of IM sharing in a decentralised environment. Because IM pages have employed metadata-embedding approaches recommended by W3C, more and more service providers (also peers) are highly likely to join this party by linking their data (including services) to these pages. The user experience is also of concern and when an IM page is rendered in the browser, the embedded annotations will be automatically detected. By dereferencing those annotation associated with URIs, users can see more detailed information about elements inside the IM so as to make a better decision on whether or not to subscribe to it. This, however, raises users interface issues also encountered by the multi-agent Systems and the discussion on this is beyond the scope of this thesis.

3.3 Inference Driven Evolution of the Peer Community

A single peer may play different roles defined in different IMs so as to have different obligations and therefore, which role a peer is able to play does not really influence the current IM discovery process. However, as long as IMs that meet a peer's requirements have been discovered, it would be better that the peer can be immediately informed of which roles defined in the discovered IMs it is capable of playing in the near future. As mentioned earlier, this discovery will be fulfilled by the *Discovery Module* of OKBook according to the queries issued on the fly over the peers's profiles along with annotated IMs using EOGP. Every time a new peer signs up on a specific OKBook server, it will not only get its user account but also be required to upload its own profile. After potentially useful IMs are found, a peer can choose to further see the analysis of each of them (this type of analysis is done by OKBook automatically) before actually subscribing to any of them. OKBook will query the peer's profile as well as originally embedded RDF triples harvested from a particular IM page being browsed by this peer in order to figure out which role can be filled. If the peer is currently not subscribing to this IM, it can simply carry out the subscription by filling a displayed role it expects to play based on the discovery done by OKBook. Otherwise, it will be informed of the current subscription to the same IM and has a chance to unsubscribe by giving up filling a role. Matching the peer's profile with the annotated IM is a basic inference rule and other inference rules can be created by engineers using SPARQL via a reliable discovery server (such as an OKBook server itself). Several basic inference rules are listed as follows (expressed via the First Order Predicate Logic and $\vdash_{cst}$ means if the right-hand side predicate(s) does (do) not hold, new triples will be constructed and added to the Knowledge Base (KB) to make the predicate(s) hold):

I. $has\_role(IM,R), can\_play(P,R) \vdash_{cst} can\_subscribe\_to(P,IM)$

Rule I is related to the subscription suggestion (role checking) which will be done by OKBook automatically. Here, P denotes a peer; IM denotes an interaction model; R denotes a role. This rule means if P can play R defined in IM, P is able to subscribe to IM. For instance, this rule can be applied inside the *Subscription Module*, as shown in Figure 3.1, to help a peer to select its desired role recommended on a specific IM page when subscribing to this IM. *Rule I* can be applied and interpreted using the following SPARQL-like queries (SPARQL queries mingled with pseudo code):

---

result_set = SELECT ?peer ?interaction_model

```
WHERE {
    ?interaction_model a openk:InteractionModel.
    ?peer a openk:Peer.
    ?peer openk:_IM ?interaction_model
    ?peer openk:can_play ?role.
    ?interaction_model openk:has_role ?role
    }
}
for each result in result_set
    if result.getURI("role") == null
        CONSTRUCT {
  ⟨result.getURI("peer")⟩ openk:can_subscribe_to ⟨result.getURI("interaction_model")⟩.
        } WHERE{}
    end-if
end-for
```

II. $publish\_IM(P,IM) \vdash_{cst} can\_play(P,R), has\_role(IM,R)$

Rule II is related to the question "what will drive peers to publish new IMs and what kind of peers are willing to fulfil this publication". It means that if P is the publisher of IM, IM contains R which P can play. A potential motive behind creating this rule is that a peer has a requirement but he/she cannot find an appropriate IM to subscribe to in the current network. Under this circumstance, the peer can create a new IM that can meet the requirement and meanwhile, this peer can apparently play a role as a requester in this IM. For instance, this rule can be applied inside the *IM Republication Module*, as shown in Figure 3.1, and each publisher will therefore have a chance to subscribe to an IM authored by him/her soon after it is published. *Rule II* can be applied and interpreted using the following SPARQL-like queries:

```
result_set = SELECT ?peer ?interaction_model ?role
WHERE {
    ?interaction_model a openk:InteractionModel.
    ?peer a openk:Peer.
    ?peer openk:publish_IM ?interaction_model
    OPTIONAL {
        ?peer openk:can_play ?role.
        ?interaction_model openk:has_role ?role
    }
}
for each result in result_set
```

if result.getURI("role") == null
CONSTRUCT {
⟨*result*.getURI("peer")⟩ openk:can_play _:user.
⟨*result*.getURI("interaction_model")⟩ openk:has_role _:user
} WHERE{}
end-if
end-for

III. $publish\_IM(P,IM), belong\_to(IM,C) \vdash_{cst} holdsAccount(P,U) \wedge has\_member(C,U)$
IV. $can\_play(P,R), has\_role(IM,R), belong\_to(IM,C) \vdash_{cst} holdsAccount(P,U) \wedge has\_member(C,U)$

Rule III and *Rule IV* are related to peer community membership. The former means if P is the publisher of IM which belongs to a peer community, C, P should hold an account, U, which also belongs to C; the latter means if P can play R defined in IM which belongs to C, P should hold an account U which also belongs to C. The above two rules can be applied in a situation where an OKBook server wants to expand the peer community hosted by itself and as long as a peer's profile satisfies either of the above heads, the server will send an invitation to this peer and create an account if approved (an active way of community expansion). Note that each role can be subscribed by several peers and each peer may hold multiple user accounts in different communities. *Rule III* can be applied and interpreted using the following SPARQL-like queries:

---

result_set = SELECT ?peer ?community ?user
WHERE {
?peer a openk:Peer.
?interaction_model a openk:InteractionModel.
?community a openk:P2PCommunity.
?peer openk:publish_IM ?interaction_model.
?interaction_model openk:belong_to ?community.
OPTIONAL {
?peer foaf:holdsAccount ?user.
?community sioc:has_member ?user.
}
}
for each result in result_set
if result.getURI("user") == null
CONSTRUCT {
⟨*result*.getURI("peer")⟩ ?peer foaf:holdsAccount _:user.

```
        ⟨result.getURI("community")⟩ sioc:has_member _:user
      WHERE{}
  end-if
end-for
```

, while *Rule IV* can be applied and interpreted using the following SPARQL-like queries:

```
result_set = SELECT ?peer ?community ?user
WHERE {
  ?peer a openk:Peer.
  ?interaction_model a openk:InteractionModel.
  ?community a openk:P2PCommunity.
  ?peer openk:can_play ?role.
  ?interaction_model openk:has_role ?role.
  ?interaction_model openk:belong_to ?community.
  OPTIONAL {
    ?peer foaf:holdsAccount ?user.
    ?community sioc:has_member ?user.
  }
}
for each result in result_set
  if result.getURI("user") == null
    CONSTRUCT {
      ⟨result.getURI("peer")⟩ ?peer foaf:holdsAccount _:user.
      ⟨result.getURI("community")⟩ sioc:has_member _:user
    WHERE{}
  end-if
end-for
```

$V.\ has\_constraint(R,T), can\_satisfy(P,T) \vdash_{cst} can\_play(P,R)$

$VI.\ has\_annot(IM,A), has\_annot(C,A) \vdash_{cst} belong\_to(IM,C)$

Rule V and *Rule VI* are two auxiliary rules dedicated to community formation. The former means if P can satisfy every constraint, T, associated with R, P can play R; the latter means if every annotation, A, which belongs to IM, also belongs to C, IM belongs to C. *Rule V* is used for deciding if a peer can play a specific role and for instance, it can be applied inside the *Subscription Module*, as shown in Figure 3.1, to check if a peer is capable of subscribing to an IM or not. *Rule VI* is used to determine if an IM originates from a specific community and for instance, it can be applied in-

side the *Discovery Module*, as shown in Figure 3.1, to recommend IMs based on the community to which they belong. Both of them involve multiple join queries so the corresponding SPARQL-like queries are not listed here for the sake of brevity.

VII. $has\_role(IM,R), has\_annot(R,A) \vdash_{cst} has\_topic(IM,A)$

VIII. $has\_topic(IM,A), belong\_to(IM,C) \vdash_{cst} has\_topic(C,A)$

Rule VII and *Rule VIII* are about finding topics of a community or topics of an IM by investigating annotations attached to a specific IM. The former means if *IM* defines *R* which has *A*, *IM* takes *A* as its topic and a situation where this rule can be applied is that after IMs are published, search engines can index them by their topics (in this case, annotations of their roles); the latter means if *IM* has *A* and belongs to *C*, *C* also takes *A* as its topic and a situation where this rule can be applied is that a peer can join new communities based on their topics also related to those IMs to which it was subscribed (a passive way of community expansion). The set of annotations belonging to a community is here called the *community signature*. *Rule VII* can be applied and interpreted using the following SPARQL-like queries:

---

result_set = SELECT ?interaction_model ?annotation
WHERE {
  ?interaction_model a openk:InteractionModel
  ?role a openk:Role.
  ?annotation a openk:Annotation
  ?interaction_model openk:has_role ?role
  ?role openk:has_annotation ?annotation.
}
for each result in result_set
  flag = ASK WHERE {
⟨*result*.getURI("interaction_model")⟩ openk:has_topic ⟨*result*.getURI("annotation")⟩}
  if flag == false
    CONSTRUCT {
⟨*result*.getURI("interaction_model")⟩ openk:has_topic ⟨*result*.getURI("annotation")⟩}
    WHERE {}
  end-if
end-for

Rule VIII can be applied and interpreted using the following SPARQL-like queries:

---

result_set = SELECT ?community ?annotation
WHERE {

```
    ?interaction_model a openk:InteractionModel.
    ?annotation a openk:Annotation
    ?community a openk:P2PCommunity.
    ?interaction_model openk:belong_to ?community.
    {?interaction_model openk:topic ?annotation}
    UNION {?community openk:has_topic ?annotation}
}
for each result in result_set
    flag = ASK WHERE {
⟨result.getURI("community")⟩ openk:has_topic ⟨result.getURI("annotation")⟩}
    if flag == false
        CONSTRUCT {
⟨result.getURI("community")⟩ openk:has_topic ⟨result.getURI("annotation")⟩}
        WHERE {}
    end-if
end-for
```

Inferences rules like the above will be fulfilled on the fly and more RDF triples may be generated thereafter due to the inference. In order to optimise the future discovering process, these new triples can be merged with the involved peers' original profiles or indexed on the discovery server for reuses. Since triples are used for representing peers' knowledge as well as annotating IMs, we make the inference take advantage of SPARQL for composing rules. For satisfying each rule here, the inference system will first check if the current peer's profile can satisfy the rule's head or not. If the head is satisfied (corresponding triples exists), the inferencer will further check if triples inside the profile can satisfy the body or not. If the body is not satisfied (corresponding triples are not found), new triples will be created in order and added to the peer's profile to make the body satisfied.

3.4 Analysis and Comparison Against the OpenKnowledge Architecture

After the running of a specific IM, each involved peer has a chance to extend its social graph by establishing new relationships with other involved peers. This interaction-

based community expansion is intended to encourage peers to interact with others they have not been in touch with before and also make them benefit from other potential interactions in the future. In this section, we give an analysis of how peers can benefit from the community formation fulfilled based on our approach.

3.4.1 Acquiring IMs From Discovered Group Members

Based on EOGP, OKBook provides peers with not only interactions in which they were involved but also interactions in which their participants were involved. These interactions can be taken as expansion seeds via which peers are likely to interact with others whom they may have found it difficult if not impossible to know only based on the searching mechanism offered by traditional Web sites through key-word search. Suppose Alice bought a product from Bob, via the trade IM described in Figure 3.8, using OKBook. This figure depicts an interaction in which a client purchases a product referenced by a product code from a shop using his/her credit card.

/∗ A client, *C*, sends out a message to a potential shop, *S*, in order to buy a product with a code, *PC*, using his/her credit card, *CC*. Then *S* sends the receipt of the product back to *C*. When *S* receives the message from *C*, it checks if *CC* has enough credit to afford the product. If the credit is enough, *S* completes *C*'s order by generating a receipt and send it back to *C*. Note there is one peer playing the client role and may be more than one peer playing the shop role. ∗/

```
a(client(PC, CC), C)::
   buy(PC, CC)⇒ a(shop, S)← payby(CC)∧ lookup(S) then
   receipt(R)⇐ a(shop, S)

a(shop, S)::
   buy(PC, CC)⇐ a(client(_),C) then
   receipt(R)⇒ a(client(_), C)← enough_credit(CC, PC)∧
                                complete_order(PC, CC, R)
```

Figure 3.8: Simple trade IM in LCC

Focused on this IM, the sequence diagram in Figure 3.9 shows how the group-base IM discovery can drive peers' interactions. Suppose Bob is a retailer who bought the same product from another peer Carol (the original manufacturer of this product) by

paying her cash via another IM similar to the above one in the past. By logging on to OKBook, Alice can reach and subscribe to the latter IM in terms of the automatically discovered peer groups. When Alice intends to buy the same product in the future, she has a chance to interact with other peers such as Carol via the latter IM instead of with Bob via the former one. It is likely to happen that Alice will get a lower price from Carol this time. On the other hand, from the perspective of Carol, OKBook assists her in discovering a new customer. Once their interaction is finished, Alice's group will be enlarged by absorbing a new group member (Carol) and a new IM (the latter one).

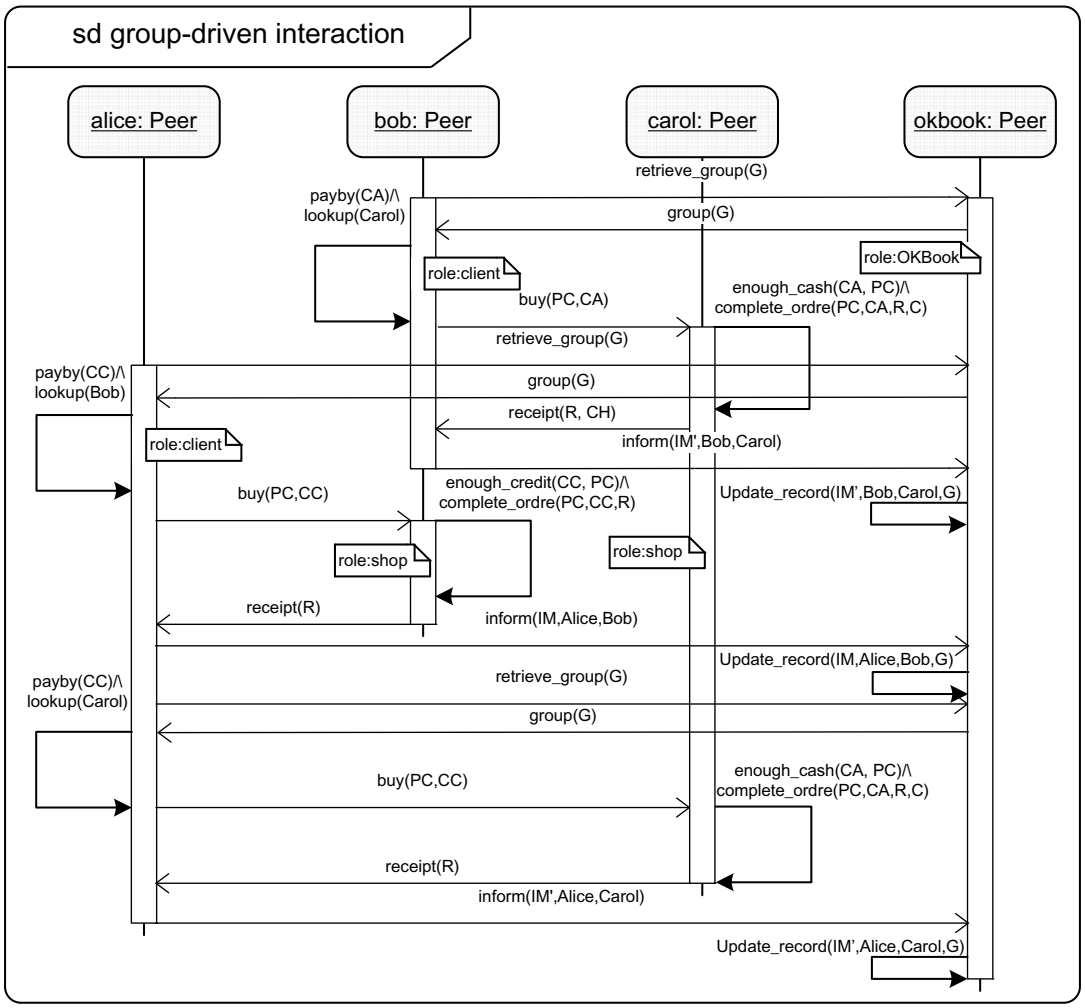


Figure 3.9: Sequence diagram for a bunch of interactions driven by peer groups

3.4.2 Peer Subscriptions and IM Consumptions

Conventionally, when a user accesses to a shopping Web site such as eBay.com, he/she searches for a required product by typing in relevant keywords via the search User Interface (UI). But this only happens under the precondition that providers have already logged on to this Web site and published adverts for this kind of products on the server. On the other hand, keyword-based search has its natural limitations due to the synonymity and the ambiguity of phrases. Employed as annotations, URIs provide unambiguous identifiers to concepts that convey meanings users want search services to be truly knowledgeable about. Moreover, compared with the OpenKnowledge system which requires a coordinator to look after subscription information about the participating peers and make them a team, on the OKBook platform, all a peer needs to do is search for an appropriate IM (recommended by the OKBook discovery service), subscribe to it and run the IM on itself without any help from intermediate coordinators. Then OKBook will try to find other peers automatically who can collaborate with this peer and start the interaction. For example, even if there is no provider providing the desired product for the time being, the subscription of this peer will be still valid for a period of time (each subscription has an expiry time, a.k.a., timeout). As soon as enough collaborative peers have subscribed to an IM, this user will be informed and meanwhile, the IM goes into the execution procedure. However, for conventional Web sites, this temporal "no provider" is likely to end up with a page indicating a HTTP "not found" (404) error.

Figure 3.10 gives the excerpt of proportional source codes of a Web page on which the trade IM described in Figure 3.8 has been published. The annotation strategy employed here will be detailed in Chapter 5. In Figure 3.11, OKBook helped a peer consume a Web page on which an IM has been republished by informing this peer of which roles it can play when the *Analyse* button was pushed. As shown at the bottom right of this figure, peers can choose to make OKBook conduct the analysis as soon as they get the search result through the search panel. As mentioned previously, this analysis was done according to the peer's local profile as well as the harvested metadata. With the advance of Linked Data, more and more client-side applications will come up and be able to parse and harness embedded metadata in a variety of ways, and further discussion of this is outside the scope of this thesis.

```
<html
xmlns=''http://www.w3.org/1999/xhtml''
xmlns:openk=''http://homepages.inf.ed.ac.uk/s0896253/openk.owl#''
xmlns:dbpedia=''http://dbpedia.org/resource/''
>
...
<div typeof=''openk:InteractionModel''>
 <span property=''openk:has_declaration''>r(client, initial, 1, 1)
 </span><br/>
 <span property=''openk:has_declaration''>r(shop, necessary, 1)</span>
 <div rel=''openk:has_role''>
  <div typeof=''openk:Role'' property=''openk:has_roletype''
  content=''initial''>a(<span property= ''openk:has_name''>client
  </span>(<span rel=''openk:has_arg'' typeof=''openk:Argument
  dbpedia:Universal_Product_Code''>
  <span property=''openk:has_name''>PC</span></span>), C)::<br/>
  <span rel=''openk:sendout''>
   <span typeof=''openk:Message''>
    <span property= ''openk:has_name''>buy</span>(
    <span rel=''openk:has_arg''>
     <span typeof=''openk:Argument dbpedia:Universal_Product_Code''>
      <span property=''openk:has_name''>PC</span></span>,
     <span typeof=''openk:Argument dbpedia:Credit_card''>
      <span property=''openk:has_name''>CC</span></span>)
...
```

Figure 3.10: Republished trade IM in XHTML

3.5 OKBook Federation

As already discussed, in a cluster of OKBook servers, each of them will curate and update the profile belonging to a particular registered peer. For other servers to whom this peer is agnostic, when other peers search on these servers, there will be no result related to this being searched peer even though it may be a potential collaborator in some interactions. Therefore, in order to improve the recall of collaborator searching, OKBook servers need to share their knowledge (of course at their own discretion) and clustering and federation are two ways to achieve this sharing. The former may require only low-level communications between OKBook servers which will end up with running the same native code, and is easier to implement due to the tightly coupled design. However, there could coexist OKBook servers which are implemented in diverse lan-

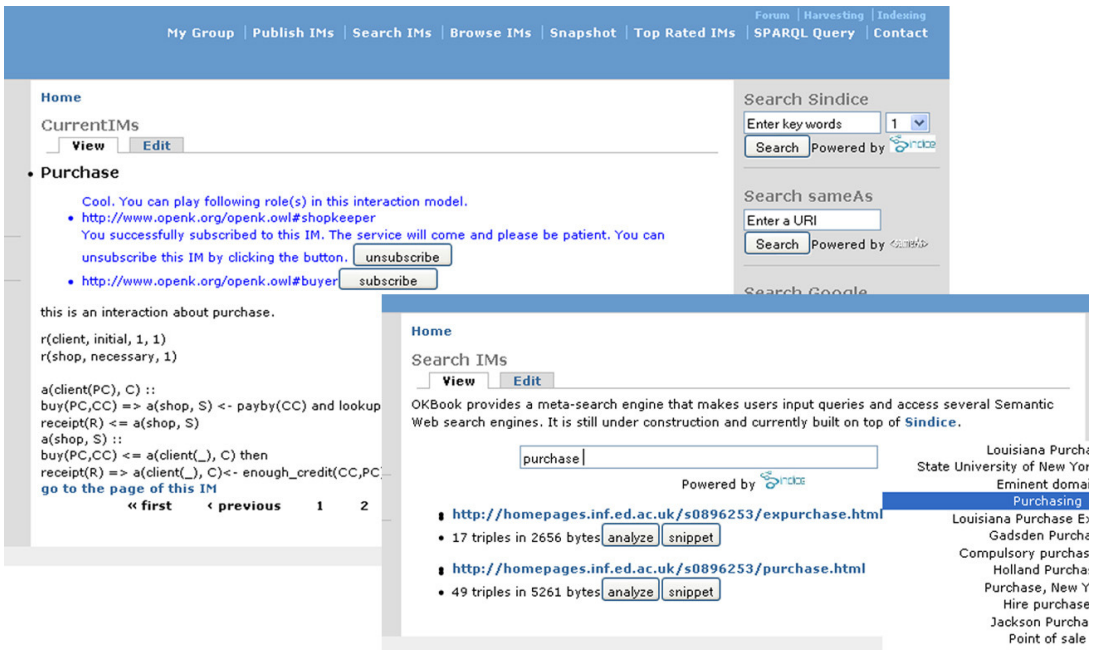


Figure 3.11: Consumption of a republished IM

guages processed by different processors with their own sets of instructions. More than one cluster could therefore exist and it is impossible to merge them into a new cluster (in which all the OKBook servers share the same native code). The latter way is comparatively more decentralised (the native code will not be called and transparent to other OKBook servers) and makes OKBook servers loosely coupled only if the federation protocols or APIs are carefully designed. Since clustering is a centralised option and goes against the spirit of sharing knowledge in a decentralised or distributed environment as OKBook targets, we chose the latter way in this thesis to improve accessibility and interoperability of peers registered on various OKBook servers. As shown in Table 3.1, a REpresentational State Transfer (REST)ful API design is employed here to allow multiple OKBook servers to call each others' services and update the data they want to federate. For instance, an OKBook server can retrieve all the interactions, which are instantiated by the same IM with the encoded URI, `encodedURI` and triggered by another OKBook server with the domain, `example.com`, by dereferencing the following URL:

`http://example.com/apis/1/federation?resource=pi&resource=encodedURI`

Within federation processes, PubSubHubbub is also employed, by which an OKBook server can push its updates on shared data to others and at the same time, receive notifications of updates from others. This combination between RESTful APIs and PubSubHubbub (i.e., the subscriber subscribes to a RESTful service provided by the

publisher via a hub) can cut down the traffic related to federated synchronisation between different OKBook servers.

Table 3.1: OKBook Federation APIs

Query Path		
`http://OKBOOK_HOST/apis/VERSION_ID/federation`		
Query Parameters		
Requirement	ParaName	ParaValue
Mandatory	`resource`	`im` (IMs) or `peer` (peers) or `pi` (interactions)
Optional	`uri`	The HTTP URI of the queried IM (when `resource=im`)
	`jid`	The Jabber ID (JID) of the queried peer (when `resource=peer`)
	`iid`	The interaction ID of the queried interaction (when `resource=pi`)
	`q`	The string-based query phrase
	`format`	`xml` or `json` or `jsonld`

An example of OKBook federation is illustrated in Figure 3.12. The OKBook portal is a public portal from which peers willing to play the OKBook server role can download the software and also curates a list on which domains of all the OKBook servers are recorded. The federation component is included in the downloadable software and after installation, the hosting peer can choose to switch on the federation functionality or not. With this functionality being switched on, the software will send a request to the OKBook portal automatically and retrieve all the shared information about other OKBook servers, to achieve the federation. So it is up to OKBook servers themselves to decide if they want to share their knowledge about registered peers, their interactions and other valuable information. This configuration of federation is fine-grained at the registered peer level, which means the peer, holding an account on a specific OKBook server, can grant this server the privilege of sharing the information about the peer itself with other OKBook servers.

Summary

Inspired by the OpenKnowledge system, this chapter proposes an approach for forming and evolving peer communities based on the shared choreography specifications (IMs). According to this approach, a proof-of-concept system, named as OKBook, has been

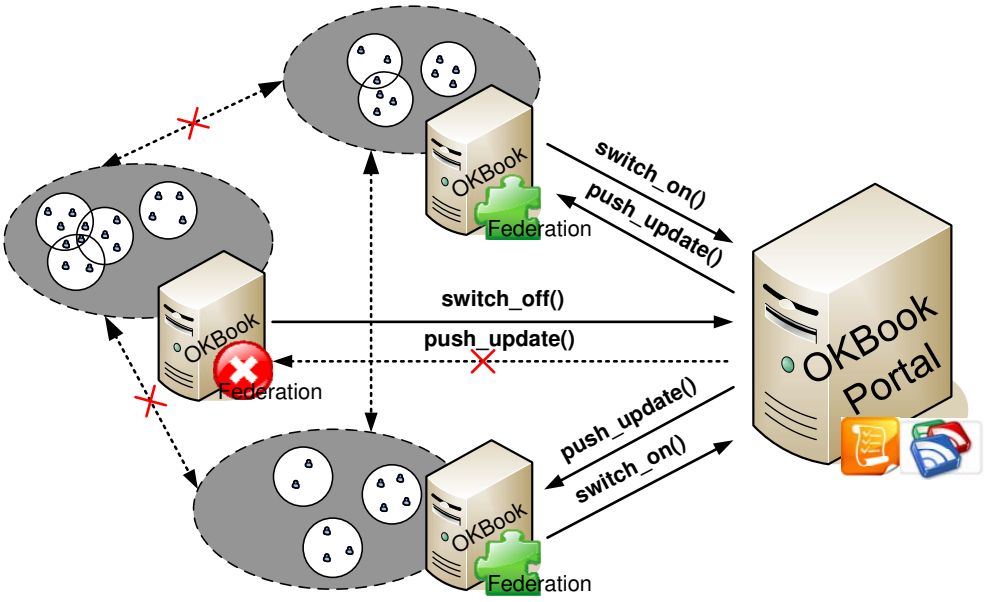


Figure 3.12: Example of the OKBook federation

implemented to assist peers in publishing, discovering and (un)subscribing to IMs. Two kinds of discovery services are currently provided via meta-search engine and peer grouping, respectively. This approach complies with principles of Linked Data so as to be capable of both contributing to and benefiting from the Web of data. Although message passing is not prescribed by LCC (Robertson, 2004), since each IM execution is a message-intensive process, this needs to be carefully handled to make our approach actually work with the current Web architecture in terms of design and implementation, and we explore this in the next Chapter.

Chapter 4

Interaction Model Execution on a P2P Communication Layer

Following Chapter 3, this chapter takes one step further the development of the OKBook system. Since the Interaction Model (IM) execution is a message-intensive process, how to deal with the message passing effectively and efficient in an open decentralised environment becomes a key issue which needs to be tackled. Focused on this issue, Section 4.1 gives a brief overview on the redesign of the communication layer of OpenKnowledge as well as describes the motivation behind that. Section 4.2 proposes a solution to message passing in a peer-to-peer manner and the involved protocols. Section 4.3 introduces a new operator which achieves the concurrency of IM execution without blocking any I/O process. Section 4.4 illustrates the overall platform architecture of OKBook as an interaction-driven peer-to-peer community.

4.1 OpenKnowledge Communication Layer Redesign

As mentioned earlier, the OpenKnowledge system has been designed to randomly select a peer as the coordinator which will run an IM. The coordinator is in charge of interpreting IM code and helping involved peers in passing messages to one another. This middle-man design can mitigate the burden on participants by handling most computation for that interaction on a single and central coordinator. However, this does not either make good use of resources available on peers themselves or comply with the spirit of the distributed computation. On the other hand, due to the autonomy of peers,

some of them may not want to be coordinators or share their limited computational resources to other peers without any payback. In this section, the OpenKnowledge system is redesigned based on a new architecture, in which peers can communicate without any assistance from coordinators.

4.1.1 Motivations

OpenKnowledge aims to provide peers with a system which allows them to interact with one another without any global agreements or knowledge about who else will participate in a particular interaction or how this interaction is executed (Robertson et al., 2009). However, several issues have not been addressed in the existing system. For instance, in the current OpenKnowledge system, peers have to interact with one another via coordinators which are selected by the OpenKnowledge kernel randomly and which behave like super peers when IMs are executed. This design has increased the burden on the limited bandwidth offered by each coordinating peers and some of them may refuse to contribute their bandwidth for delivering messages during interactions. In this case, the triggering of the interaction will fail and the kernel has to keep on searching until it finds another peer willing to play the coordinator role. Sometimes this may cause a significant delay on running an interaction.

According to the above analysis, OKeilidh has been implemented as a prototype of the redesigned architecture based on a cross-domain instant messaging protocol. OKeilidh is an extension of the communication layer of OpenKnowledge [1] and focused on choreography execution within the peer-to-peer environment.

4.1.2 Communication Layer Framework

OKeilidh provides a decentralised environment for peers to interact with one another on the Web. In order to implement this design and make it comply with the traditional Web architecture, a domain-crossing strategy needs to be applied. Extensible Messaging and Presence Protocol (XMPP) allows users to pass messages to others from different domains (servers). This protocol eliminates the limitation of the conventional server/client structure in which clients can send requests to servers but servers cannot

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

send requests to clients. In this chapter, OKeilidh has been built based on this protocol, although note that any instant messaging protocols which are capable of getting around widely spread Firewalls or Network Address Translations (NATs) can replace XMPP here and serve the same purpose. As a matter of fact, OKeilidh makes a super peer (server) delegated to each normal peer (client) and when a client is queried, its delegated server is identified and becomes its spokesman. Figure 4.1 depicts the framework for OKeilidh in which a number of OKBook peers from different domains are involved. *Alice* and *Bob* registered in the same sub-domain of the server *okbook1*. *Carol* also registered on the *okbook1* but in another sub-domain while *Daniel* registered on another server *okbook2*. Here, some of them are from different domains, but XMPP allows them to do the cross-domain interactions even if the Transmission Control Protocol (TCP) ports are blocked by firewalls. Under this circumstance, both the client and the server can listen to the normal Hypertext Transfer Protocol (HTTP) ports anyway. Besides an XMPP-awared client, each peer has been equipped with the LCC Interpreter (LCCI) which takes the IM and its execution state as input and tries to solve the constraints this peer has committed to satisfying and the output will be this IM itself plus its new execution state if the interaction has not yet been completed.

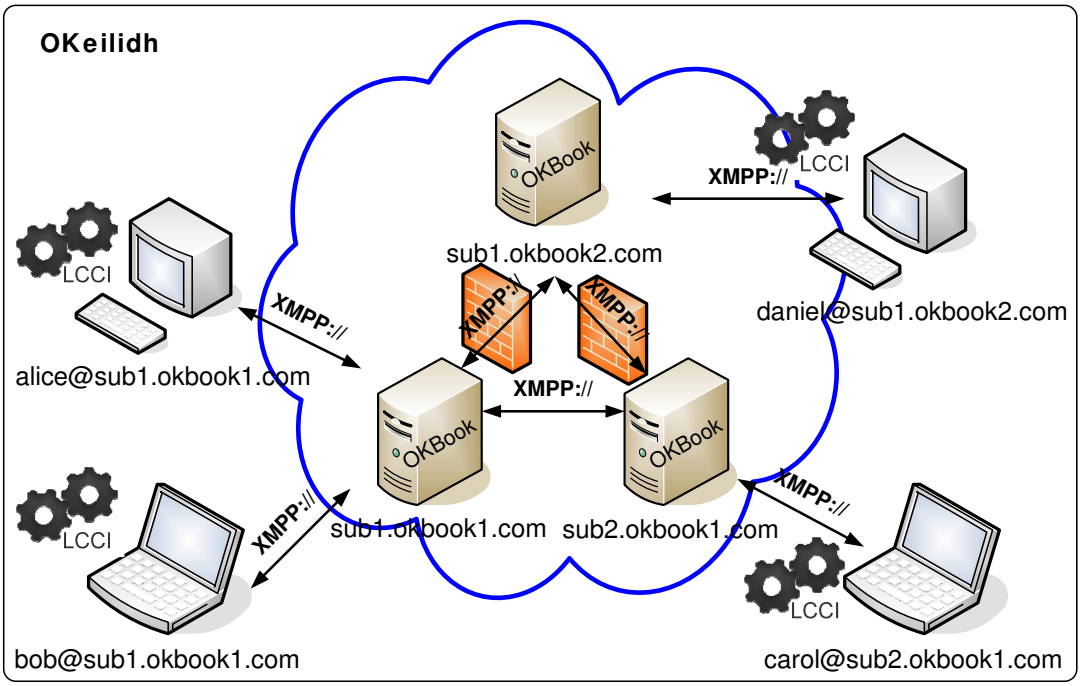


Figure 4.1: Communication Layer Framework

4.1.3 Peer Interaction Messaging Flows

Each message defined in a specific IM is *de facto* the information carrier and plays a key role within the peer interaction process. In this section, its schema is designed carefully for assisting peers in understanding the current state in which the execution of the IM is and what the message receiver should do in the next step, and at the same time maximising the compatibility with respect to a variety of software/hardware environments peers may live in.

4.1.3.1 Message Schema

In OKeilidh, each message is in XML and contains several fields such as *clause*, *IM_content*, *common_knowledge*, *from_role*, *from_jid*, *to_role*, *to_jid*, *last_msg*. These fields behave like metadata for each message and provide the LCCI with necessary information required for running part of an IM as well as generating the next message if necessary. A message handler will wrap these fields into a particular predicate or method, which will be taken as the initialisation before the Lightweight Coordination Calculus (LCC) is actually interpreted.

4.1.3.2 Message Parsing and Update

Here, we use Openfire [2] to equip super peers as discussed above with XMPP servers. Each peer installed with an OKBook server has been installed with an Openfire server as well, which will employ a different port to do the message passing. Therefore, peers which registered on a specific OKBook server will be assigned with a corresponding OKeilidh account. For instance, if a peer has an OKBook account with the name *an* on a server with the domain name *okbook.com*, this peer will also get an OKeilidh account with the name *an@okbook.com* (Jabber ID) by which this peer can interact with other peers via XMPP. Message parsing will be done in terms of the message schema defined above. LCCI has a method which can bind values of fields parsed from received messages to the internal variables used for initialising the OpenKnowledge Components (OKCs). There is a configuration file sitting on the peer-side terminal device (a PC or a mobile phone, etc) and it is used for informing the LCCI of the paths of OKCs installed on this peer.

[2] `http://www.igniterealtime.org/projects/openfire/`

A LCC clause is a role definition of an IM, whose constraints and message passing are supposed to be solved and performed respectively by the host peer. After the LCC clause extracted from the received message begins to be executed, the LCCI will inform the host peer of the value update of each argument and all updates are wrapped into a new message which will be passed to the receiver in terms of the peer's OKeilidh account thereafter. Taking the IM described in Figure 3.8 as an example, when shop S receives message $buy(PC,CC)$ from client C, the value of variable R (which denotes the receipt) is *undefined*. After constraint $complete\_order$ is solved at S's side, R will be assigned the actual ID of the receipt and sent back to C along with other arguments via message $receipt(R)$. Note that the LCCI can be implemented using various programming languages and this diversity can be handled by our LCC engine which provides a generic interface used for communicating with LCCIs such as the Prolog LCCI and the Java LCCI.

4.1.3.3 Message Passing Between Various Peers

LCC was developed and adopted in the OpenKnowledge project [3] for choreographing services provided by a large number of peers. It is a compact but expressive choreography description language and in this thesis it is also employed for writing IMs, although other choreography description languages could be used alternatively. Since human users can be taken as units which are capable of fulfilling tasks during interactions, they are also recognised as peers along with computing devices. Autonomous peers may play multiple roles simultaneously and as mentioned earlier, it is recommended that the discovery peers (also called community peers, on which IMs and collaborators are searched) are also installed with the communication components so as to become Discovery and Communication (D&C) peers for making better use of limited resources. Technically speaking, any peers can be installed with both a discovery service and a communication service (in charge of message passing) at their own will so as to be D&C peers. With respect to the hub peers (see in Section 3.1.2), on each of them there may be a hash table curated for storing the subscription information of peers. So we are basically discussing three types of servers here, including discovery servers, communication servers and hub servers, any of which could be installed on any peers.

[3] `http://www.openk.org`

In the peer community formed based on our approach, peers exchange messages using HTTP and Transmission Control Protocol/Internet Protocol (TCP/IP) essentially. Communication peers (e.g., equipped with XMPP (Saint-Andre, 2004) servers) also use these protocols to pull messages from or push messages to others. Since many of polls do not return new data based on the long polling mechanism, pushing is more efficient, but a real push also requires circumstances under which TCP is possible. Messages passed between different peers may be processed and delivered in different ways. A super peer is a discovery peer and/or communication peer which is always online, changes less frequently and can provide trustworthy services continuously. Peers that are not super peers are called normal peers. The ways of messaging between these different roles are summarised here. In our approach, the messages are passed between peers according to XMPP under the following circumstances:

a. *Messaging Between Normal Peers and Discovery Peers*. For users' activities (e.g., browsing an IM, querying a SPARQL Protocol and RDF Query Language (SPARQL) endpoint, signing in/out, etc.), the browser itself is the equivalent of a peer which delegates the user to send HTTP queries to another peer inside the community and renders the result sent back.

b. *Messaging Between Discovery Peers and Peers Playing the Initial Role*. As soon as all roles defined in an IM are filled up by certain peers, a discovery server will bootstrap the interaction by sending the message containing subscription information to the peer which is committed to playing the initial role.

c. *Messaging Between Discovery Servers*. When a peer cannot find an IM which is good enough to meet its requirement, the discovery server on which this peer has been logged on will forward an IM querying message (a SPARQL query wrapped in an HTTP request) to other adjacent discovery servers.

d. *Messaging Between Normal Peers During an Interaction*. During the running of an IM, involved peers talk to one another according to the protocol employed by the communication server. There is no coordinator (an intermediate peer) in charge of interpreting involved messages and each peer has its own message handler which is capable of parsing the incoming messages and sending out messages wrapping LCC clauses together with the variables (which may be bound to some values) to the receiver's local LCCI).

e. *Messaging Between Normal Peers and Communication Peers* Normal peers send

messages to and receive messages from communication servers according to the protocol these servers have been employing.

4.2 Peer-based IM Execution Design

Nowadays, widely used Web browsers (e.g., Internet Explorer, Firefox, Safari, Opera and Chrome, etc.) have become windows through which clients can interact with a variety of Web Services or even other clients. In this section, we introduce a framework for designing an LCC interpreter embedded in the browser which will render a user interface so as to provide a rudimentary and easy-to-access peer for users to use. A proof-of-concept implementation for the framework has been built and the demonstration is available in here[4].

Most Web browsers nowadays have built-in JavaScript interpreters and thus our Web version of the LCC interpreter has been also developed in JavaScript. Moreover, JavaScript incorporates functional programming capabilities and makes the event-based programming possible, which share common features with process languages like LCC and many of them are strait forward to be implemented in JavaScript. On the other hand, one of the differences between XMPP and HTTP is that HTTP is a stateless protocol but XMPP is stateful. This statefulness is an essential requirements for running IMs in LCC. We use Bidirectional-streams Over Synchronous HTTP (BOSH) [5] here for connecting HTTP requests with XMPP servers from inside browsers due to the unavailability of TCP connection there. Our IM Web Interpreter is discussed in detail below.

4.2.1 Bridge HTTP and XMPP

Since browsers normally have restrictions on supported protocols and have not widely supported XMPP, it is impossible for users to create direct connections from browsers to XMPP servers. XMPP requires a long term TCP connection between the client and the server, which cannot be realised via HTTP's very short term connection. Therefore, if users want to connect with XMPP servers from inside browsers using HTTP, there must be a proxy-like middleware to reconcile these two different protocols. BOSH is

[4] http://www.openk.org/okeilidh

[5] http://wiki.xmpp.org/web/Tech_pages/BOSH

a transport protocol that emulates a bidirectional stream between two entities (such as a client and a server) by adopting multiple synchronous HTTP request/response pairs without requiring the use of polling or asynchronous chunking (Paterson et al., 2010). In this specification, we use BOSH to forward HTTP requests to XMPP servers. There are currently two types of BOSH connections. Firstly, the XMPP server itself can expose a BOSH HTTP endpoint to outside but this endpoint can however just be used for connecting with one delegated server; Secondly, an increasing number of stand-alone BOSH proxies have been or are being implemented and they can be used for connecting any XMPP servers which published their Service Record (SRV). The latter type of the XMPP proxy needs to be powerful since there could be hundreds of thousands of clients try to connect with various XMPP servers through this proxy which in this case the proxy needs to curate a great number of long term TCP connections and cause large overhead on itself. Still, the stand-alone proxy solution is recommended because of its flexibility and possible higher quality than the former endpoint solution. OKeilidh harnesses public stand-alone bindings of HTTP and XMPP.

4.2.2 Overview of XLCC Grammar

The LALR(1) parser and lexical analyser has been used for creating our eXtended Lightweight Coordination Calculus (XLCC) Web interpreter (XLCC is an extended version of LCC and dedicated to the Web ecosystem on top of the Internet). Compared with LCC, XLCC allows peers to script IMs via JavaScript-aware user agents such as Web browsers as discussed in (Bai et al., 2012). The Backus-Naur Forms (BNFs) of LCC are not discussed here and have been detailed in (Robertson, 2004) and Table 4.1 describes the BNFs of XLCC.

The grammar overview of XLCC is described as in Table 4.2 (LHS stands for the left-hand side and RHS stands for the right-hand side). Note that since XLCC will be interpreted in JavaScript, JavaScript Object Notation (JSON), as a lightweight data-interchange format based on a subset of JavaScript, is supported and recognised as a default data model (e.g., the subscription information can be fed into the XLCC interpreter as a JSON object). The *state overview*, the *pop* table, the *action* table and the *goto* table related to XLCC parsing are listed in Appendix A.

Table 4.1: XLCC syntax

IM	:=	*Clause_List*|*head*(*Constant*). *Clause_List*
Clause_List	:=	*Clause*|*Clause_List*
Clause	:=	*Role* :: *Def*.|*Role*.|*plays*(*Constant*, *Constant*).|*knows*(*Constant*).|*iid*(*Constant*).
Role	:=	*a*(*Type*, *Id*)
Def	:=	*Message*|*Def then Def*|*Def or Def*|*Def niob Def*|{ *Def* }
Message	:=	*M* ⇒ *Role*|*M* ⇒ *Role* ← *C*|*M* ⇐ *Role*|*C* ← *M* ⇐ *Role*|*null* ← *C*|*Role*|*Role* ← *C*
C	:=	*Constant*|*Constant*()|*Constant*(*Terms*)|*C* && *C*|*C* || *C*|*list*(*Variable*, *Variable*, *Variable*)
		C = *C*|*C Operator C*|*not*(*C*)
Terms	:=	*Term*, *Terms*|*Term*
Type	:=	*Term*
Id	:=	*Constant*|*Variable*
M	:=	*Constant*(*Term*)
Term	:=	*Constant*|*Variable*|*Constant*(*Terms*)|_
Operator	:=	==|!=|>|<|>=|=<
Constant	:=	lower case character, sequence or number
Variable	:=	upper case character, sequence or number

4.2.2.1 Interpreter Input

Besides the LCC code, the LCC Web interpreter needs extra information to run an interaction. For instance, the interpreter needs to know which role defined inside a particular IM is associated with which peer. Since each involved peer also needs to know which OKC(s) provided by itself has been registered in the peer community before the interaction, this information about OKCs should also be fed into the interpreter to help peers "recall" these predefined OKCs on their servers. This kind of information can be serialised in any formats which are friendly for data exchange on the Web. We use JSON here due to its reputation for high processing speed and scalability (Crockford, 2006). Then the final data fed into the interpreter contains two sections: The header section is the subscription information serialised in JSON and the body section is the IM coded in LCC. An example input of our LCC Web interpreter is given in Figure 4.2.

Note that the interpreter identifies each interaction within the running of an IM by giving each of them an interaction ID. An interaction ID is a unique identifier generated from a particular peer community which triggered this interaction. Since there exists more than one community on the Web and peers may interact with those ones from other communities, if the interaction IDs were not defined carefully, some of

Table 4.2: XLCC grammar overview

Grammar Overview		
Left-hand side	FIRST-set	Right-hand side
IM′	**head plays knows iid a**	IM
Clause_List	**a**	Clause
		Clause Clause_List
BuiltIn_List	**plays knows iid**	BuiltIn
		BuiltIn BuiltIn_List
IM	**head plays knows iid a**	Clause_List BuiltIn_List
		BuiltIn_List Clause_List
		head (JSONLIST) . Clause_List
		Clause_List **head (JSONLIST)** .
		head (JSONLIST) . Clause_List BuiltIn_List
		head (JSONLIST) . BuiltIn_List Clause_List
Clause	**a**	Role **::** Def .
		Role .
Role	**a**	**a(** Type , Id **)**
Def	{ **null a Constant not Variable LIST String list**	Interaction
		Def **then** Def
		Def **or** Def
		Def **niob** Def
		{ Def }
BuiltIn	**plays knows iid**	**plays (Constant , Constant)** .
		knows (Constant) .
		iid (Constant) .
Type	**Constant Variable LIST String** _	Term
Id	**Constant Variable LIST String**	**Constant**
		Variable
		LIST
		String
Term	**Constant Variable LIST String** _	**Constant**
		Variable
		LIST
		Constant (Terms **)**
		String
		_
Interaction	**null a Constant not Variable LIST String list**	Message **=>** Role
		Message **=>** Role **<-** Constraint
		Message **<=** Role
		Constraint **<-** Message **<=** Role
		null<- Constraint
		Role
		Role **<-** Constraint
Message	**Constant**	**Constant(** Terms **)**
Constraint	**Constant not Variable LIST String list**	**Constant**
		Constant ( )
		Constant(Terms **)**
		not(Constraint **)**
		Id **==** Id
		Id **!=** Id
		Id **>** Id
		Id **<** Id
		Id **>=** Id
		Id **=<** Id
		Variable = Id
		Constraint **&&** Constraint
		Constraint **||** Constraint
		list (Id , Id , Id **)**
Terms	**ConstantVariableLISTString**_	Terms , Term
		Term

them could overlap. These overlapped IDs may cause issues within the processes of running IMs because they have the same identifier but represent different interactions, respectively. Therefore, we attach the above interaction ID (e.g., a unique timestamp plus the secure hash of the IM Uniform Resource Identifier (URI)) with the ID of the

```
head(
   [
      {
         jid :  "admin@okbook.inf.ed.ac.uk",
         role:  ",
         roleType :  "trigger",
         okcs :  [],
         iid :  "XYZ"
      }
      ,
      {
         jid :  "alice@okbook.inf.ed.ac.uk",
         role :  "client",
         roleType :  "initial",
         okcs :  ["http://okbook.inf.ed.ac.uk:8188/okbook/im/alice.js"]
      }
      ,
      {
         jid :  "bob@okbook.inf.ed.ac.uk",
         role :  "shop",
         roleType :  ",
         okcs :  ["http://okbook.inf.ed.ac.uk:8188/okbook/im/bob.js"]
      }
   ]
).

a(client(PC, CC), C)::
   buy(PC, CC)⇒ a(shop, S)← payby(CC) && lookup(S) then
   receipt(R)⇐ a(shop, S).

a(shop, S)::
   buy(PC, CC)⇐ a(client(_),C) then
   receipt(R)⇒ a(client(_), C)← enough_credit(CC, PC)&&
                                complete_order(PC, CC, R).

a(client("Harry Potter", "8686868686868686"), C).
```

Figure 4.2: Simple trade IM in XLCC with the header

community which triggered this interaction to form a unique ID for each interaction on the whole network, which is here called the cross-Community Interaction ID (XCI ID). Peer communities use XCI IDs to trace interactions triggered by themselves. On the other hand, a specific peer may interact with more than one group of peers simultaneously and a message routing mechanism is required to distinguish incoming and outgoing messages since this peer may receive messages belonging to different interac-

tion processes. According to RFC 3920 [6], a full Jabber ID (JID) may have a resource binding that indicates which resource has the higher priority than other resources in terms of handling and processing messages passed around. Here, in order to make peers differentiate messages coming from different interactions, we use the above XCI ID as the JID resource name, which will behave like a channel for each peer, with all the messages belonging to a particular interaction being filtered out based on this XCI ID and further processed by the interpreter interpreting the corresponding IM. But how do we deal with an extreme case in which an IM has more than one instance (interaction) triggered by the same community server, involving the same group of peers and starting at the same time? To address this problem, we further improve the XCI ID by adding a random factor to the original timestamp to generate a new randomised timestamp. All in all, In terms of the above analysis, an interaction can be identified by its XCI ID whose value will be influenced by the triggering community, the triggering time, the corresponding IM and a random factor over the triggering time.

As aforementioned, each peer has a LCCI running on itself and a single LCCI is supposed to be dedicated to the LCC clause(s) which defined the role(s) its owner peer is about to play. Therefore, the community server will hide other irrelevant LCC clauses (i.e., other role definitions) and segments of sensitive information (e.g., the OKC locations of other peers) for the sake of privacy before sending the triggering information to each peer that will be involved later on. During the execution of IMs, each involved peer's states will be maintained by its installed interpreter and these states (recognised as key-value pairs by the LCCI) could be also wrapped in messages being passed between peers. This has been discussed in (Robertson, 2004) as an alternative method for peer coordination. However, since the peer interaction is a message-intensive process which may cause considerable overhead on the network node with limited bandwidth or resources allocated on each of them. LCCIs do not allow peers to pass around their states (which may be sensitive or confidential) via messages.

4.2.2.2 Interpreter Output

The output of the LCC Web interpreter will be generated and sent back to the peer community which triggered a specific interaction after every interpreter on each involved peer is terminated. This termination can be caused when an interaction is finished

[6] http://xmpp.org/rfcs/rfc3920.html

successfully (there is no more XLCC code that needs to be executed for all involved peers) or when some exceptions/errors occur during the interaction. Since for each interaction, the involved peers will finish their jobs and leave the interaction separately. So in the body of each XMPP message, there needs to be an indicator to show how many involved peers had finished their interactions when the message is created in order to inform the peer community of the progress on peer interactions. Based on the interpreter input described in Section 4.2.2.1, each peer can know how many peers are involved in a particular interaction by parsing out the JSON header of the corresponding IM forwarded to itself beforehand. The value of the indicator will be increased as the interaction progresses. If the number of peers is equal to the value of the indicator plus one, the interaction is finished and there will not be any message passing afterwards. Peers need to know when they should increase the value of the indicator and this requires locally-installed LCCIs to look ahead to the rest of their role definitions before sending out any messages. In other words, each interpreter needs to pend the sending of a message and check if the peer will be waiting for a new incoming message or will send another message out in the near future. If not, the value of its indicator will be consequently increased by one, and otherwise, the value stays the same as before. Figure 4.3 depicts how this method works using an dummy IM in which messages are passed in a circle. From this IM, we can see that *Bob* will first finish his interaction job and *MSG2*, containing the indicator about this accomplishment, will be sent to *Carol*. Then *Carol* finishes her interaction job and the value of this indicator will be changed and sent to *Alice* who is the last peer that finishes its interaction job. After that, Alice's LCC interpreter will send a message informing the original interaction trigger *sub1.okbook1.com* of the accomplishment of the whole interaction. Note that the arrow lines with dots denote that the messages is not passing directly but via XMPP servers.

The advantage of this method is that there will be only one message sent from the last peer which is on completion of its interaction duty and before that the communication server does not have to record any accomplishment information from any other involved peers. However, since there is only one shared indicator, some message passings may need to be pending, which could block both the interaction to which the indicator is associated and other interactions possibly running at the same time. In this case, all other potential message recipients cannot get messages from the same message sender. There is another issue for this method when there is no single "last

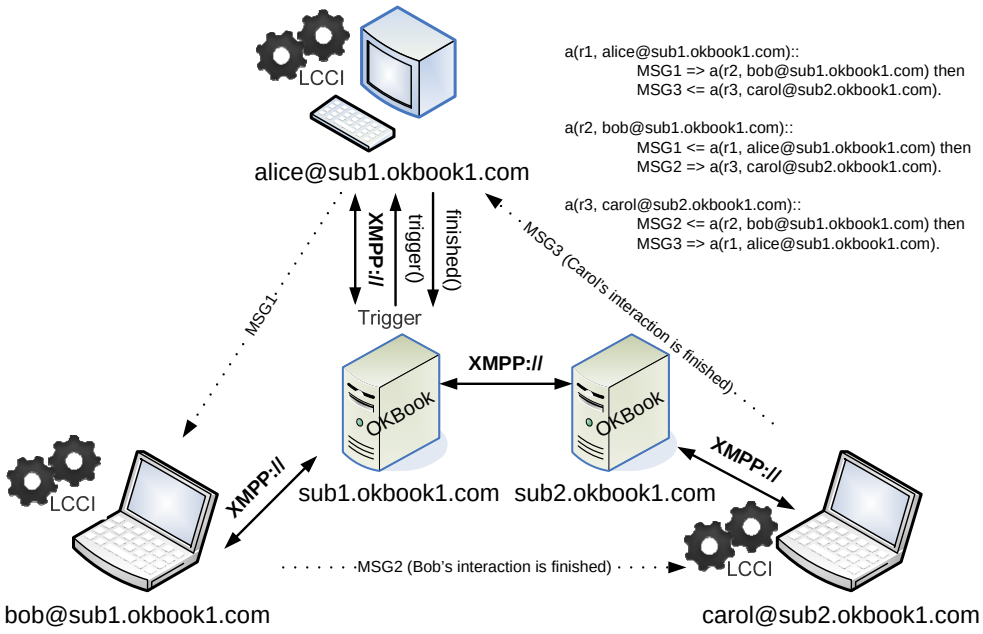


Figure 4.3: Interaction accomplishment indicators are wrapped in messages

recipient" for the running of a specific IM. Figure 4.4 illustrates this issue by giving an interaction with more than one last recipient. We can see that *Alice* sends a message to *Bob* and *Carol* respectively and there is no interaction between *Bob* and *Carol*. So within this interaction, *Bob* and *Carol* both think themselves as the last recipient and unsurprisingly, two finishing signals will confuse the trigger peer. Thus, another method needs to be designed here to address the above issue.

As mentioned earlier, based on our peer community design, it is recommended that each community server also has an cross-domain instant-messaging service installed so we can let involved peers ping the community server which triggered the interaction when each peer finishes its interactive task. In this case, the community server needs to maintain an interaction accomplishment list and every time when it gets a ping from a involved peer it will update the state of this peer in the list and check if all the peers have done their jobs. This maintenance will not cause extra burden on community servers since each of them has already got a list of peers' subscriptions and the accomplishment information can be just attached to that list as an extra property for each participant. This however requires that, in the header section of every message, there needs to be a JSON object which contains the information about the trigger such as the Uniform Resource Locator (URL) or JID (or both) for identifying and pinging the community server. Figure 4.5 depicts how this method works on the same IM already described in Figure 4.3. *Bob* is the peer who first sent his interaction accomplishment

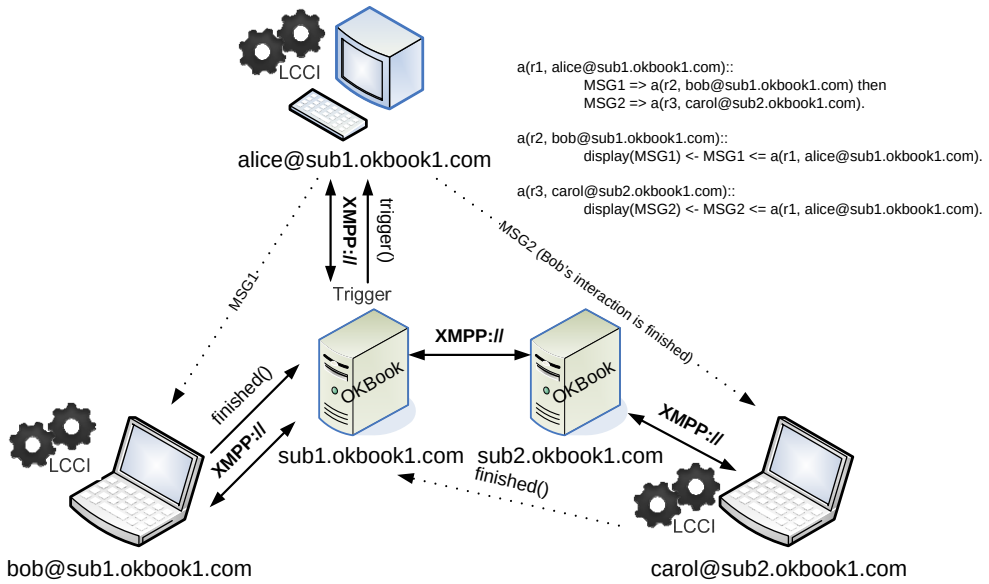


Figure 4.4: Indicator relay for the two-last-recipient case

signal to the trigger *sub1.okbook1.com* and *Carol* is the second. After the trigger gets the signal from *Alice*, it will know the whole interaction is accomplished. Thanks to the maintained list on each server, this method can therefore distinguish accomplishment signals coming from different participants and will not have the issue depicted in Figure 4.4.

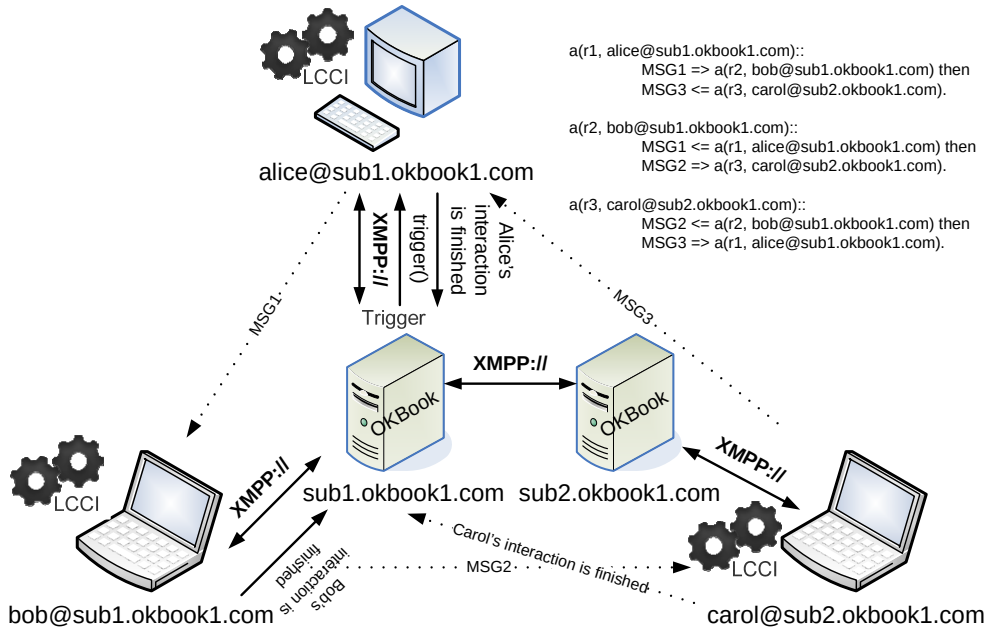


Figure 4.5: Interaction accomplishment signals are sent to the trigger

4.2.3 Security

Security in the peer-to-peer community is provided by message encryption, platform authentication and authorisation. As discussed in Section 4.1, XMPP has been employed as the message passing protocol and based on its specifications (Saint-Andre, 2004), the robust security which is achieved via Simple Authentication and Security Layer (SASL) [7] and Transport Layer Security (TLS) [8] has been built into the core. Alternatively, other appropriate certification authority, digital certificate or encryption technique may be used here alternatively to solve possible security issues. However, discussion of this is out of the scope of this thesis.

4.3 Event-Driven Concurrent Interpretation of IMs

Each IM is a work-flow description of an interaction between peers and their obligations (reflected by constraints). Actually, every IM models a process driven by diverse events including constraint solving and message passing, etc. Some events may trigger other events that influence activities operating in parallel. Multi-threads (or processes) and event-driven single thread are two typical ways of enabling this sort of concurrent programming, and both of them have pros and cons. The former way is straightforward to implement and has been boosted with the development of multi-core processors but suffers from overhead of configuration, initialisation and memory consumption. The latter way is more complicated to implement compared to multi-threads (or processes) but is more lightweight and scales to environments with massive computation units such as peer-to-peer networks. Traditional event-driven programming requires the thread to be blocked when an I/O event occurs during the execution progressing. The I/O operation will occupy the whole thread so other operations, which do not require the I/O operation to be finished and could run simultaneously, will be unnecessarily blocked and wait until that I/O blocking is unblocked. Under this circumstance, as time goes by, it is not surprising that considerable computing resources will be wasted due to some unnecessary I/O blocks. Message passing is a key operation within the process of running IMs and demands careful design to tackle the potential scalability issue but so far little attention has been paid to the optimisation. In this section, by

[7] http://tools.ietf.org/html/rfc4422
[8] http://tools.ietf.org/html/rfc5246

taking advantage of both thread-driven programming and event-driven programming, we propose an efficient way of interpreting IMs, which also incorporates non-blocking I/O mechanism.

On that basis, here we employ the Event-based Asynchronous Pattern (EAP), one of concurrency design patterns, to interpret IMs coded in LCC.

4.3.1 IM Events

Based on the design of LCC, so far four types of interaction-related events have been extracted as follows (Within the process of running IMs, these events will be curated in a so-called event list and functions attached to them will be executed after the events are emitted.):

1. **OKC Loaded Events.** This kind of events will be registered in the event list as soon as an IM starts running and will be emitted when the OKCs provided by an involved peer are all loaded into its runtime environment (i.e., a browser). Multiple OKC files may be harnessed and in order to improve the loading efficiency, peers are allowed to load them asynchronously. This requires peers to possess knowledge about the how many OKCs are going to be used and also maintain a counter to check if all the OKCs needed have been loaded. Each loading will trigger a callback function in which the value of the OKC counter will increase by one.

2. **Message Arrival Events.** This kind of events will be registered in the event list as soon as the LCC interpreter encounters a message receiving definition during the parsing of the IM and will be emitted when an external message actually arrives. As already defined in Table 4.1, a message-receiving definition in LCC is in the form of either `Constraints` ← `Message` ⇐ `Peer` or `Message` ⇐ `Peer`.

3. **Next Then Events.** This kind of events will be registered in the event list as soon as the LCC interpreter encounters the operator `then` during the parsing of the IM code and will be emitted when the left-hand-side definition (of `then`) is successfully executed (`true` is returned). If not successful, the whole interaction will be terminated and an exception will be thrown and later bought by a specific handler.

4. **Next Or Events.** This kind of events will be registered in the event list as soon as the LCC interpreter encounters the operator `or` during the parsing of the IM code and will be emitted when the left-hand-side definition (of `or`) is not successfully executed(`false` is returned). if successful, this event will be removed from the event list and never be emitted in the rest interaction.

Modern Web browsers are designed based on the Publish/Subscribe which is by nature one of models for event-based programming and this also influenced us to materialise our event-driven LCC interpreting in browsers.

4.3.2 Non-Blocking Messaging

As aforementioned, significant overhead during the IM execution will be caused by message passing and a message-intensive system such as the one capable of running IMs requires a non-blocking I/O architecture to mitigate the efficiency problem. Traditional virtual machines have been designed to support multiple processes/threads so in this case, each IM execution will make a virtual machine either fork a new process or spawn a new thread to handle itself. In each thread, if there is an I/O event, the whole thread will be blocked so other actions following this will be suspended and tremendous time will be wasted on waiting for the I/O to finish or in even worse case, the I/O action will not finish properly so the waiting will be infinite until a preset time-out is reached. Modern browsers normally are single-threaded (Google Chrome has one thread for each tab) and support non-blocking I/O natively. Our LCC interpreter has been designed to run in the browsers and thanks to its non-blocking I/O design, considerable resources and time will be saved during the running of IMs, as indicated by the experimental results in Section 7.3.

4.3.3 Design of the `niob` Operator

LCC originally had the operator `par` to connect definitions that will be expanded in parallel and this requires the virtual machine to fork two processes or spawn two threads in a single process to perform these parallel tasks. As discussed above, this type of concurrency programming is expensive and computational resources could inevitably wasted, especially when lots of I/O blockings occur. Event-driven programming harnessing I/O blocking can provide a relatively lightweight way to run a lightweight

language like LCC but since message passing play a key role in IM executions, the I/O blocks during peers' interactions will bring considerable delays in the whole execution process. Therefore, we propose a novel operator `niob` here to make XLCC leverage the event-driven interpretation with non-blocking I/O, which can improve the efficiency on the message-intensive peer interaction.

Besides `par`, there are currently two operators to determine the message sequence in LCC clauses: `then` and `or`. The former describes the left-hand side *def* denoted by S_1 will be completed first and after that, the right-hand side *def* denoted by S_2 will be completed. The later describes either S_1 or S_2 will be completed while S_1 will be attempted first. Now with the new operator `niob`, we have the following interpretation (in Javascript-like pseudo code) for three operators supported in XLCC:

Table 4.3: Interpretation for sequence operators in XLCC

Operators	Interpretations

S_1 `then` S_2	`execute(`S_1`, function(err) {`
	`  if (!err) execute(`S_2`);`
	`});`

S_1 `or` S_2	`execute(`S_1`, function(err) {`
	`  if (err) execute(`S_2`);`
	`});`

S_1 `niob` S_2	`execute(`S_1`); execute(`S_2`);`

In Table 4.3, callback functions are used for assuring the strict execution sequences for `then` and `or` in which S_1 will be completed first and S_2 will be attempted in the callback body with the `err` parameter which indicates whether the completion of S_1 is successful or not. However, in the `niob` sequence, there is no callback function so in this case, if there is a message passing that occurs in S_1, the execution of S_2 will not be blocked if it does not share any variables updated in S_1. Note that the XLCC interpreter employs lazy evaluation to interpret the XLCC codes.

4.3.4 Handling Multiple `niob` Operators

As of now, we have introduced a new operator to materialise the non-blocking I/O within the processes of running IMs which involve one or more messages. The ex-

ecution of a specific IM which does not have message passing will not benefit from applying `niob` operators so suppose S_1 and S_2 are two *def*s without message passing and we have the following formula showing that in this case `niob` can be replaced with `then`.

$$S_1 \texttt{ niob } S_2 \quad \Leftrightarrow \quad S_1 \texttt{ then } S_2 \quad (if\ neither\ S_1\ nor\ S_2\ has \Leftarrow or \Rightarrow) \tag{4.1}$$

In order to interpret IMs with more than one `niob` operator, the LCCI needs to take more care about the relations between them. Assume there are n `niobs` ($\{n|n \geq 2, n \in N\}$) appear in a single role definition which is consequently split into $n+1$ segments named as *niob contexts* as shown as follows:

$$\begin{array}{l} a(role,X) :: \\ \qquad \left.\begin{array}{l} ThenOrDef \\ \ldots \\ ThenOrDef \end{array}\right\} \texttt{context}_1 \\ \qquad niob\texttt{---------niob}_1 \\ \qquad \left.\begin{array}{l} ThenOrDef \\ \ldots \\ ThenOrDef \end{array}\right\} \texttt{context}_2 \\ \qquad niob\texttt{---------niob}_2 \\ \qquad\qquad \ldots \\ \qquad niob\texttt{---------niob}_n \\ \qquad \left.\begin{array}{l} ThenOrDef \\ \ldots \\ ThenOrDef. \end{array}\right\} \texttt{context}_{n+1} \end{array}$$

During the running of an IM, the `defs` belonging to a particular context will be executed in sequence (blocking I/O) and the `defs` from different context may be however executed in an arbitrary sequence that depends on the progress in each context. Each IM has a default `niob` context even if it does not employ any `niob` operator so all `defs` derived from this default context will be also executed in sequence. In the LCCI, each context will be maintained with a specific identifier by which the rest `defs` will be resumed after the main thread comes back to that context.

4.3.5 XLCC Semantics

The semantics of XLCC mainly inherits the operational semantics originally defined in LCC (see in (Robertson, 2004)). Also, XLCC has extended LCC by bringing in a new operator and several built-in predicates. As mentioned in (Robertson, 2004), LCC does not prescribe the means of transmitting messages, but since XLCC has been designed as a browser-focused service choreography scripting language in this thesis, the semantics behind message passing in a peer-to-peer manner needs to be grounded.

4.3.5.1 Messaging

In XLCC, $\Rightarrow$ and $\Leftarrow$ denote sending a message to and receiving a message from another peer respectively. In order to achieve peer-to-peer message passing, any cross-domain messaging protocol could be used here for serving this purpose. By "cross-domain", we mean any messaging client is able to fulfil incoming connections in either a physical manner or a logical manner.

4.3.5.2 Concurrency

XLCC has not employed the `par` operator originally designed in LCC and instead invents the `niob` operator which is inspired by non-blocking I/O to achieve the concurrent computing. The `niob` operator is a binary operator and differentiate itself from the `then` operator by removing the unnecessary I/O blocking when the left-hand-side `def` is evaluated. As soon as a message reading is encountered, the XLCC interpreter will create a callback function and wrap all the remainder of the left-hand-side, which has not been evaluated, into this function, and after that, the interpreter will begin to evaluate the right-hand-side `def` of `niob`. During this evaluation, if the message passing (which occurred when the left-hand-side sub-clause was evaluated) finishes, a callback signal will be sent back to the interpreter from the listening socket and the corresponding callback function will be retrieved and evaluated immediately afterwards. After this moment, the evaluation of the right-hand-side sub-clause may or may not have finished. If not, the remainder of the `def` will be appended to a queue maintained internally by the XLCC interpreter and as soon as the evaluation of the left-hand-side `def` finishes or another message passing occurs, the interpreter will grab that remainder of the queue and resume the evaluation on it. On the other hand, if the left-hand-side

of `niob` does not involve any message passing, the evaluation will proceed exactly the same when the `then` operator would be applied. Under this circumstance, the `niob` operator can be substituted by the `then` operator, as already described in Equation 4.1.

4.3.5.3 Built-In-Predicates

At time of writing this thesis, XLCC is still evolving and has following built-in predicates which will handle diverse supportive information required by the IM execution.

a. `plays`: This predicate defines which peer will play which role during the IM execution. It has two parameters, the first of which denotes the peer's ID and the second one denotes the role name. This predicate does not need to support the multi-role selection since if a peer intends to play multiple roles defined in the same IM, role changing would be applied here to serve the same purpose. Hence, a peer just needs to select the entry role to play using this predicate.

b. `knows`: This predicate defines which OKC(s) the current logged-in peer will provide. It has a single parameter which denotes the URL of the OKC document. Multiple OKC URLs can be specified by repetitively employing this predicate.

c. `iid`: This predicate defines the universal ID (a.k.a., XCIID describe din Section 4.2.2.1) of an interaction which denotes a one-time execution of a specific IM. It has a single parameter which denotes the XCIID. The value of this ID can be generated by a community peer (Bai et al., 2010) which is about to trigger this interaction.

d. `list`: This predicate mimics the list operations in Prolog which inspired the original design of LCC. It has three parameters, the first of which denotes the whole list while the second and the third ones denote the head and the tail of this list respectively.

4.4 Overall Platform Architecture

So far, we have a redesigned communication layer supporting peer-to-peer message passing and the way of integrating this layer into OKBook discussed in Chapter 3, and constituting a fully functional knowledge sharing ecosystem is actually pretty much

straightforward and will be discussed in this section. As aforementioned, each OKBook peer will persist information about peers' subscriptions till the actual interaction is triggered. During this persisting process, each new subscription will make the OKBook peer check if all the roles in the target IM have been filled with at least one peer. If they are, an IM execution will come into the launching stage and otherwise, the execution will be pending for the time being and conducted later when all the roles are filled. For the role(s) which has(have) multiple peers attempting to play, through OKBook other role players will have a chance to vote for their favourite peer(s) they want to interact with. It can be a basic five-star voting and peers which are waiting to be voted will have initial ranks on the basis of their historical performances. After synthesising the votes, the OKBook peer will select peers, either with highest votes or with no competitor for a particular role, as a collaborative team for an interaction (peers with the same number of votes will be randomly selected). Thereafter, the OKBook peer will assign each team member an interaction-triggering URL with the subscription information as the URL parameters. Every time a peer is logged in, it can access its pending-interaction page displaying those interaction-triggering URLs (behind each of which there is an interaction waiting to be triggered). When the peer clicks on a specific URL, a service on OKBook will be called with those attached URL parameters based on which the OKBook peer will automatically generate and embed the required XLCC code into a Web document. Finally, this kind of documents will be reduced into the peer-side browsers which will commence interacting with other team members on behalf of the current logged in peers. Figure 4.6 describes a sequence diagram about choreographing Web Service (WS) from the perspective of a single peer and the OKBook server on which it is logged.

According to previous discussions, the peer side layered architecture and the OKBook side layered architecture are summarised and depicted in Figure 4.7 and Figure 4.8, respectively. The former contains two layers in Figure 4.7: the representation layer and the communication layer, both of which are also contained by the OKBook side architecture but on OKBook, the representation layer does not provide the profile annotator and moreover, the communication layer is not installed with any LCCI. Besides the above two layers, the OKBook side architecture contains an acquisition layer and a discovery layer in between, and as shown in Figure 4.8, both the representation layer and the acquisition layer need the PuSH implementation.

It is worth noticing that when having one or more pending interactions displayed on

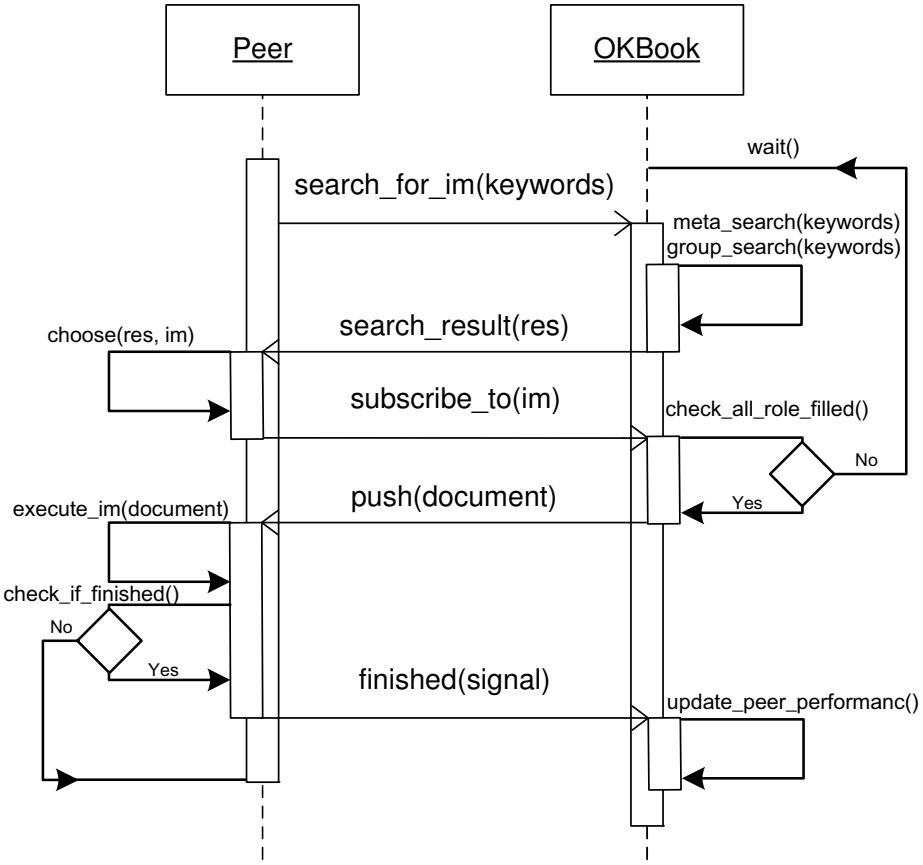


Figure 4.6: Choreographing WS from a single peer's perspective

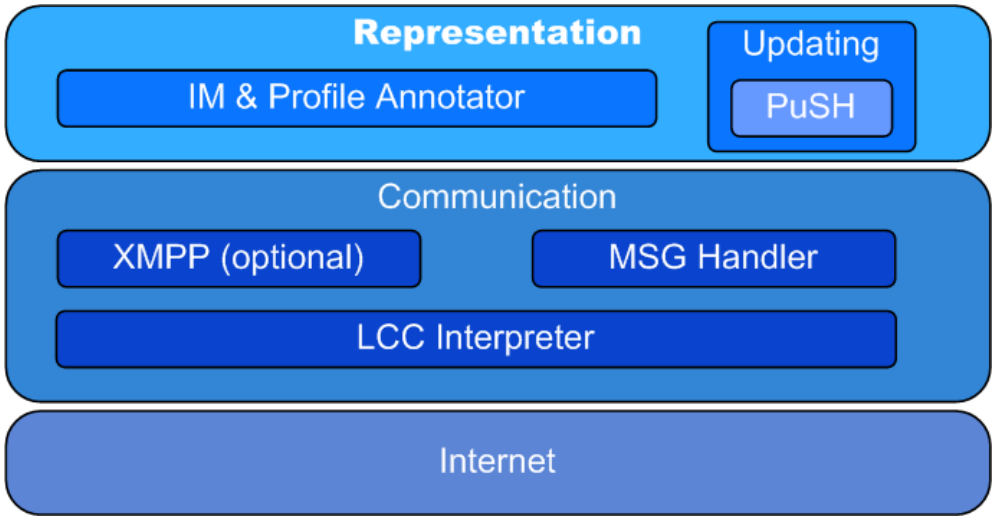


Figure 4.7: Layered architecture of a peer

OKBook, a peer will be provided with two environments to actually run the IM(s) behind those interactions, as described in Figure 4.9. One of environments is named as the*executing environment*, in which all the peer needs to do is clicking a start button

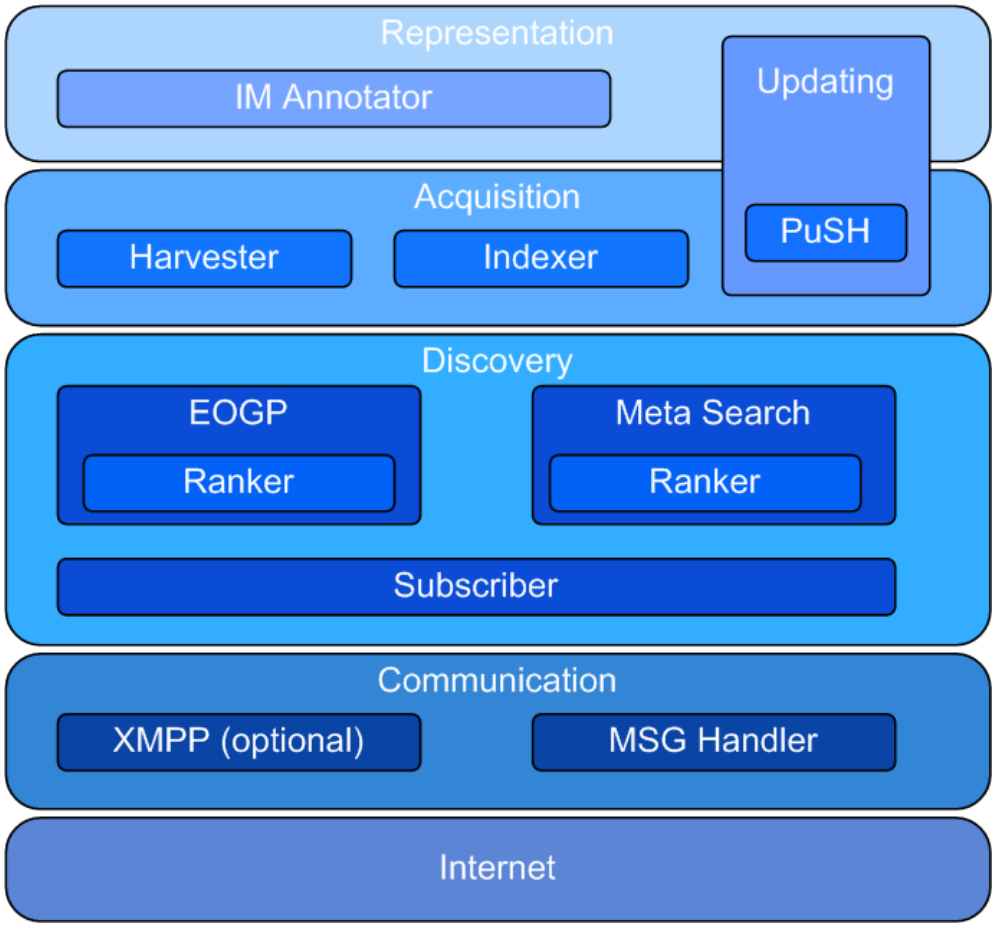


Figure 4.8: Layered architecture of OKBook

to fire an interaction or use the annotator to semantically enhance the current IM and republish it. The other environment is named as the *debugging environment*, in which the peer side browser will be equipped with more functionalities provided by OKeilidh in order to enable peers with professionals on both XLCC and JavaScript to debug and improve the current IM, configure the BOSH endpoint, and also import or generate OKCs on the fly if needed. Screenshots on IM editing and other supplementary functionalities in the OKeilidh *debugging environment* can be found in Figure 4.10 and Figure 4.11, respectively.

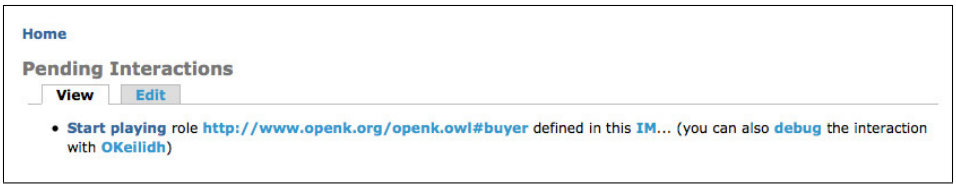


Figure 4.9: Screenshot on pending interactions

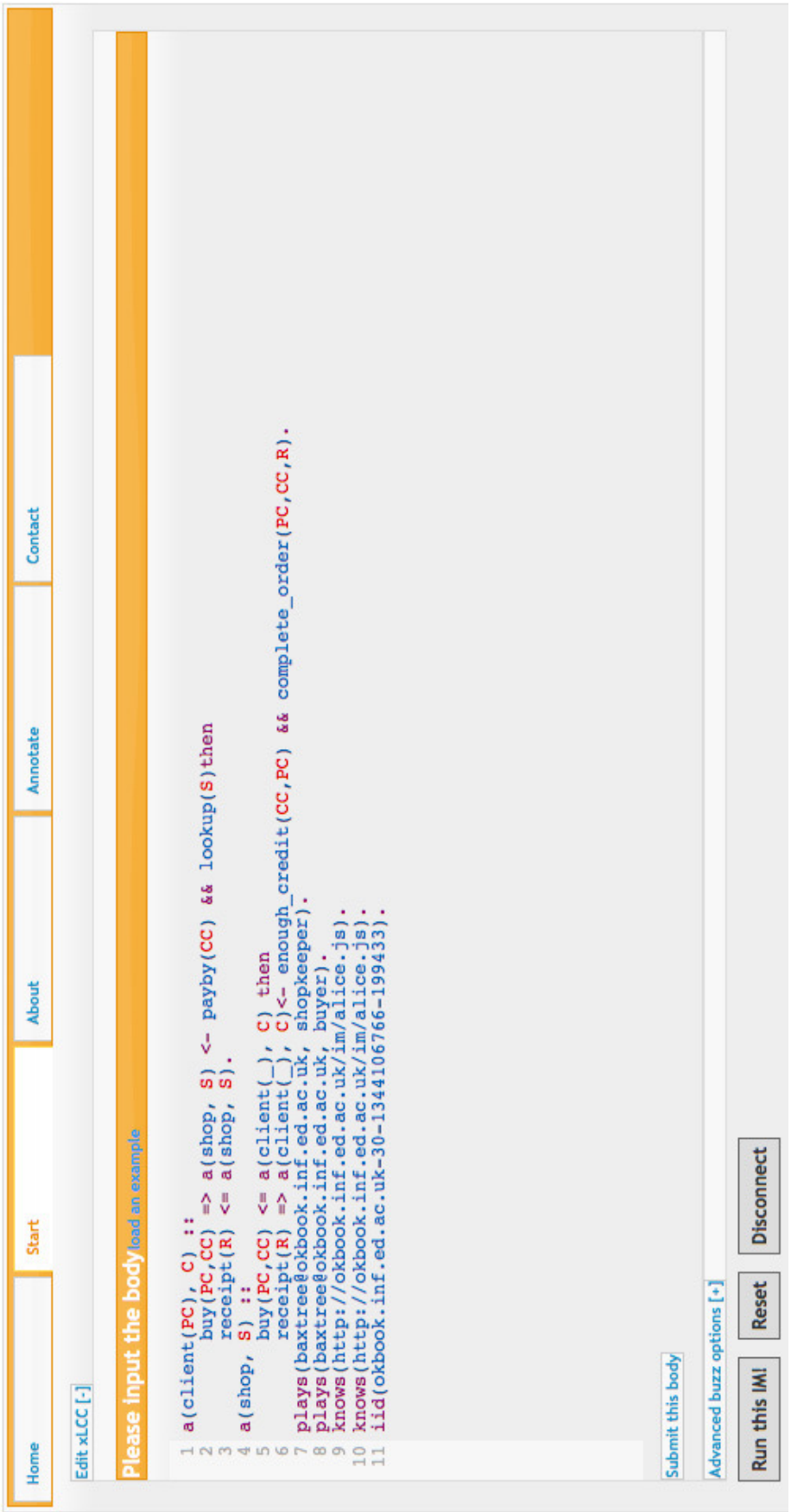


Figure 4.10: Screenshot on editing an IM in OKeilidh

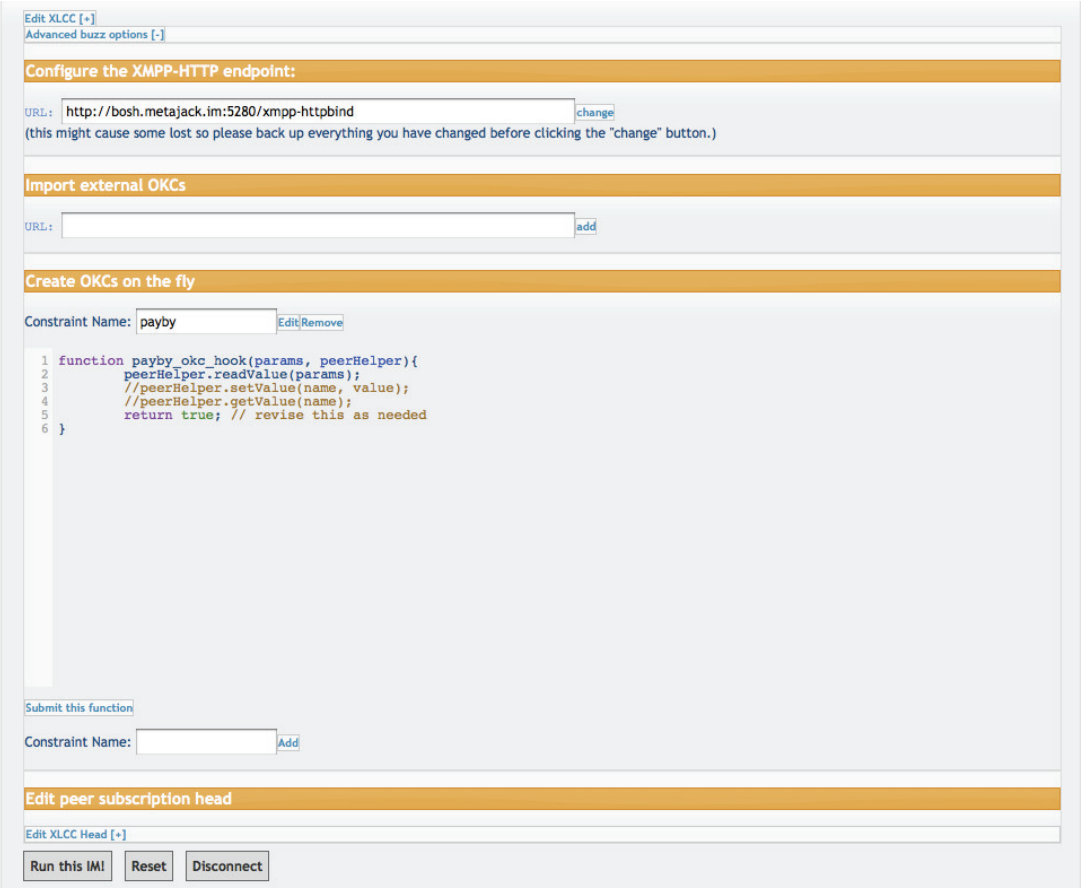


Figure 4.11: Screenshot on creating OKCs with a template snippet

Summary

In this chapter, the underlying message passing mechanism employed by OKBook is described as the redesign of the communication layer of OpenKnowledge. Even though LCC does not prescribe that mechanism, in order to function in the Web environment, the LCCI needs to ground semantics behind the mechanism in one way or another and OKBook chose an effective and efficient way to synthesise Web-based peer-to-peer communication protocols and a concurrent implementation which does not allow blocking I/O within the progress on interactions. In the next chapter, we will look further into the messages themselves to see how annotations can semantically enhance IMs and benefit peer interactions, and how to (semi)automatically generate annotations.

Chapter 5

Interaction Models as Web Documents

Several solutions exist for semantically describing Web Services (WSs) from the perspective of orchestration but little is known about how semantics benefit WS choreography. The most extreme challenge encountered by a choreography task occurs in peer-to-peer systems where shared semantics of data may need to be established via service interactions. In this chapter, we present a solution to this problem by sharing metadata via semantically enhanced Interaction Models (IMs) and since no pre-unified ontology is required in our approach, peers can make use of existing heterogeneous resources having been described in the Resource Description Framework (RDF) data model flexibly and compatibly. The experimental results in Chap 7 indicate that our approach can semantically enhance WS choreography (described in Lightweight Coordination Calculus (LCC)) in a lightweight way which complies with principles of Linked Data and published IMs with suitable semantic annotations can further facilitate the development related to other available linked WSs in one way or another (e.g., the formation of peer communities generated through peers' interactions). The remainder of this chapter is described as follows. Section 5.1 gives the motivations behind annotating IM documents. Section 5.2 designs an interaction-dedicated vocabulary which underpins the whole annotation process. Section 5.3 details the mechanism behind the generation of annotation-embedded documents and how these annotations will be consumed. Section 5.4 introduces an auxiliary tool, which can help publishers to generate IMs semiautomatically based on existing RDF triples, and describes its internal architecture.

5.1 Motivations

In a peer-to-peer network, peers are equal to each other from the perspective of autonomy and each of them has both server and client capabilities as mentioned earlier. From the perspective of choreography, peers collaborate through interactions in this thesis and LCC has been used for describing the choreographies inside the peer-to-peer network. Although our approach employs LCC here, the specific choice of the process language is not essential to the core arguments of this chapter, which means other choreography languages could be used for the same purpose but, since LCC has been carefully designed as a neat and lightweight languages based on the *process calculus*, we recommend it to peers who expect to avoid possible overheads caused by the understanding of the execution of choreography description languages. The syntax of LCC is described in (Robertson, 2004) and this thesis employs eXtended Lightweight Coordination Calculus (XLCC) as its extended version, whose syntax has been described in Table 4.1.

IMs work as protocols which direct involved peers to interact with one another and LCC has first-class processes, which means it treats processes as first-class objects. A first-class object is an entity that can be constructed at run-time, passed as a parameter to a subroutine, returned as the result of a subroutine, assigned in variables and data structures or have an intrinsic identity (Scott, 2009). In order to choreograph peers's services, we have this choreography description language now but how do peers share the meanings behind these IMs in the peer-to-peer environment? Figure 5.1 describes a journey-planning IM in XLCC involving six roles including one role-change.

The concepts employed in LCC are lightweight and message passing occurring under particular satisfiable conditions is a key which is easy for IM publishers to understand and define. However, LCC's lightweight feature also has its downsides due to there being no explanation about any of the elements (e.g., roles, messages and constraints) inside IMs so it is difficult if not impossible for a peer to easily recognise whether an IM is exactly the one he/she really need. Unfortunately, the vocabulary employed by the IM in Figure 5.1 is not fully machine-readable, nor is it easy for humans to understand. For instance, we cannot understand what *CC* denotes unless the original publisher has added free text comments as shown in between /$*$ and $*$/ in Figure 5.1. The IM publisher could use more self-descriptive names for arguments, like *CreditCard* instead of *CC* but this does not help from the perspective of machines unless there is

/∗ First, a traveller T sends to a travel agent *TA* the times and locations of her departure and arrival. Second, *TA* normalises the query and sends it to a CRS (Carrier Routing System) C which will generate routes from the journey start and endpoint information. *TA* also sends the journey query to an evaluation unit E in order to constantly get latest statistics on travellers' queries. Third, C sends each generated route to a GDS (Global Distribution System) G to obtain costs for each route and then sends journey information back to *TA*, which will reprice each journey and also generate final options for T. After receiving a message with journey options from *TA*, T makes a choice and notifies a TMC (Travel Management Company) *TM* for booking by her credit card. Finally, *TM* sends the ticket and the receipt back to the interaction initiator T. ∗/

```
a(traveller, T)::
 search(Departure, Arrival, DepTime, ArrTime) ⇒ a(tavelAgent, TA) then
 display(Options) ← options(Options) ⇐ a(travelAgent, TA) then
 book(JourneyID, CC) ⇒ a(tmc, TM) ← chooseJourney(Options, JourneyID) && payby(CC) then
 booked(Tickets, Receipts) ⇐ a(tmc, TM).

a(travelAgent, TA)::
 search(Departure, Arrival, DepTime, ArrTime) ⇐ a(traveller, T) then
 journeyQuery(Query) ⇒ a(crs, C) ← normalise(Departure, Arrival, DepTime, ArrTime, Query) then
 recommend(Journeys) ⇐ a(crs(Routes, Journeys), C) then
 options(Options) ⇒ a(traveller, T) ← repricing(Journeys, Options) [then] -----> [niob]
 record(Query) ⇒ a(evaluator, E) then
 evaluation(Statistics) ⇐ a(evaluator, E).

a(crs, C)::
 journeyQuery(Query) ⇐ a(travelAgent, TA) then
 a(crs(Routes, []), C) ← findRoutes(Query, Routes).

a(crs(Routes, Journeys), C)::
 recommend(Journeys) ⇒ a(travelAgent, TA) ← Routes == [] or
 {
  priceQuery(Route) ⇒ a(gds, G) ← list(Routes, Route, RestRoutes) then
  price(Route, Price) ⇐ a(gds, G) then
  a(crs(RestRoutes, Builtup), C) ←list(Builtup, [Route, Price], Journeys)
 }.

a(gds, G)::
 priceQuery(Route) ⇐ a(crs(Routes, Journeys), C) then
 price(Route, Price) ⇒ a(crs(Routes, Journeys), C) ← calculatePrice(Route, Price).

a(tmc, TM)::
 book(JourneyID, CC) ⇐ a(traveller, T) then
 booked(Tickets, Receipts) ⇒ a(traveller, T) ← charge(JourneyID, CC, Receipt) && print(Ticket).

a(evaluator, E)::
 record(Query) ⇐ a(travelAgent, TA) then
 evaluation(Statistics) ⇒ a(travelAgent, TA) ← evaluate(Query, Statistics).
```

Figure 5.1: Basic Travel Planning IM in XLCC

accompanying ontology which provides a semantics for that more readable label, such as `http://dbpedia.org/resource/Credit_card`. On the other hand, IMs without semantic enhancement cannot be properly discovered or repurposed. Suppose a user originally does not know that the above basic travel planning IM can provide the service he/she desires. Then he/she creates a query by typing in keywords like *buy tickets, credit card* in order to find suitable IMs and collaborative peers. Since there is no string in this IM can match these keywords, this query will be ignored by the keyword-based discovery service. Thus, IMs without semantical enhancement will affect the discovery recall. In the following sections, we propose an approach for semantically enhancing interactions between peers from the perspective of choreography using annotations later on. Figure 5.7 will give the excerpt of proportional source codes of a Web page on which the trade IM described in Figure 3.8 has been annotated with OWL for Processes and Protocols (OWL-P) (Mallya et al., 2005) (a process and protocol modelling language encoded in Web Ontology Language (OWL)) and the OPENK vocabulary for OpenKnowledge. OWL-P defines several process-calculus-related concepts such as *messages*, *protocols*, *roles*, *propositions* and *commitments*. With respect to commitments, commitment machines (Yolum and Singh, 2002) are used to formally represent social relationships between agents. Since OWL-P has enclosed this concept, it can be used for describing protocols that actually form commitment machines forever. LCC so far cannot be used for describing a commitment machine in a straightforward manner because it uses constraints to restrict peers to their obligations. Therefore, checking if policies inside IMs have been obeyed boils down to a Constraint Satisfaction Problems (CSP) for peers to solve. Since LCC itself cannot describe the CSP semantics, here, we use the CSP vocabulary (Badra et al., 2011) to express the constraint solving (constraints are expressed by boolean expressions on fluents) during the execution of IMs. On the other hand, the IM involves extra process-calculus-dedicated concepts such as peers, OpenKnowledge Components (OKCs) and so on, which have not been covered in either OWL-P or CSP. Therefore, we propose a lightweight choreography ontology named as Web Service Choreography As Interaction Models (WSCAIM)[1] by bringing in OWL-P, CSP and OPENK which was originally designed in (Bai and Robertson, 2010) for describing the interaction-driven peer-to-peer community. WSCAIM is a lightweight ontology and its content is illustrated in Figure 5.2 with the assistance of RDFGravity [2].

[1] `http://openk.org/wscaim.owl`

[2] `http://semweb.salzburgresearch.at/apps/rdf-gravity/`

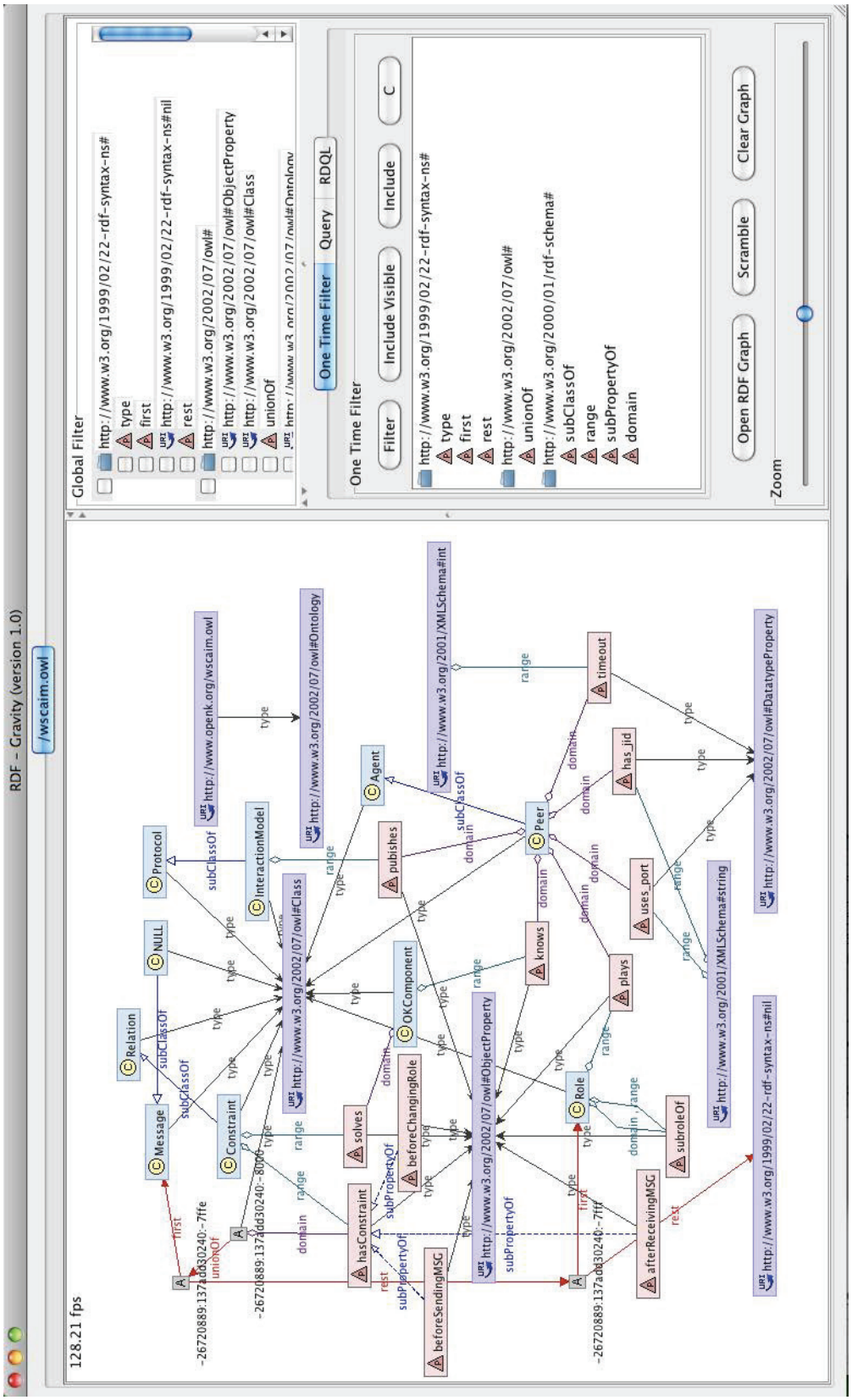


Figure 5.2: WSCAIM ontology visualised in RDFGravity

5.2 Marking Up IMs Using WSCAIM

In this section, we present our semantically enhanced approach for publishing IMs linking them to the Web of data. The corresponding publishing tools have been designed and implemented, which assist IM publishers in annotating IMs and semi-automatically generating Web pages with semantic markups. Methods for consuming embedded annotations are given based on the logics behind the OpenKnowledge system. Web-page-based marking-up provides a lightweight way of embedding surrounding information about IMs into Web pages. Several formats such as Embedded RDF (eRDF), Microformat and Resource Description Framework in Attributes (RDFa) were proposed. eRDF does not support full RDF while Microformat requires users to create new data models for new formats. RDFa allows users to embed any resources complying with the RDF model and has been a W3C recommendation since 2008. Here, we use RDFa in our approach and make it carry the semantic that an IM element should have. Section 5.1 described an example for semantically enhancing IMs using compound vocabularies derived from OWL-P and CSP, which are actually used for annotating different aspects of each IM and will be discussed as follows.

5.2.1 Process-Dedicated Annotations

LCC has been designed as one of related approaches from the diverse *process calculi* family for modelling interactions between peers in an open knowledge-sharing environment. OWL-P is employed here to provide IM publishers with a vocabulary focused on message passing between peers. As mentioned above, in this chapter, IMs will be annotated and serialised in HTML or XHTML ((X)HTML)+RDFa. Figure 5.3 describes an excerpt of message-passing-related RDF triples extracted from an IM document and serialised in Turtle (Beckett and Berners-Lee, 2008).

5.2.2 Constraint-Dedicated Annotations

An IM does not handle peers' commitments explicitly using any particular syntax but commitments are interleaved with LCC codes all over the IM. Peers are expected to have their own constraint solvers and message handlers to serve the similar purpose the commitment machine can serve. Therefore, we use the CSP vocabulary here to

```
@prefix wscaim:  <http://www.openk.org/wscaim.owl#>
@prefix owlp:  <http://research.csc.ncsu.edu/mas/OWL-P/Protocol.owl#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .

<http://homepages.inf.ed.ac.uk/s0896253/purchase.html#purchase> a
 wsc:InteractionModel ;
 rdfs:comment """this is an interaction model about purchase.""" .
:buyer a owlp:Role ;
 owlp:hasRole :shopkeeper .
:shopkeeper a owlp:Role .
:buy owlp:later :receipt .
 [ a owlp:Message ;
  owlp:sendMessage :buy].
 [ a owlp:StringSlot].
 [ a owlp:IntegerSlot].
 [ a owlp:Role ;
  owlp:hasReceiver :shop].
 [ a owlp:Message ;
   owlp:receiveMessage :receipt].
 [ a owlp:IntegerSlot].
 [ a owlp:Role ;
  owlp:hasSender :shop].
 [ a owlp:CompositionProfile ;
  owlp:definedBy [ a owlp:EventOrderAxiom ;
   owlp:stipulate [ owlp:earlier :buy ; owlp:hasRole :buyer],
     [ a owlp:KnowledgeBase ; owlp:consults
     [ owlp:contains [ a owlp:Proposition], [ a owlp:Commitment]]
    ]
]].
```

Figure 5.3: RDF triples related to message passing

annotate the constraint elements of IMs. Figure 5.4 describes an excerpt of constraint-solving-related RDF triples extracted from a semantically enhanced IM document and serialised in Turtle.

Note that the above CSP vocabulary however does not support comparison between values of variables and unfortunately, without the more expressive annotations related to this comparison, it is difficult if not impossible for IM publishers to annotate constraints on relations between variables. Here, we extend CSP with the Mathematical

```
@prefix csp:  <http://vocab.deri.ie/csp#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .

<http://homepages.inf.ed.ac.uk/s0896253/purchase.html#clientArgumentRelation>
  a csp:Relation ;
   csp:isSatisfiable "true"^^<http://www.w3.org/2011/XMLSchema#Boolean> ;
   csp:supports (
    [csp:or
     [csp:var :PC; csp:val :UPC], [csp:var :PC; csp:val :EPC]]).
<http://homepages.inf.ed.ac.uk/s0896253/purchase.html#shopArgumentRelation>
  a csp:Relation ;
   csp:isSatisfiable "true"^^<http://www.w3.org/2011/XMLSchema#Boolean> ;
   csp:supports (
    [csp:and
     [csp:var :CC; csp:val :VISA], [csp:var :R; csp:val :VISA_REC]]
    [csp:and
     [csp:var :CC; csp:val :MASTER], [csp:var :R; csp:val :MASTER_REC]]).
```

Figure 5.4: RDF triples related to constraint solving

Markup Language (MathML) [3] in order to make the data comparison concerned about by the IM constraint solving possible. The triples dedicated to realising this extension are described in Figure 5.5.

5.2.3 Annotation Serialisation

Our approach for serialising this semantic enhancement will be exemplified in this section. Here, we improve the IM described in Figure 3.8 a bit by adding extra arguments *CCC* and *PCP* which denote the remaining credit in one's credit card and the price for a specific product respectively. The new version of the trade IM we have got is described in Figure 5.6.

[3] http://www.w3.org/TR/MathML3/

```
@prefix m3:  <http://www.w3.org/TR/MathML3#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .

...
<http://homepages.inf.ed.ac.uk/s0896253/purchase.html#shopArgumentRelation>
 a csp:Relation ;
  csp:isSatisfiable "true"^^<http://www.w3.org/2011/XMLSchema#Boolean> ;
  csp:supports (
   [csp:and
    [csp:var :CC; csp:val :VISA], [csp:var :R; csp:val :VISA_REC],
    [m3:apply [m3:geq [m3:ci :CCC], [m3:ci :PCP]]]]
   [csp:and
    [csp:var :CC; csp:val :MASTER], [csp:var :R; csp:val :MASTER_REC],
    [m3:apply [m3:geq [m3:ci :CCC], [m3:ci :PCP]]]]).
```

Figure 5.5: Constraint solving with mathematical comparisons

```
a(client(PC, CC, CCC), C)::
   buy(PC, CC, CCC)⇒ a(shop, S)← payby(CC)∧ lookup(S) then
   receipt(R)⇐ a(shop, S)

a(shop, S)::
   buy(PC, CC, CCC)⇐ a(client(_),C) then
   receipt(R)⇒ a(client(_), C)← enough_credit(CC, CCC, PC, PCP)∧
                                complete_order(PC, CC, R)
```

Figure 5.6: Yet another trade IM in LCC

5.3 IM Annotation Injection and Consumption

As already discussed in Chapter 3, the peer community can be initialised by automatically discovering peer groups and in order to describe peer community, OpenKnowledge in the peer community (OPENK) has been designed and serves as a vocabulary for the OpenKnowledge peer community (Bai et al., 2009). By extending WSCAIM, a new vocabulary dedicated to choreography-driven peer community has been proposed and named as VOOK (Vocabulary Of OpenKnowledge) [4]. As aforementioned, there is no explicit concept about commitments inside IMs coded in LCC and also, peers are encouraged to have their own constraint solvers to meet the policies incorporated

[4] http://www.openk.org/vook.owl

```
<html
xmlns=''http://www.w3.org/1999/xhtml''
xmlns:wscaim = ''http://www.openk.org/wscaim.owl#''
xmlns:owlp=''http://research.csc.ncsu.edu/mas/OWL-P/Protocol.owl#''
xmlns:dbpedia=''http://dbpedia.org/resource/''
>
...
<div typeof=''wscaim:InteractionModel''>
 <div rel=''owlp:hasRole''>
  <div typeof=''owlp:Role''>a(<span property= ''openk:has_name''>
 client</span>(
  <span rel=''openk:has_arg'' typeof=''openk:Argument
   dbpedia:Universal_Product_Code''>
  <span property=''openk:has_name''>PC</span></span>), C)::<br/>
  <span rel=''openk:sendout''>
   <span typeof=''openk:Message''>
    <span property= ''openk:has_name''>buy</span>(
    <span rel=''openk:has_arg''>
     <span typeof=''openk:Argument dbpedia:Universal_Product_Code''>
      <span property=''openk:has_name''>PC</span></span>,
     <span typeof=''openk:Argument dbpedia:Credit_card''>
      <span property=''openk:has_name''>CC</span></span>)
    </span>
   </span>
  </span>
...
```

Figure 5.7: Excerpt of the annotated trade IM document

in IMs (Willmott et al., 2006). Moreover, neither OWL-P nor CSP has covered the notions incorporated by peer-to-peer communications. Therefore, WSCAIM has been designed used for serving both above purposes as an imported vocabulary in VOOK.

In Vocabulary Of OpenKnowledge (VOOK), three important notions are *wsc:Peer* (as a subclass of *foaf:Agent*), *vook:P2PCommunity* (a subclass of *sioc:Usergroup*) and *wsc:InteractionModel* (a subclass of *owlp:Protocol*). Here, we make our best efforts in using existing ontologies and creating as less classes/properties as possible. These three notions are coined in terms of concepts involved in the OpenKnowledge ecosystem. Since the Friend of a Friend (FOAF) (Brickley and Miller, 2007) vocabulary is good at describing persons, activities and relationships, we employ it here to describe

peers and their relationships. VOOK has been created according to semantics behind a lightweight language (LCC), which has been employed by the OpenKnowledge kernel as a Web Service choreography description language and gives an intimate connection between interaction processes and the community ontology. To revisit our architecture already discussed in Chapter 4, each peer has a profile composed of triples dedicated to peer features such as which roles this peer can play and which OKCs this peer can provide (with the WSCAIM vocabulary). An OKC is a plug-in component that contains methods used for solving constraints in IMs (sometimes human interventions are also involved in the process of constraint solving). With regard to the Linked Data principles, the Web is essentially very close to a peer-to-peer network because each user on the Web can digest existing data as a consumer or publish new data and also interlink it with other external data as a data contributor. Figure 5.8 gives an illustration of the network topology of a Web-based peer-to-peer community. Each peer has a Knowledge Base (KB) composed of a local vocabulary, a peer profile, an IM repository and an OKC repository. Users log on to or join a specific community via their existing user accounts or register to get new ones.

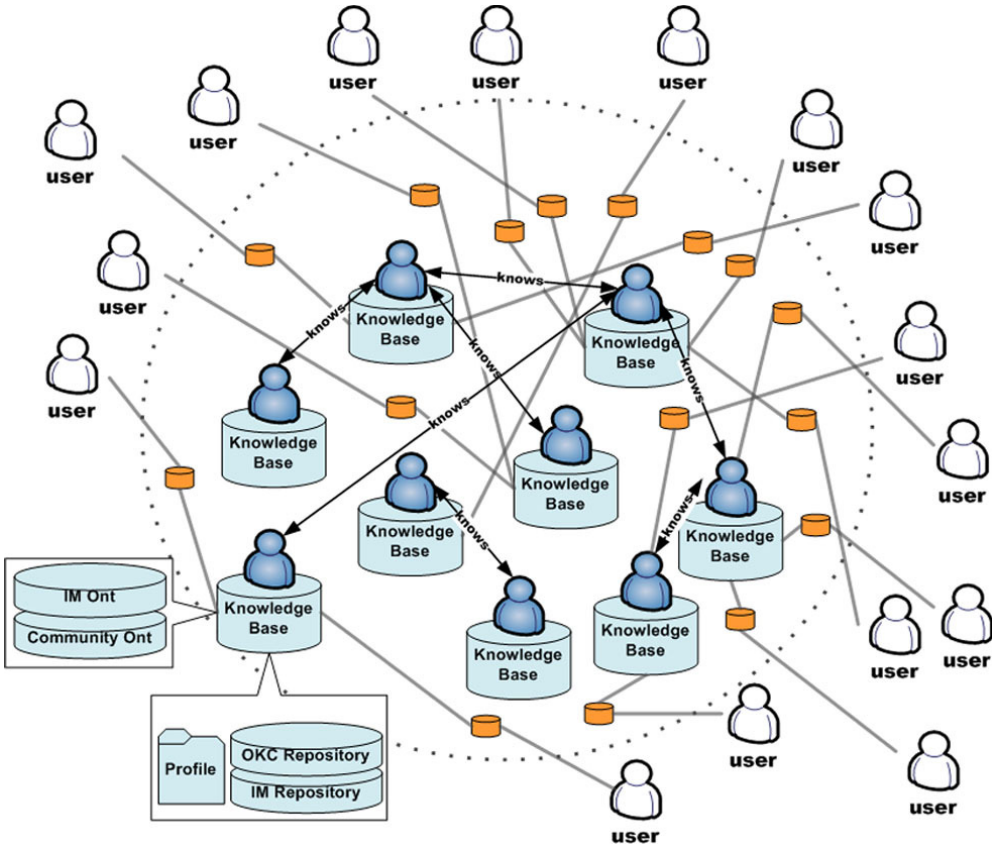


Figure 5.8: Peer-to-peer network topology

5.3.1 Annotation Injection

The annotation-embedding strategy will be employed within the IM publishing process. An IM defines the behaviours of roles involved in a specific interaction and in reality users can use whatever ontologies at their own will to annotate and publish IMs. On the other hand, when peers receive IMs published by others using heterogeneous ontologies, they can also use local ontology matchmakers to fulfil this mapping task, and this has been explored in the OpenKnowledge project (Besana et al., 2009) and will however not be discussed here due to the scope targeted by this thesis. Traditionally, users are allowed to query RDF repositories through exposed SPARQL Protocol and RDF Query Language (SPARQL) endpoints. Compared with that, using annotation-embedded Web pages to publish IMs and exchange them with other peers, publishers can keep their own private KB confidential and just expose the knowledge related to the being-published IMs they want other peers to know. Several problems need to be tackled here for us to achieve the above goal. Firstly, a preprocessor is needed for understanding different input methods in terms of the diverse IM locations. For example, a user can import existing IMs from local file systems or remote repository via Hypertext Transfer Protocol (HTTP), or compose new IMs on the fly. Secondly, during the annotating process, a user needs to be allowed to browse the employed ontology and existing instances in the local profile in order to pick up proper Uniform Resource Identifiers (URIs) to annotate the IM content. Thirdly, a user needs to be allowed to revise existing annotations and dump them in various popular data-exchange formats. Therefore, our IM publishing tool accordingly provides the following functionalities: LCC code editing, peer profile browsing, IM annotating/revising and annotation dumping. These functionalities are achieved through the following modules which have been designed and implemented in our IM publishing tool:

a. **Preprocessing Module.** This module is used for assisting a peer in importing the IM it expects to publish as well as its local profile. An IM can be imported in two ways, one of which is that the user can type in raw LCC code directly and then submit the code as an IM on the fly. The other way is that the user can select and submit an IM file which already exists in the local or remote storage.

b. **Profile Browsing Module.** In order to give IM publishers an intuitive and user-friendly environment for publishing IMs, this module is dedicated to displaying auxiliary information which will be derived from peers' profiles. A class-

hierarchy view can assist publishers in browsing classes and properties that can be used by publishers to annotate IM elements. Information about existing instances inside profiles will be displayed as well for the purpose of reusing peer-side resources.

c. **Annotating Module.** This module helps publishers to describe IM elements by linking them to ontologies. It provides a user interface for publishers to achieve this goal. Via this module, IM publishers can reuse existing instances which have been annotated inside local profile documents, or create brand new instances if needed. For the former case, other existing instances related to the reused instance will be also imported during the annotation process based on Algorithm 2, through which triples containing a particular resource can be extracted from an existing profile in RDF and weaved into an (X)HTML+RDFa snippet.

Algorithm 2: Marking Up Algorithm

Input: The triple store derived from a profile, *triples* and the URI of the targeted instance, R_{uri}.

Output: The published snippet, *snippet*.

```
begin
    pat = new Pattern([[<R_uri>, _, _], [_, _, <R_uri>]]);
    statements = triples.match(pat);
    for each statement stat ∈ statements do
        subject = stat.getSubject();
        predicate = stat.getPredicate();
        object = stat.getObject();
        if subject.uri equals R_uri then
            if object instanceOf Literal then
                snippet = snippet + "<span property=' " + predicate + " ' content=' "
                + object.value + " '/>";
            else
                snippet = snippet + "<span rel=' " + predicate + " ' resource=' " +
                object.uri + " '/>";
        else
            snippet = snippet + "<span rev=' " + predicate + " ' resource=' " +
            subject.uri + " '/>";
```

d. **Revising Module.** Embedded RDFa may be inappropriate or incorrect, so there is a chance for IM publishers to delete, modify or replace existing annotations through this module before the final publication.

e. **Issuing Module.** This module has two functionalities, one of which is to harvest embedded RDFa which belong to a specific IM as new knowledge and add it to the host peer's local KB. The other functionality is to generate a Web document and push it on the Distributed Discovery Service (DDS) thereafter. The RDFa harvest task can be fulfilled using one of existing RDFa parsers [5].

Figure 5.9 is supplied to depict the sequence diagram for the process of publishing IMs. Section 7.4 will present the preliminary implementation for assisting users in publishing IMs based on the above design, and give an example of publishing an IM that describes how a peer sends a greeting text to another peer followed by the latter one displaying the received text. A screenshot on our publishing tool is also depicted in Figure 7.8 of that section.

5.3.2 Annotation Consumption

Published IMs can be consumed in a variety of ways and an intuitive one is to assist the DDS in discovering IMs which meet peers' requirements. The DDS is one of the original key components for the OpenKnowledge system, and with IMs being located on different peers in a distributed manner, the DDS is in charge of discovering the desired IMs and corresponding collaborative peers in terms of the input query (a "query" is here keyword-based). OKBook is an open online platform on which peers are enabled to publish IMs, discover IMs and subscribe/unsubscribe to IMs (Bai et al., 2010). It has a *Discovery Module* (as shown earlier in Figure 3.1) which actually combines the service discovery and the service-repository curation previously done separately in the OpenKnowledge system so we replaced DDS with OKBook (as shown earlier in Figure 3.6). Moreover, this online platform supports not only keyword-based queries but also URI-based queries, which is comparatively less ambiguous thanks to the URI suggested by OKBook. But URIs are usually difficult to memorise and some of them are also heterogeneous. Therefore, on OKBook, users are allowed to use URIs found by RDF search engines such as Sindice or URI curated by DBpedia [6] and discovered

[5] `http://rdfa.info/rdfa-implementations`
[6] http://www.dbpedia.org

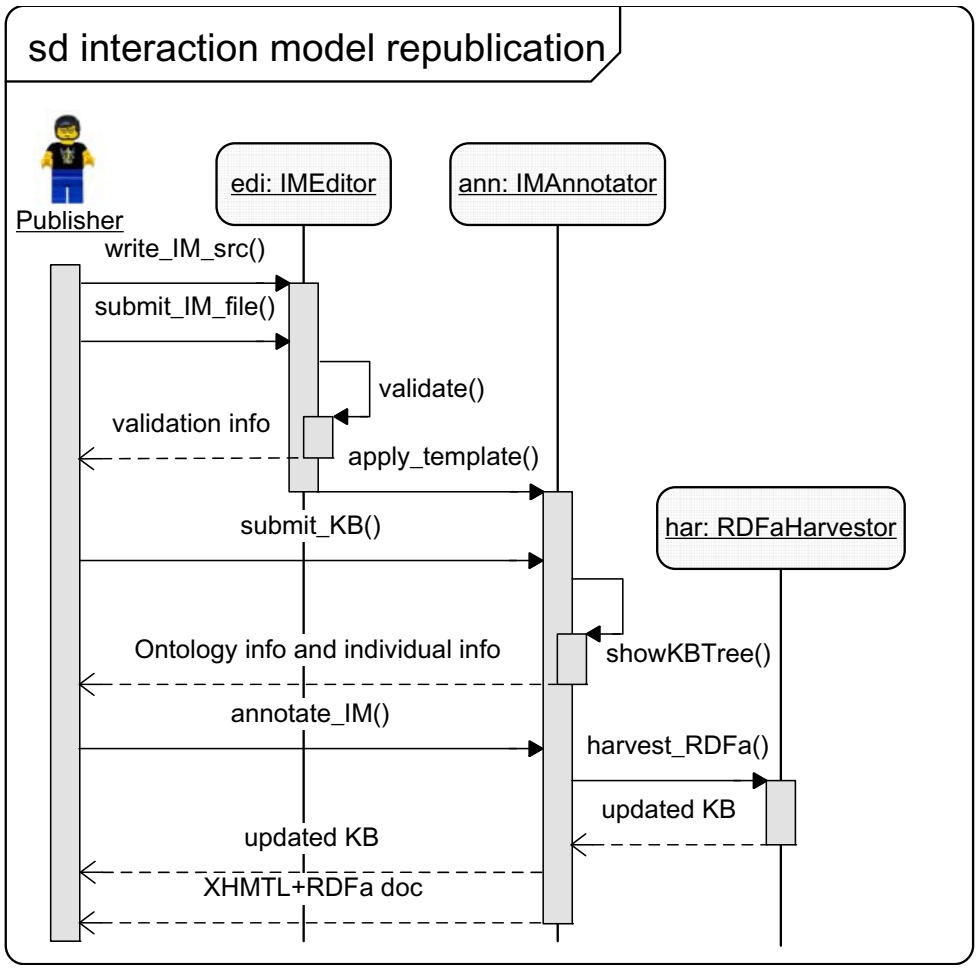


Figure 5.9: Sequence diagram for IM republication

by the Lookup service [7]. After being published, IM documents are already attached with markups and, by harvesting embedded RDF triples, the *Discovery Module* can provide a more precise and extensible query processing also thanks to dereferenceable URIs which are widely spread on the Web of Linked Data. In terms of service discovery, another way of making use of IM annotations is to rely on search engines which are capable of providing metadata indexing. Taking the process-dedicated annotations (discussed in Section 5.2.1) as an example, users can discover desired IMs via those search engines based on annotations associated with processes, policies or commitments (powered by OWL-P) carrying detailed service descriptions which have not been covered by (X)LCC semantics. Under this circumstance, a service requester has a chance to type in finer-grained and semantically richer search phrases, which can be later mapped to corresponding annotations, compared with those phrases used on search engines indexing the annotations about IM elements only.

[7] http://lookup.dbpedia.org/api/search.asmx

Another way of consuming embedded RDFa will be adopted when an IM is executed after all required roles are filled with specific peers. Note that one role could be filled with more than one peer and peers will then be allowed to select those they want to interact with based on the *Peer Ranking* (Robertson, 2008) mechanism employed in the OpenKnowledge kernel. Within the above process, sometimes peers, especially who are interested in data integration and reuse, may query the embedded annotations themselves, which may help them to fulfil complex interactions. For example, a peer may be willing to know the author of an IM, its publish date and revision history, etc. However, the type of these queries is different from the aforementioned one that triggers the IM execution as well as the one used for collaborative-peer discovery. This type of query will be more specific and more targeted at tasks themselves such as a tourist asking an airline service for a cheap flight ticket with a concrete destination and relevant departure/arrival time or an unemployed man asking a job vacancy service for a job with a reasonable salary and located close to where he/she lives. Under this circumstance, peer-side applications or user agents (e.g., Web browsers) should have annotation gleaners equipped to extract the embedded data and conduct interesting queries over them in one way or another, and however, the discussion on this is out of the scope of this thesis.

5.4 Semi-Automatic IM Publication Using RDFa[2]

An approach relying on the WSCAIM ontology to helping publishers to annotate their IMs has been discussed in the above sections. WSCAIM covers the basic elements and skeleton of IMs encoded in LCC and XLCC, and publishers however still need to find the URIs themselves with the help from the Semantic Web search engines (e.g., Sindice) to annotate arguments which are subject to particular use cases and cannot be dealt with in a generic way. On the other hand, an increasing number of WS descriptions in the RDF model exist (e.g., seekda![8] is curating more than 28,000 WS descriptions in its catalogue at the time of writing this thesis), which are retrievable thanks to the accessibility of HTTP URIs, and are ready to reuse by providing IMs publishers with URIs for arguments or other elements which cannot be covered by WSCAIM. With the aim of addressing the RDFa publishing bottleneck, we propose a generic approach to automatically generating hypertext content in (X)HTML+RDFa

[8] `http://webservices.seekda.com/`

using existing RDF triples on the Web and have also implemented a proof-of-concept online tool called RDFa^2 (Bai, 2011; Bai et al., 2011). With the help of RDFa^2, the above problem encountered by IM publishers can be alleviated during the annotation processes. OWL-S (Martin et al., 2004) (formerly DAML-S) and Web Service Modeling Ontology (WSMO) are ontologies used for encoding WS descriptions and adopted in the Semantic Web community in parallel with other alternatives. These ontologies are either from the perspective of service orchestration, or service choreography or both. Inside those WS description documents sit a large amount of triples which are hidden and not accessible to users without expertise. By employing RDFa^2, IM publishers can cherry-pick their favourite WS descriptions from any RDF repository (e.g., seekda!) and publish IMs attached with some of those triples as annotations in a both human-readable and machine-readable manner. The remainder of this section details the our approach to generating hypertext content with annotations.

5.4.1 Topic Nodes and Topic Trees

Our approach for transforming RDF documents to (X)HTML+RDFa pages is based on automatically generated templates. These templates are schematic (X)HTML documents, and have a tree structure. By contrast, the RDF data model is a graph, and cannot be converted to a single tree without duplicating re-entrant nodes. In order to overcome this problem, the conversion from RDF requires users to select a specific node in the RDF graph which then forms the root of a tree of RDF statements. Which node should the user choose? In practice, this seems to follow straightforwardly from the user's goals, namely to focus on the resource which is his or her main topic of interest in the resulting (X)HTML page. For example, in the case of a FOAF file, the obvious resource to choose is the value of the `maker` or `primaryTopic` property.

The node that is targeted in this way is called the *topic node*. The RDF document from which the topic node is derived is called the *RDF context*, and relative to a context $\mathcal{C}$, a set of RDF statements rooted in a topic node is called a *topic $\mathcal{C}$-tree*. We distinguish between two kinds of topic trees, depending on the position (the subject or the object) of the topic node inside a specific triple. Given a resource r, context $\mathcal{C}$, and RDF statement (s,p,o), the *subject (topic) $\mathcal{C}$-tree based on r* is defined as $\{(s,p,o) \in \mathcal{C} \mid s = r\}$, and similarly for the *object (topic) $\mathcal{C}$-tree based on r*.

The notion of a topic tree for a topic node is essentially the same as a *bounded de-*

scription of a resource; that is, where "a sub-graph can be extracted from a data set which contains all of relevant properties and relationships associated with a resource" (Dodds and Davis, 2010). For the sake of clarity, a topic node is not necessarily the global topic of an RDF document; rather, it corresponds to a resource in the document which the user regards as interesting enough to represent in (X)HTML. Figure 5.10 illustrates the selection of a subject topic tree from an RDF context.

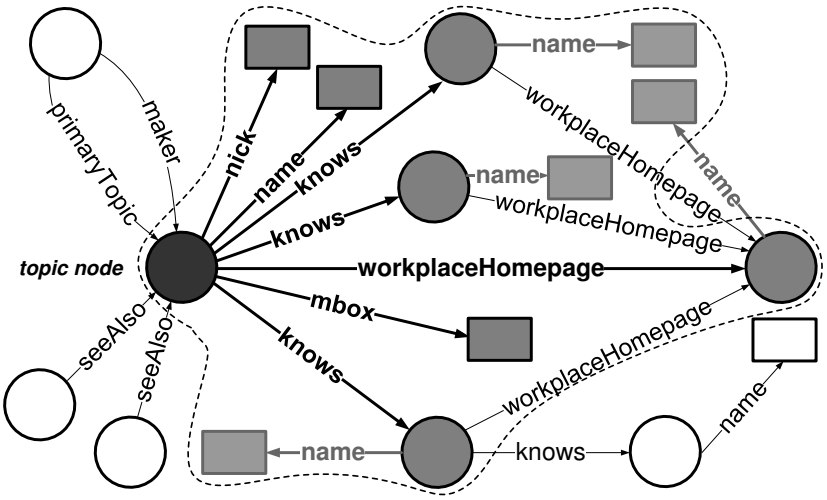


Figure 5.10: *Subject (topic) $\mathcal{C}$-tree* of a FOAF document

In this figure, for the sake of brevity, we have omitted the name spaces (henceforth abbreviated as Name Space (NS)) of all properties here. In this figure, circles denote resources and squares denote literals. The node coloured in dark grey is the current topic node while the sub-graph surrounded by the dashed line is the subject topic tree for this node. The labelling information about the resources in the subject position are also included in the topic tree in order to make the resources themselves human-readable on the RDF-embedded page. Although the most straightforward use case for our approach creates a standalone (X)HTML page from an RDF document, RDFa[2] accommodates cases where the output of this approach is inserted as a snippet into a larger (X)HTML document.

5.4.2 Embedded-Annotation Generation

Our approach to assisting users (e.g., Web content publishers) in generating annotations embedded in their hypertext content is detailed in this section. This approach has the ability to automatically discover a candidate set of topic nodes (from existing

RDF contexts) which can be offered to the user thereafter and also supports federated integration in the sense that users can embed multiple topic nodes from multiple RDF contexts into a single Web page. Within the publishing process, publishers can revise suggested annotated blocks or raw pages in terms of their individual requirements. Moreover, templates are also provided via our approach and customised by publishers (and also stored, loaded and reused) if needed. Algorithm 3 describes the annotation generation process (generating the partial snippet for the subject topic tree) and will be further discussed in the following subsections. Likewise, the snippet generation corresponding to the object topic tree is not described here for the sake of brevity but can be achieved by revising this algorithm and moving the topic node from the subject position to the object position. For mashup purposes, the embedded RDF triples can be harvested and serialised in several formats such as Notation3 (N3) (Berners-Lee, 1998), RDF/XML (Beckett and McBride, 2004), N-Triples (Grant et al., 2004) and Turtle (Beckett and Berners-Lee, 2008).

5.4.2.1 Topic-Node Discovery

In the preceding section, we assumed that topic nodes will be selected by the user. However, this requires the user to understand the basic syntax of the RDF context inside which these node are represented. One way of automatically identifying topic nodes in a given RDF context is to query the document for URIs with properties that are diagnostic of topic-hood, such as `foaf:primaryTopic` or `foaf:maker` in FOAF files. However, not all RDF documents contain such properties, and even in FOAF files which do employ them, they do not always take semantically appropriate values. Consequently, topic nodes cannot reliably be detected just in terms of the semantics of statements in the RDF context itself. Zhang et al. (2007) compared five measurements from three categories (degree centrality, shortest-path-based centrality and eigenvector centrality) for automatically summarising ontologies in a topic-independent manner and their interesting evaluation showed that weighted in-degree centrality measures and several eigenvector centralities all have good performance on ontology summarisation. As analysed in (Bai et al., 2008), for the case that the target RDF documents mix up ontology-related triples and individual-related triples, the above topic-free measurements may be affected by unforeseen noise nodes. Moreover, each property could have a corresponding inverse property so it is difficult if not impossible to draw a conclusion that an RDF node's in-degree (or out-degree) prioritises its out-degree (or

Algorithm 3: RDFa Snippet Generation Algorithm (subject (topic) $\mathcal{C}$-tree)

Input: *topic_uri*, the URI of the topic node and *model*, the model containing triples in the current context.
Output: *rdfa_snippet*, the RDFa snippet representing the information about the inputted topic node.

```
begin
    def rdfa_snippet = getDIVHead(topic_uri);
    def sub_topic_tree = model.getStantementsBySubject(topic_uri);
    def properties = model.getUniquePropertiesBySubject(topic_uri);
    for each property in properties do
        def objects = sub_topic_tree.getObjectsByProperty(property);
        def prop_local_name = property.getLocalName();
        def prop_node_name = (property.getNameSpace() + "_" + prop_local_name
        + "rel").replace("_", "dash");
        def prop_curie_name = model.getPrefix(property.getNameSpace()) + ":" +
        prop_local_name;
        for each object in objects do
            if object.isListeral() then
                rdfa_snippet += "<#if topic." + prop_node_name + "??>" + "<#list
                topic." + prop_node_name + "?keys as key>" +
                getLiteralStyle(prop_local_name, property.getURI()) + ... ;
            else
                def snippet = "";
                if object.isURIResource() && object.getURI().indexOf(".") != -1
                then
                    def obj_uri = object.getURI();
                    def expansion = obj_uri.subString(obj_uri.lastIndexOf("."));
                    snippet += getSnnipetByExpansion(prop_curie_name,
                    prop_node_name);
                else
                    snippet += "<a rel='" + prop_curie_name + "' href='${topic." +
                    prop_node_name + "[key].uri}' onclick='return false;'>${topic"
                    + prop_node_name + "[key].uri}</a></span><br/>";
                rdfa_snippet += "<#if topic." + prop_node_name + "??>" + "<#list
                topic." + prop_node_name + "?keys as key>" + "<# if topic." +
                prop_node_name + "[key].uri??>" +
                getResourceStyle(prop_local_name, property.getURI(), true) +
                snippet + "<#if><#list><#if>";
    return rdfa_snippet;
```

in-degree). In this thesis, we propose an improved algorithm for semi-automatically discovering and recommending topic nodes. Since the RDF data model is a directed graph and nodes are connected to one another through directed edges, one solution for discovering the topic node is based on node connectivity. In other words, the more edges (outgoing or incoming) a node has, the more important it is likely to be. In order to maximise the accuracy of this heuristic, our algorithm selects the top *n* most highly

connected URIs and offers them to users for subsequent confirmation [9]. Perhaps not surprisingly, this algorithm works especially well for RDF documents such as FOAF files that usually do have a central topic.

When a user inputs the URI of a resource that she wants to integrate into her Web page, together with an RDF context, RDFa^2 will query this context with the selected URI for all statements in which the URI is either subject or object. From this set, a subject (respectively, object) topic tree will automatically be selected if it exists. Its root will be the topic node and its corresponding properties and values will be stored in other nodes or leaves. Then the user can refer to any information about this topic node using the path structure `root.predicate.values[key].[resource]` or `root.predicate.values[key].`
`[literal]` in the template which will be discussed in Subsection 5.4.2.3. Here, `root` denotes the resource currently being integrated; `predicate` denotes a specific property with which this resource is associated; and `values` is a list that stores the values of a property (since some properties may have multiple values).

5.4.2.2 Federated-Annotation Generation

We do not want to exclude the possibility of the user selecting more than one topic node from a given RDF context. For example, a user may wish to render the FOAF document vocabulary (i.e., encoded as a set of RDF statements) as (X)HTML, and in this use case, all of the nodes `foaf:Person`, `foaf:Agent` and `foaf:Document`, for example, should be treated as topics. We can use multiple templates to help the user achieve this goal. Once a user selects a temporary topic node, a hash tree, a template and an (X)HTML+RDFa page will be generated based on node occurrences. Meanwhile, the relevant NSs are also grouped and displayed on the final page. Thereafter, the generated (X)HTML+RDFa snippets will be automatically combined into a single snippet.

It is not uncommon that users publish an (X)HTML+RDFa page using triples from different RDF sources (or in our terminology, from different contexts). We can accommodate this in a way similar to our approach to dealing with multiple topic nodes. Our approach supports federated integration by managing the NSs derived from dif-

[9] The value of n can be any reasonable integer. Although RDFa^2 takes n to 10, by default it only just shows the top three URIs to users. It is also worth noting that blank nodes are filtered out from the set of candidates.

ferent RDF documents separately and combining them at the final stage. However, it should be noted that different vocabularies do not necessarily employ the same Qualified Name (QName) prefix for a given NS. *prefix.cc* (*PCC*) [10] alleviates the issue that RDF documents involve different prefixes indicating the same NS or the same prefix indicating more than one NS by allowing users to look up the collected NSs on *PCC* and vote for their favourite ones. Nevertheless, it is difficult if not impossible to stop people from using ambiguous prefixes. Our approach can automatically detect if a prefix is ambiguous across a set of contexts, and will synthesise new prefixes to ensure disambiguation by generating different prefixes as substitutions. Moreover, many of the NSs in the original RDF context set are unused in the final (X)HTML+RDFa Web pages. In order to avoid an unnecessary burden on browsers rendering the page, the NSs which are not used in the user's RDF-embedded Web page will be automatically excluded. Note that RDFa 1.1 harnesses `@profile` to come over the lengthy declaration of NS prefixes recommended in RDFa 1.0 and this can be also used for avoiding possible ambiguous prefixes to some extent.

Figure 5.11 illustrates how our approach assists users in creating Web pages annotated with RDF triples derived from different data sources. Users inform RDFa^2 of the target in one or more RDF contexts by providing one or more Uniform Resource Locator (URLs). These documents will be retrieved on the fly and each of them forms an RDF context. After the topic nodes are selected, triples related to them will be extracted. Finally, the page with RDFa annotation will be sent back to users.

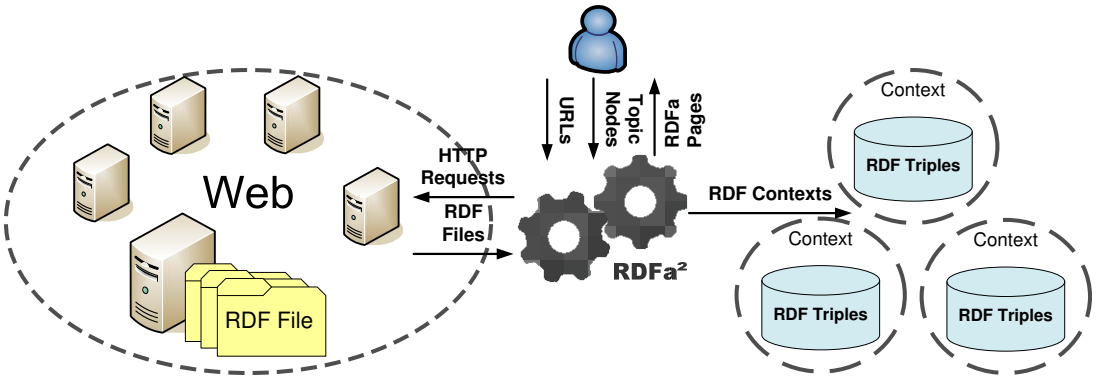


Figure 5.11: Context-based federated integration

[10] `http://prefix.cc`

5.4.2.3 Customisation and Template Reuse

One of the primary functions of our approach is to automatically carry out a template-based transformation of RDF to (X)HTML+RDFa. However, the result of the transformation will almost certainly not be in the precise form required by users, and consequently it is important to allow users to further edit the output. The RDFa^2 interface provides the user with both a rendered preview and the source code of the generated (X)HTML+RDFa. Users without expertise in RDF(a) can modify the output by clicking and editing elements on the preview page or editing the content in the What-You-See-Is-What-You-Get (WYSIWYG) way as shown in Figure 5.12. More experienced users can edit the page source and check its preview but it is recommended that revisions are limited to the text nodes of the page since manually edited RDFa needs revalidation.

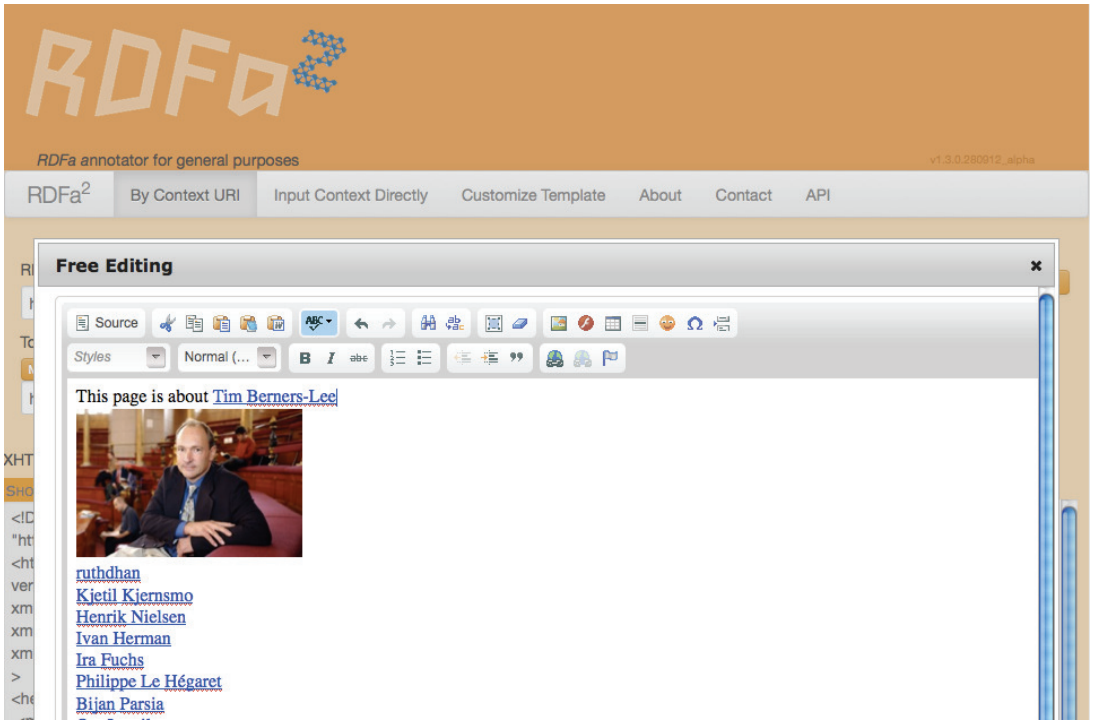


Figure 5.12: Personalise the automatically generated Web page

When users deal with a great number of RDF documents of the same type (e.g., all of them are FOAF documents), they may have to carry similar or even identical manual revisions for each document processed by RDFa^2. To avoid this unnecessary effort, we provide users with another way of personalising the RDFa-embedded web pages by letting them revise the templates. Each transformation will generate a template and this

template will be returned before being applied to the RDF context. A basic template is generated using placeholders of the kind standardly offered by template tools (e.g., FreeMarker [11] applied here). Each placeholder indicates a piece of information which will be extracted during the transformation process (e.g., `personal.firstname` and `personal.lastname` are two placeholders which will be replaced with the first name and the last name of a particular person, respectively). As long as a template is generated, a hash tree that stores the data about the topic nodes is also generated, based on the RDF context: we call this an *intermediate tree*. The structure of the intermediate tree is derived from the structure of the topic tree but is more friendly to templating.

The triples taking the topic node as subjects or objects may take literals or other resources as their objects. Both of these two cases have to be taken into consideration before the template is generated. If the object is a literal, it will be enclosed within an HyperText Markup Language (HTML) tag with `@property` (`@ATTRIBUTE` is used hereafter for denoting a tag's attribute in terms of the XML Path Language (XPath) syntax) indicating the predicate attached with this object. If the object is a resource, it will be enclosed within by an HTML tag with `@resource` taking this object as its value and `@rel` indicating the predicate attached to this object. As mentioned in Section 5.4.1, the text value of this tag node will be the preferred label (if exists) of the resource rather than its URI. This complies with the modelling pattern introduced in (Dodds and Davis, 2010).

5.4.2.4 Self-Adaptability and Reflections on RDF Features

Our approach queries the RDF context using SPARQL with the topic node either given by the user or discovered by the topic recommender semiautomatically, which has been discussed above. The result will be used for replacing the pre-generated placeholders insides templates. In a specific RDF vocabulary, some properties may be defined as functional properties (e.g., `foaf:gender` and `foaf:primaryTopic` in FOAF). Each of them only takes one object or one literal as its value. Other properties (e.g., `foaf:maker` and `foaf:member`) may take more than one object or literal as their values. The SPARQL query results are grouped in terms of properties. With respect to the evolution of an RDF vocabulary, new classes or properties may be involved and some classes or properties may be deprecated. Since templates are created and

[11] `http://freemarker.org`

applied on the fly and always are based on the given vocabularies (RDF contexts), the above evolution will be transparent to users. For those users who want to reuse their templates, their existing templates can be merged with the newly generated ones. However, some manual reconciling work on these two kinds of templates and basic knowledge of (X)HTML+RDFa may be involved within this process.

According to (Adida et al., 2008), `@resource` and `@href` can be used for hooking the object of an RDF triple. The value of the former is a URI which is "not intended to be clickable" and normally denotes a non-information resource while the value of the latter is a URI which normally denotes a information resource. The minters of non-clickable URIs need to provide relevant information resources as these URIs' representations (Lewis, 2007). RDFa^2 currently assumes each non-information resource has an informational representation and by clicking it, users will be redirected to another information resource associated with it. Thus, either information resources or non-information resources will be wrapped in `<a>` tags and attached to `@href` rather than `@resource` here. Since `@href` supports only URIs, the object of each RDF triple will not be expressed in Compact URI (CURIE) syntax in the final page. With respect to Blank Nodes (BNodes), the labelling property (if exists) and corresponding value surrounding a specific BNode in the original RDF context will be used as the representation. Nevertheless, users are recommended not to use BNodes when publishing Linked Data on the Web (Bizer et al., 2007).

5.4.2.5 Linking Annotations to the LOD Cloud

There is one step to go before RDF triples are injected into Web pages because these embedded triples may otherwise cause provenance and trust issues. RDF statements are focused on describing who said what but statements themselves may or may not be true. Additionally, the licence is another thing that should not be ignored especially when users attempt to reuse data by other data providers. Therefore, the enriched documents need to be associated with provenance information and linked to the Linking Open Data (LOD) Cloud [12]. Here, we use the Vocabulary of Interlinked Datasets (voiD) (Alexander et al., 2009) to describe the relationships between the annotations and the RDF contexts from which the harnessed triples are derived. This vocabulary has been used here due to its simplicity and concision but alternative linked dataset

[12] `http://linkeddata.org`

vocabularies could be applied here for the same purpose. Suppose the URI of the topic node is denoted by T_{uri} and the URI of the RDF context (provenance) is denoted by C_{uri}. An (X)HTML+RDFa snippet will be automatically generated to describe the provenance of T_{uri} as follows:

```
<div about="Turi" xmlns:void="http://rdfs.org/ns/void#"
   xmlns:dcterms="http://purl.org/dc/terms/">
   <span rel="dcterms:isPartOf">
      <span typeof="void:Dataset">
         <span rel="void:dataDump" resource="Curi"/>
      </span>
   </span>
</div>
```

Online profiles have been widely used by various Web sites for managing user identification. FOAF is currently one of the most widely used profile vocabulary for the RDF data model. RDFa^2 can help users inject their FOAF triples into their online profile documents such as homepages. Our experiment first involved asking students who participated in a masters level course on Semantic Web technologies to use RDFa^2 to publish their own profiles (FOAF documents) along with information about their favourite actors/actresses denoted by URIs minted and curated on DBPedia[13] and submit URLs of these documents to Sindice[14]. Fresnel (Pietriga et al., 2006) and SPARQLScript (Nowack, 2008) were also introduced during the course as alternatives. In total, 64 students were on this course and 60 of them successfully submitted their assignment reports. On a public server, each student has been allocated personal space to store his or her own documents (e.g., the homepage). By searching documents on Sindice with the domain name of the above homepage server as well as students' matriculation numbers, we found that 96.6% of students finally published their RDFa profiles and also successfully managed to make Sindice index them. 93.33% of students chose the first topic nodes (at the top of the generated topic-node lists) recommended by our topic-node discovery algorithm as their priority within the process of RDFa snippet generation while two students chose the second topic nodes as their priority. Based on their feedback, RDFa^2 made straightforward the process for generating triple-embedded Web pages from existing RDF data sets and Fresnel as well as SPARQLScript are however more flexible for users with expertise on specific languages as well as RDFa itself to customise pages.

[13] http://dbpedia.org/

[14] http://www.sindice.com/main/submit

Summary

In this chapter, by introducing interaction-dedicated vocabularies, we have described our approach to generating, injecting and consuming annotations of IMs. A proof-of-concept implementation has been delivered for helping publishers to semiautomatically generate ready-to-publish IM Web pages from existing RDF triples and a demonstration can be found here.[15] More experiments and use cases on IM annotations will be provided in Chapter 7 and before that, based on the approaches discussed previously, a generalised specification on forming peer communities will be presented in the next chapter, as a guider for peer community builders.

[15] http://demos.inf.ed.ac.uk:8836/rdfasquare/

Chapter 6

Social Group Formation and Maintenance

An increasing number of Social Networking Sites (SNSs) and online communities have emerged, aiming at providing users with online experiences which run alongside their real daily life. This virtualisation movement facilitates the development of techniques for instant messaging, electronic payment and social relationship management, etc. However, most of these techniques are focused on forming online communities in architectures that have several well-known disadvantages (Yeung et al., 2009). Most machines are behind firewalls and clients cannot interact with one another without the supervision of a centralised computer and it is common that in this centralised client/server environment, the vendor (e.g., Facebook, Myspace, Twitter, etc.) is not only the community curator but also the data curator delegating each network node. Little work has been done in forming and maintaining communities in more dynamic and decentralised environments such as *ad hoc* and peer-to-peer networks. On the other hand, traditional online communities are usually formed and augmented via subscription links which can be either unidirectional (e.g., Twitter) or bidirectional (e.g., Facebook). However, in real life people are not simply connected via links but via interactions that are more dynamic and task-dependant. For example, a single person may play different roles within different interactions and these roles will determine his/her temporary relationships with others. *Role* is a common and important concept in the real world but it is often ignored or is at most implicit in traditional online communities. In this chapter, based on semantic-enriched peer interactions, a generic specification to forming and evolving online communities is proposed for peers to share

knowledge in a decentralised manner. Inside the ecosystem inspired by this specification, peers can freely join any community without sacrificing their private data or rights because each of them can personally control the Interaction Model (IM) it uses in interacting with others and can choose the role to play during the interaction. Section 6.1 provides a specification by generalising our approaches which have been discussed in previous chapters. Section 6.2 analyses features of the IM, which are related to service composition. Section 6.3 describes the social effects of the ecosystem driven by interactions and powered by the peer-to-peer community.

6.1 Interaction-Driven Peer-to-Peer Community Specification

This specification involves terms defined according to either common concepts derived from traditional peer-to-peer networks and Web architecture or particular concepts minted for describing the interaction-driven peer community. A glossary of peer community terms can be found in the Glossary section of this thesis. Some of them have been used in previous chapters, and considering the distributed and dynamic environment, this glossary gives more concrete definitions on these terms which are significant when peers communicate and share knowledge with each other in an open environment. In this specification, all statements of optional behaviour use either *must*, *must not*, *required*, *may*, *may not*, *should*, *should not*, *recommended* and *optional* which are interpreted as described in (Bradner, 1997).

6.1.1 Messaging Among Interaction Participants

In the peer community ecosystem, peers exchange messages using the Hypertext Transfer Protocol (HTTP) protocol essentially. The communication servers, such as Extensible Messaging and Presence Protocol (XMPP), also use HTTP in a polling or pushing way. Since (in polling) many of polls do not return new data, pushing is comparatively more efficient. Messages passed between different peers may be processed and delivered in different ways. In this specification, a super peer is either a discovery peer or a communication peer (or both) which is always online, changes less frequently and provides trustworthy services continuously while peers which are not super peers are

called normal peers. As mentioned in Section 4.1.2, a peer's community ID and communication ID can be the same one when the registered server is installed with both a discovery service and a communication service. Message passing between interaction participate fall into the following categories:

a. *Between Normal Peers and Discovery Peers*. For users' activities (e.g., browsing an IM, querying a SPARQL Protocol and RDF Query Language (SPARQL) endpoint, signing in/out, etc.), the browser itself is the equivalent of a peer which delegates the user to send HTTP queries to another peer aware of by the community and render the result sent back. A discovery peer behaves like a resource sniffer and should be integrated with functionalities such as peer group discovery and the meta-search engine, which can help peers in finding their group members of similar interests as well as the desired IMs and corresponding collaborative peers. Other functionalities which are dedicated to bootstrapping peer groups or providing searching with higher precisions and recalls may be integrated into discovery peers which should be always-online and change less frequently. Here, the messages are passed between peers and discovery servers according to HTTP 1.1.

b. *Between Normal Peers During an Interaction*. During the running of a specific IM, involved peers talk to one another according to the protocol employed by the communication server. There is no coordinator (middle man) in charge of interpreting messages involved and each of these peers has its own message handler which is capable of parsing the incoming messages and sending the results to the peer's local Lightweight Coordination Calculus (LCC) Interpreters (LCC Interpreters (LCCIs)) as well as encapsulating the outputs after interpretations and sending them to other collaborators. The message handler may be implemented in more than one form such as a browser plugin or a desktop application which can work with Transmission Control Protocol/Internet Protocol (TCP/IP). Note that, though LCC is recommended, it may be replaced with other service choreography description languages and then LCCI should be replace with the corresponding interpreters if possible.

c. *Between the Discovery Peer and the Initial Role*. As soon as all roles defined in an IM are filled in by specific peers, a discovery server should bootstrap the interaction by sending the message of subscription information to the peer which is about to play the initial role. This message should contain the interaction ID,

the content of this IM or the indicator pointing to the content, communication IDs of all the peers which will be the involved in the target interaction as well as the corresponding roles also described in the same IM. Here, the messages are passed between the discovery peer and the peer about to play the initial role according to HTTP 1.1.

d. *Between Discovery Servers*. When a peer cannot find an IM which meets its requirement, the discovery server on which this peer has logged will forward its message (HTTP request) for querying IMs (in SPARQL) to other adjacent discovery servers. The "adjacent" here should be defined by the designers themselves and this will possibly bring overhead to the discovery server. So the server must deal with the overhead without affecting the precision and the user experience of the discovery. Here, forwarding messages are passed between discovery servers according to HTTP 1.1.

e. *Between Normal Peers and Communication Peers*. Peers send messages to and receive messages from communication servers according to the employed protocol. The protocol must guarantee that in most cases this message passing will not be blocked. For instance, even if the Transmission Control Protocol (TCP) port is blocked by a firewall, communication servers should still communicate with peers via the normal HTTP port which can maintain the robustness of the connectivity. XMPP is recommended as the protocol to handle the interactions between normal peers and communication peers.

6.1.2 Service Registry inside the Peer Profile

Peers subscribing to a specific IM need to provide services to satisfy the corresponding roles' constraints and fulfil their committed obligations. The services a peer can offer should be described in its peer profile which may be serialised in Resource Description Framework (RDF) or Resource Description Framework in Attributes (RDFa) or other Web-friendly data-exchange formats. Simple Object Access Protocol (SOAP) (Curbera et al., 2002), REpresentational State Transfer (REST) (Fielding, 2000), or RESTful SOAP (Mitra and Lafon, 2007) are optional and can be used for launching a peer's Web Service (WS) interactive interfaces but the peer should indicate which option a particular service chose. Technically speaking, since REST-based WSs have a lower barrier to entry than SOAP-based ones, the REST way is recommended in this speci-

fication. In the OpenKnowledge (Robertson, 2008) system, services are wrapped into the OpenKnowledge Components (OKCs), and Figure 6.1 and Figure 6.2 showcase a peer's specific OKC described in RDF with the SOAP model and the RESTful model respectively:

```
<openk:Peer rdf:about="http://www.example.org/pid">
 <openk:registerService>
 <openk:OKC rdf:resource="http://www.example.org/okcs/enoughCredit">
   <openk:serviceArchitecture>SOAP</openk:serviceArchitecture>
   <openk:request>
    <openk:method>POST</openk:method>
    <openk:param>
     <openk:bindings
     rdf:resource="http://www.example.org/okcs/enoughCredit/wsdl"/>
     <rdf:label>envelope</rdf:label>
    </openk:param>
   </openk:request>
   <openk:response>
    <openk:chunk rdf:datatype="xsd:boolean"/>
    <openk:bindings/>
   </openk:response>
  </openk:OKC>
 </openk:registerService>
</openk:Peer>
```

Figure 6.1: OKC described with the SOAP model

In the above example, an OKC identified by "`http://www.example.org/okcs/enoughCredit`" will be registered as soon as this peer registers on a community peer which will harness all OKCs described in its profile and do interesting computations based on them later on. OKCs will be invoked when the peer's local LCCI finds relevant constraints which need to satisfy during the running of a specific IM and OKCs can be either internal (on the same server with the peer) or external (on different servers). Internal OKCs are always accessible to the host peer whilst external OKCs are accessible only if they are published as public on-line services and the retrieving peer holds the corresponding legitimate authentication.

As described in Chapter 5, IM publishers may mint new Uniform Resource Identifiers (URIs) or reuse existing ones to annotate the IM being published. For a pair of a

```
<openk:Peer rdf:about="http://www.example.org/pid">
 <openk:registerService>
  <openk:OKC rdf:resource="http://www.example.org/okcs/enoughCredit">
   <openk:serviceArchitecture>REST</openk:serviceArchitecture>
   <openk:request>
    <openk:method>POST</openk:method>
    <openk:param
    rdf:resource="http://www.example.org/okcs/enoughCredit/args/creditCard">
     <rdfs:label>cc</rdfs:label>
    </openk:param>
    <openk:param
    rdf:resource="http://www.example.org/okcs/enoughCredit/args/productCode">
     <rdfs:label>pc</rdfs:label>
    </openk:param>
   </openk:request>
   <openk:response>
    <openk:chunk rdf:datatype="xsd:boolean"/>
    <openk:bindings/>
   </openk:response>
  </openk:OKC>
 </openk:registerService>
</openk:Peer>
```

Figure 6.2: OKC described with the RESTful model

constraint and an OKC, if they were annotated using different URIs in the IM page and in the OKC provider's profile, respectively, ontology mapping (Choi et al., 2006) may be used for mapping one URI to the other, and needless to say, the accuracy of this mapping will affect the IM discovery process. Discussion on this is however out the scope of this specification.

6.1.3 Peer CRUD Features

Create, Read, Update and Delete (CRUD) are basic functionalities for database management systems. Likewise, in the peer community, peers are also associated with these features in order to work as persistent storage:

a. *Create*: peers should only create and publish documents (e.g., profiles, IMs) on their own local servers.

b. *Read*: peers may not only read documents on their own local servers but also retrieve remote documents dereferenceable via specific URIs using HTTP requests or query remote SPARQL endpoints.

c. *Update*: peers should only modify documents on their own local servers.

d. *Delete*: peers should only remove documents on their own local servers.

Other subscribers (e.g., OKBook servers) of these documents should be automatically notified of any document updates via Publish/Subscribe protocols (e.g., the PubSubHubbub protocol as discussed in Section 3.1.2).

6.2 Service Composition and IMs

WS composition can be investigated from two different perspectives: orchestration and choreography. Actually, both views are supported in IMs. Orchestration is more concerned about the internal service composition for a single peer, and in an IM, the constraints a clause uses to define roles are an orchestrating composition of these services this role needs to provide in order to satisfy these constraints. On the other hand, choreography is more concerned about the external service collaborations among different peers, and from this perspective, the IM itself is a description of WS choreography. Orchestrated services are OKCs which are either accessible endpoints (SOAP style) or on-line resources (REST style) as aforementioned. Since each IM is also regarded as an on-line document and has a URI pointing to itself. From both above perspectives, WSs and their compositions are all accessible via HTTP requests and also indexable on various search engines which actually become distributed WS discovery hubs in the peer community.

6.3 Social Effects

Just like human beings influence one another in society, there are social phenomena and effects within the peer community as well including relationships, social groups, trust, reciprocity and tolerance.

6.3.1 Peer Relationship Layers

The formation of the peer community is driven by interactions which include communication, transactions, data sharing, competitions, collaborations and any other activities involving at least two peers. Peers's multidimensional relationships in this distributed environment are more sophisticated and prolific than binary relationships which have been widely used on most SNSs. Hierarchically speaking, a peer community has two layers and the top one is composed of nodes and edges but peers do not connect with one another directly via links but via interactions. The bottom layer is composed of conventional binary relationships for "physically" connecting nodes. In Figure 6.3, these two layers can be briefly depicted via an example related to the simple trade IM used in Chapter 3. Each IM has one or more interactions (circles in this figure) as its instances which may involve different peers playing the same role(s). Inside a particular interaction, involved peers may be collaborators with one another but outside it, peers may be either friendly or rival to one another. In Figure 6.3, suppose the underlying market is a buyer's market in which the supply from producers (e.g., *shop* in Figure 3.8) is more than the demand from consumers (e.g., *client* in Figure 3.8). Unsurprisingly, peers playing the *shop* role become competitors to one another while peers playing the *client* role are friendly since some of clients may follow other clients which are aware of source information about better sales and some of them may even become friends in order to share the information of common interest without others in between.

Note that peers which are rivals for one type of interaction may be collaborators for another type of interaction. In the example above, the relationships would be changed in the market dominated by sellers. Based on these analysis, we can see that the interaction-driven peer community is not static and peer's relationships differ according to viewpoints as time goes by.

6.3.2 Peer Group

Peers may join or leave a community at their own will and this process is termed *community evolution*. Within the growth of a peer community, a group of peers will interact with one another more frequently due to some particular common types of interactions but rarely interact with other peers outside this group. These peers actually

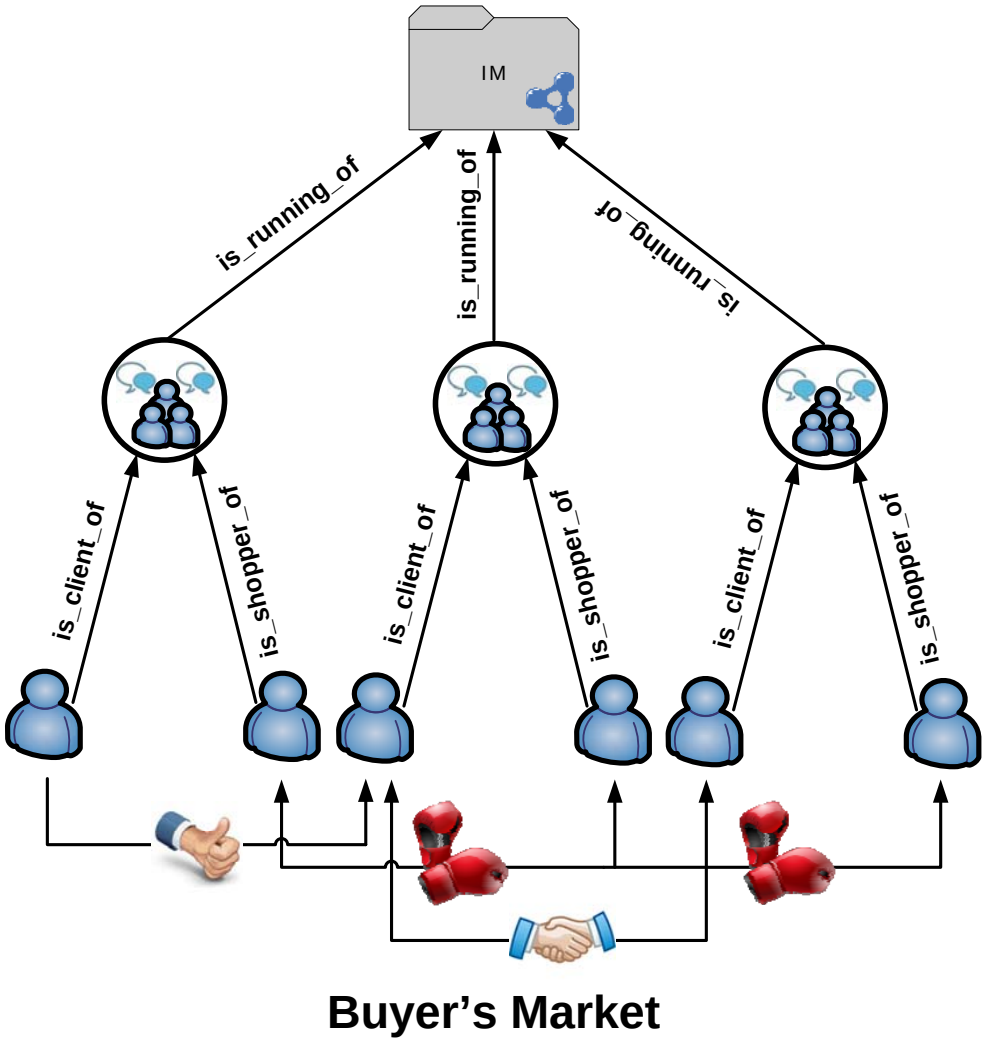


Figure 6.3: Community relation layer

form a sub-community called a *peer group* in this specification. As mentioned in (Nejdl et al., 2003), peer clustering can benefit query routing in peer-to-peer networks and the authors classified this clustering into three categories: ontology-based clustering, rule-based clustering and query-based clustering. In this specification, peer grouping is driven by shared IMs which may be viewed as protocols (or rules) for guiding peers to interact with one another so it shall fall into the rule-based clustering category.

A group may be formed surrounding one or more IMs which are relevant to the interests of this group itself in one way or another. Peer groups do not have explicit boundaries and may overlap each other. Through interacting with the overall community, groups evolve continually by so new members may join in and existing members may leave. In the interaction-driven peer community, IMs play a key role within the formation of peer groups. There is however more than one approach to discovering

to which group a particular peer belongs and the results of grouping the same bundle of peers may be different according to these different approaches, which basically fall into two categories:

a. *Single IM Focused.* An intuitive and simple approach to grouping peers is that each IM will yield a peer group in which all the peer members were involved in the running of this IM in the past. This approach works especially when a limited number of IMs are shared in the current community.

b. *Multiple IMs Spanned.* Some IMs are related to each other and they share more common peers than other IMs in their past executions. Starting with a single particular IM, we can discover those peers also involved in the executions of other IMs. This approach works especially when a large number of IMs are shared in the current community.

6.3.3 Trust

In the peer community, peers may come across other strange ones with which it has not interacted before and trust therefore becomes important for them to make a decision on whether or not to interact with those strangers. Efforts have been paid to finding a way of managing trust in a decentralised environment (Aberer and Despotovic, 2001; Cornelli et al., 2002; Damiani et al., 2002; Kamvar et al., 2003). Since our peer community driven by peer interactions, it is straightforward that the trust can be calculated according to a peer's performance on its historical interactions (i.e., interaction logs) and OKBook adopts a lightweight `foaf:knows`-driven solution in calculating the trust degrees between peers and it is detailed in this subsection.

There is originally no notion of degree for `foaf:knows` prescribed in the FOAF vocabulary and the issue of "to what extent a peer can believe what another peer commits" has not yet been addressed. Needless to mention, a peer knowing another peer does not imply that they 100% trust each other. We could ignore this issue and leave subtle judgements of trust to human interpretation but, as an illustration of how this issue might be addressed, we propose an approach for letting peers declare how well they know one another.

Suppose Peer P_a has OKBook curate its profile copy in which P_a is linked to Peer P_b via `foaf:knows`. At the time when this link was about to be created, P_a was asked "How

well do you know P_b?" by OKBook. For the sake of simplicity, P_a had the chance to provide a percentage indicating the extent to which it trusts P_b. By subjectively choosing a percentage, P_a actually assigned a weight to the outbound $foaf:knows$ link pointing to P_b. OKBook however will not simply take this percentage as P_a's degree of trust in P_b because, in the real world, a person meets another normally through introduction from their common friends or events they were both attending (i.e., people make friends in terms of their similar interests). PageRank (Page et al., 1999) can indeed be applied to the peer community which is composed of peers and directional connections thereof to calculate trust degrees and further discussion on this is outside the scope of this thesis. Here, we however expect peers to have subjective options on trusts in others instead of using an overall centralised algorithm such as PageRank to rule them all. Driven by this motivation, we give a lightweight algorithm for calculating the trust degree, as shown in Algorithm 4.

Algorithm 4: Trust-Degree Calculation Algorithm

Input: A peer P and a community C in which P participates.
Output: An array *trust* containing P's degrees of trusts in other peers participating in C.

```
begin
    for each peer p in C do
        trust[p] = 0;
        for each neighbour n_p of P do
            trust[n_p] = trust_in(P, n_p);
            //trust_in() returns the original degree of trust in an adjacent peer.
        S = the set of all peers in C except P;
        while |S| ≠ 0 do
            u = peer in S with highest trust degree;
            if trust[u] == 0 then
                break;
            else
                remove u from S;
            for each neighbour n_u of u do
                new_trust = trust[u] × trust_in(u, n_u) - 1/(|S|−1);
                if new_trust > trust[n_u] then
                    trust[n_u] = new_trust;
        for each peer p' in S do
            trust[p'] = trust[p'] + 1/(|S|−1);
            if trust[p'] < 0 then
                trust[p'] = 0;
```

In this algorithm, note that the trust degree is calculated not only based on the weight

associated with each link in the graph of linked peers but also based on how many links via which a peer is pointing to others. $\frac{1}{|S|-1}$ punishes the trust degree once a new link is added to the path, where $|S|$ denotes the size of the set S. The reason for carrying out this punishment is because the more peers a `foaf:knows` path goes through the less trustworthy its source peer (where the path begins) will be to them. The punishment weight relies on the total number of peers in a community. Suppose the source peer reaches the target peer (where the path ends) by going through several ones and each of them in between 100% trust its superseding peers. If there is no punishment involved, a conclusion will be drawn that the source peer knows the target peer well. Actually, this is not the case in the real world especially when the size of the community is humongous. If the punishment weight is set to $\frac{1}{|S|-1}$, the above extreme case will not happen (note this punishing strategy will not work when $|S|$ is too small). However, since the minimum size of a peer community can be possibly predefined by peers who are willing to run OKBook servers, the circumstance under which the size of community is too small (e.g., a community just accommodates two peers) is unlikely to occur.

Our trust-degree calculation algorithm has been designed to be associated with `foaf:knows` links. Since `foaf:knows` is not defined as a symmetric property in the original FOAF vocabulary and the description on this property is generic, the direction of `foaf:knows` needs to be clearly considered in our algorithm which is "direction-sensitive". Hence, peers described in Algorithm 4 form a directed graph in which network nodes are connected by directional `foaf:knows`.

As an example, Figure 6.4 depicts the relationships between peers participating in a community with the size ten (it accommodates ten peers). Suppose *PeerA* is the source peer whose degrees of trust in other peers are going to be calculated. Based on Algorithm 4 and initial trust degrees associated with each edge in the above community graph, *PeerA* can figure out the extent to which it can trust other nine community members. From the above figure, we can see that even though *PeerA* does not know *PeerC*, *PeerE*, *PeerF*, *PeerG*, *PeerH*, *PeerI* and *PeerJ*, it can still trust some of them thanks to propagations of trust degrees. For instance, if we set the trust threshold to 70%, in Figure 6.4, *PeerA* can trust *PeerC*, *PeerE* and *PeerG* besides *PeerB* and *PeerD* known beforehand. In the real world, people make new friends and trust them through their existing friends who are trustworthy. Here, "trustworthy" is actually a neutral word (neither positive or negative) and a trustworthy peer does not imply that a task

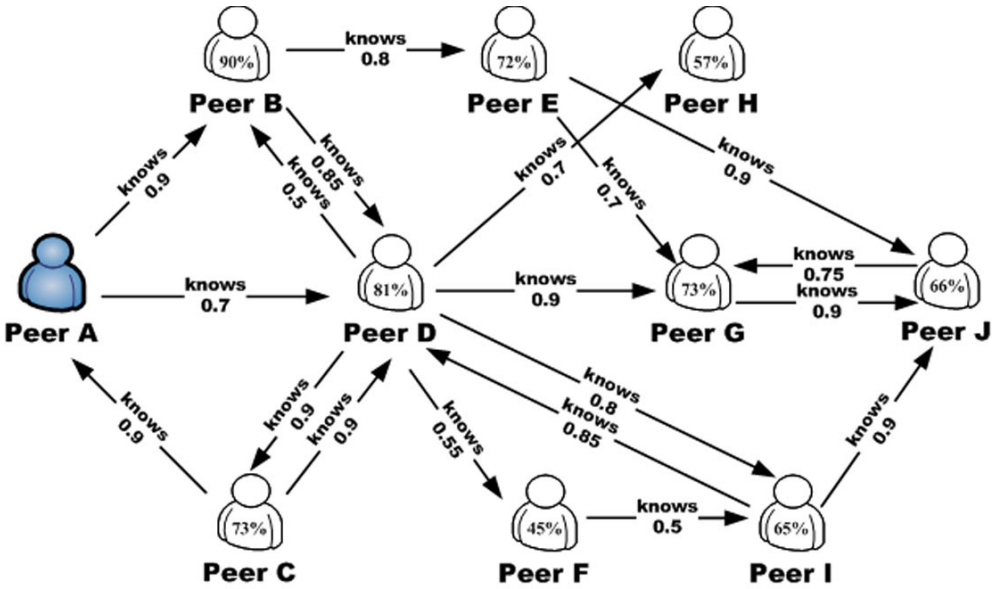


Figure 6.4: *PeerA*'s trusts in other community members

committed by itself will be finally performed in a positive way (i.e., requirements are met) and this peer could always break its commitments in a negative way. Thus, one peer's trust in another is nothing but a judgement on the latter's commitments and it does not mean those commitments will be successfully fulfilled in the end. Therefore, a peer trusting another means the former either believes the latter will always keep its commitments or believes the latter will always break its commitments.

Based on Algorithm 4, we can figure out how well each peer has known others in a community. Furthermore, we can also calculate the overall trustworthiness of a peer for other community members. Peers' trust degrees in an n-peer community ($|s| = n$), denoted by TC , can be described with the following $n \times n$ trust matrix:

$$TC = \begin{pmatrix} T_{11} & T_{12} & T_{13} & \cdots & T_{1n} \\ T_{21} & T_{22} & T_{23} & \cdots & T_{2n} \\ T_{31} & T_{32} & T_{33} & \cdots & T_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ T_{n1} & T_{n2} & T_{n3} & \cdots & T_{nn} \end{pmatrix} \tag{6.1}$$

In the above matrix, T_{ij} denotes the ith peer's degree of trust in the jth peer. Columns denote peers and rows denote degrees of trusts in those peers. Unlike Algorithm 4 in which we let $T_{ii} = 1(1 \leqslant i \leqslant n)$, when we calculate the overall trust degree of a specific peer, we will not take its trust in itself into consideration, and in the trust matrix, we let $T_{ii} = 0(1 \leqslant i \leqslant n)$. Therefore, peers' overall trust degree in an n-peer community,

denoted by $Trust_o$, can be calculated according to the following equation:

$$Trust_o = \begin{pmatrix} T_{11} & T_{12} & T_{13} & \cdots & T_{1n} \\ T_{21} & T_{22} & T_{23} & \cdots & T_{2n} \\ T_{31} & T_{32} & T_{33} & \cdots & T_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ T_{n1} & T_{n2} & T_{n3} & \cdots & T_{nn} \end{pmatrix}' \times \left.\begin{pmatrix} \frac{1}{n-1} \\ \frac{1}{n-1} \\ \frac{1}{n-1} \\ \vdots \\ \frac{1}{n-1} \end{pmatrix}\right\} n \text{ rows} \qquad (6.2)$$

Here, $Trust_o$ is an $n \times 1$ matrix and the value in the ith row denotes the overall degree of trust in the ith peer. For instance, the overall degree of trust in each peer, as shown in Figure 6.4, can be calculated according to Equation 6.2 and the final result can be found in Figure 6.5. A threshold should be selected for filtering out the peers which are not trustworthy. Suppose this threshold is set to 0.6 and *PeerG* and *PeerJ* will be selected as trustworthy peers based on Figure 6.5. Although *PeerD* has more `foaf:knows` connections (6 outbound `foaf:knowss` and 4 inbound `foaf:knowss`), other peers have relatively low degree of trust in *PeerD*. The reason for this is because actually *PeerE*, *PeerG*, *PeerH* and *PeerJ* have no path (either directly or indirectly) to *peerD*. Although *PeerD* apparently has more connections in terms of the number of outbound and inbound `foaf:knowss`, it is actually less close to other community members. Our trust-degree calculating algorithm assures that intuitively implicit trustworthy peers can be discovered from the peer community, which would otherwise be overlooked due to the ignorance to their implicit (or indirect) relationships with other peers.

6.3.4 Peer Reciprocity and Community Tolerance

One of the most important reasons for why the community exists is due to peers' autonomy and social behaviours. The peer community will grow fast when all peers or most of them are reciprocal to one another. Here, reciprocity basically means that peers can easily acquire desired services shared by other community members via interactions.

Peers are egocentric so it is inevitable that some of them are not always on-line or sticking to their commitments. Therefore, a robust community should have tolerance for either intentional or unintentional faults caused by those members. This means there should be a community norm, especially for peers which are not responsible for

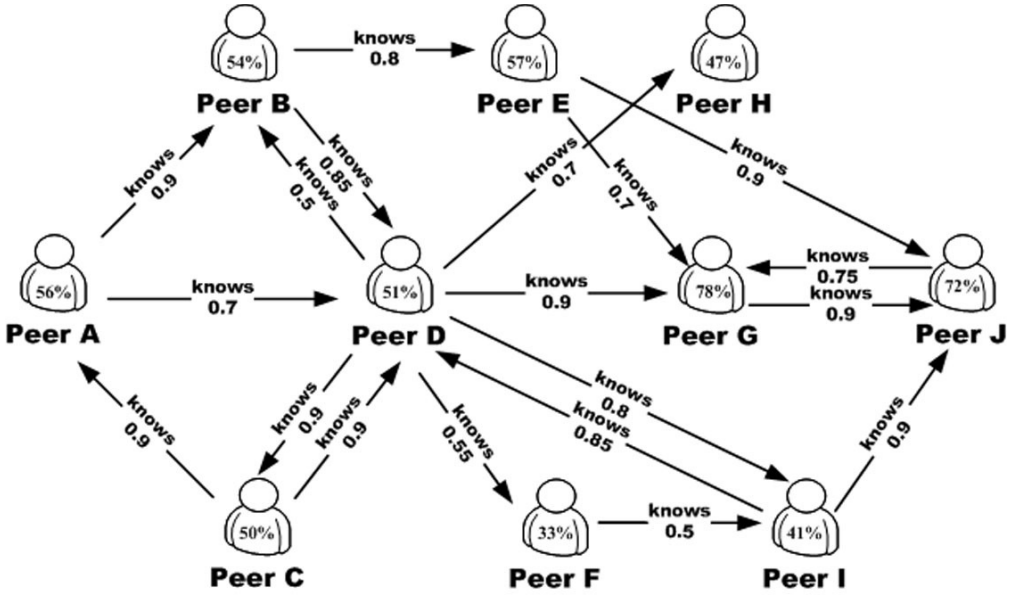


Figure 6.5: Overall trusts of community members

those faults, for dealing with peers' faults and mitigating the resulting damage on it members and our next-step research will be focused on improving fault tolerance of the peer community.

Summary

In this chapter, a generic specification on forming peer-to-peer communities is described based on approaches already discussed in previous chapters. Also, features and social effects of peer communities are described. Developers or online community maintainers may refer to this specification when building peer-to-peer communities by employing alternative techniques and at same time maximise the flexibility and the extensibility by preserving the features and social effects as discussed earlier. The next chapter will focus on experiments with our system, including effectiveness of annotation-based discovery, stress tests on distributed peers curating communities, the comparison between two message-passing methods based on non-blocking I/O and blocking I/O and IM semantic enhancement as well as several case studies in different user scenarios.

Chapter 7

Evaluation of the OKBook Architecture

This chapter provides evaluations and a usage scenario on functionalities so far provided by the system implemented based on our approaches in this thesis. Section 7.1 describes how peer groups can make service discovery effective. Section 7.2 presents stress tests on our system and shows how peer community evolvement can be achieved in terms of scalability. Section 7.3 provides evidences on that our non-blocking-I/O design is superior to the traditional methods which usually achieve concurrency by suspending threads or processes. Section 7.4 and Section 7.5 give a basic usage scenario where IM annotation/consumption and the OKBook system are demonstrated as a whole, respectively.

7.1 Effectiveness of IM Discovery Based on Peer Groups

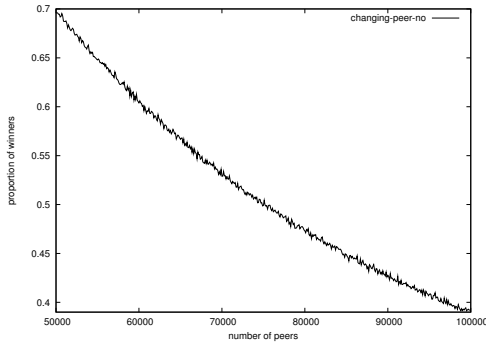
In order to showcase how Interaction Models (IMs) and collaborative peers can be discovered via peer groups, we experimented with the Interactions From An Interaction (IFAI) algorithm (described in Algorithm 1) by simulating the peer interactions in this section. Firstly, 100,000 peers and 10 IMs were generated as the test set. According to the Openfire (mentioned in Section 4.1.3.2) scalability test reported by Jive Software, 300 to more than 50,000 peers can be handled by a Sun 280R Server with two 1.2GHz UltraSPARC-III CPUs and 4 GB RAM[1], and therefore, 100,000 peers were chosen here since our machine doubled the CPU capacity and the memory of the above one. Since each interaction will basically involve at least two peers, we here assumed

[1] http://www.igniterealtime.org/about/OpenfireScalability.pdf

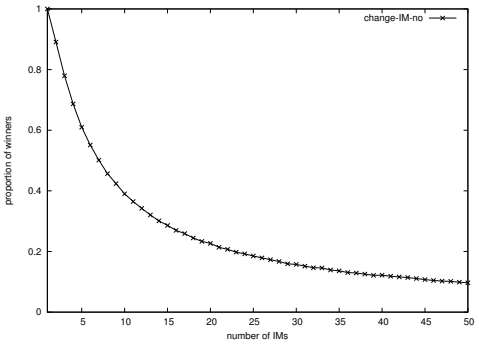
each IM owns two roles and 80,000 interactions in total take place (where the CPU usage reached 100%) over each experiment. In each interaction, one IM and two peers were randomly selected to generate the interaction instance and play the defined roles respectively. Moreover, a bidirectional link was generated to connect every pair of collaborative peers. After peers were grouped based on IFAI, we calculated how many peers can find desired IMs only making use of our peer-group-based discovery rather than the assistance of the meta-search engine on OKBook. The aim of the experiment is to measure the percentage (named as Winning Proportion (WP)) of peers that can obtain the IMs they need from their group members. Empirically, there is more chance for a peer to find interesting IMs from its groups in which it has (in)directly interacted with others before, compared with the chance to find those IMs from groups not containing this peer as their member. In order to minimise the number of cases of unforeseen situations, omissions or errors, we ran this experiment 1,000 times and found that on average, 38.99% of peers got desired IMs from their own group members without searching on the meta-search engine, which indicates that peer groups can reduce the burden on OKBook as well as other Semantic Web Search Engines (SWSEs) which have indexed republished IMs and exposed IM search services.

Secondly, since this proportion was probably related to the parameters such as the number of peers, the number of IMs and the number of interactions, in order to discover the relations, we redid the above experiment by changing one parameter (e.g., the number of peers) but keeping other two (e.g., the number of IMs and the number of interactions) fixed each time and calculating the WP in three cases. The results are depicted in Figure 7.1(a), Figure 7.1(b) and Figure 7.1(c). Figure 7.1(a) and Figure 7.1(b) indicate that for those peer communities which have a relatively large number of peers and shared IMs, it is more difficult for their members to discover desired services and collaborators from peer groups, compared with small sized peer communities. On the other hand, we can also see that even though the augmentation of the whole peer-to-peer network and the number of interactions are unpredicted (especially for popular peer communities), community members can still try to make the number of harnessed IMs as small as possible to maximise the WP as shown in Figure 7.1(b) (i.e., by limiting the interests of each peer community). Moreover, as time goes by, more and more interactions will occur in the peer-to-peer community and the WP will also go up in terms of our experimental result as shown in Figure 7.1(c).

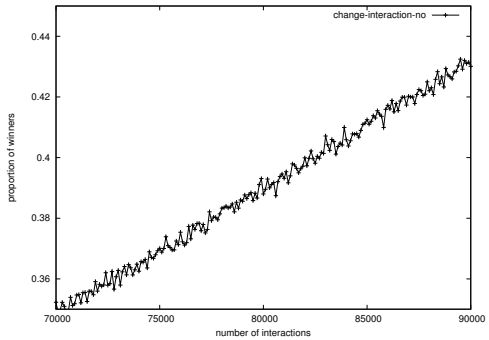
Thirdly, IMs may contain more than two roles and the WP value in reality is probably



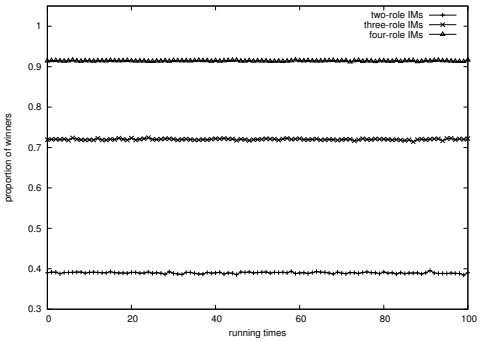
(a) WP change with peers



(b) WP change with IMs



(c) WP change with peer interactions



(d) WP change with IM roles

Figure 7.1: Experimental results of the group-based IM discovery

higher than the above two-role cases because under this circumstance, more group members and connections may come up after each interaction. Based on the previous configuration of our experiment (100,000 peers, 10 IMs and 80,000 interactions), we also investigated the performance of the peer-group-based discovery in the situations where interactions involved three peers or four peers. The discovery programme was run for 100 times (when the WP value of each case had become almost flat) in each of the two cases and the result is shown in Figure 7.1(d). For the three-role case, on average, 72% of peers got desired IMs via their own group members while for the four-role case, this proportion went up to 91.43%. Therefore, for communities with a limited number of resources (peers and IMs), if more peers (roles) are involved in each interaction, community members will gain more benefits within the discovery process.

7.2 Stress Tests on Distributed Peers Curating Communities

Scalability is a particular concern when the Web is taken as the infrastructure that underpins the peer community discussed in this thesis. In this section, an experiment was designed and completed to simulate peers as well as their interactions and do stress tests on a communication peer (which is a PC with an Intel Pentium® D 3.00GHz processor and 1 GB RAM). Another peer (which is a laptop with an Intel Core™2 Duo 2.2GHz processor and 1 GB RAM) was used as the client for constantly registering new users and passing messages. This experiment took 1,800 seconds in total to finish. During this period, in the interval of each second there was a new peer which was launched and registered on the communication peer. Here, each peer engaged in five transactions including *connection*, *registration*, *authentication*, *online_msg* and *offline_msg*. Note that in each transaction, one or more requests were sent to the communication peer. The *connection* transaction sent out the *connect* request; The *registration* transaction sent out the *register* request; The *authentication* transaction sent out three requests including *authenticate*, *password* and *initial*; The *online_msg* transaction sent out a message to a randomly selected online peer while the *offline_msg* transaction sent out a message to a randomly selected offline peer. The thinking time, which is used for simulating users' idle activity, for each user between each two transactions is set to one second. After the running of this experiment, 1,467 peers in total successfully joined the community by registering on the community server. The highest rate for the size of responses is 29.15 Kbits/sec and their total size after the experiment is 1.95 MB. Whilst, the highest rate for the size of requests is 15.73 Kbits/sec and their total size is 1.21 MB. Figure 7.2 describes both the change of membership and the change of simultaneous online members.

As shown in this figure, around the $600th$ second, the number of connected peers reached 1,009 and then exceeded the limit that the client can handle. Around the $900th$ second, the client exceeded the time of max connection retries and began to disconnect existing connections. The maximum number of simultaneous online peers are 1,146 in the end. We can see that although the performance is currently hampered by our limited but improvable computing resources, the result still indicates that the peer community can be formed incrementally based on our approach and the system can adapt to changes in peer membership via communication protocols that have been

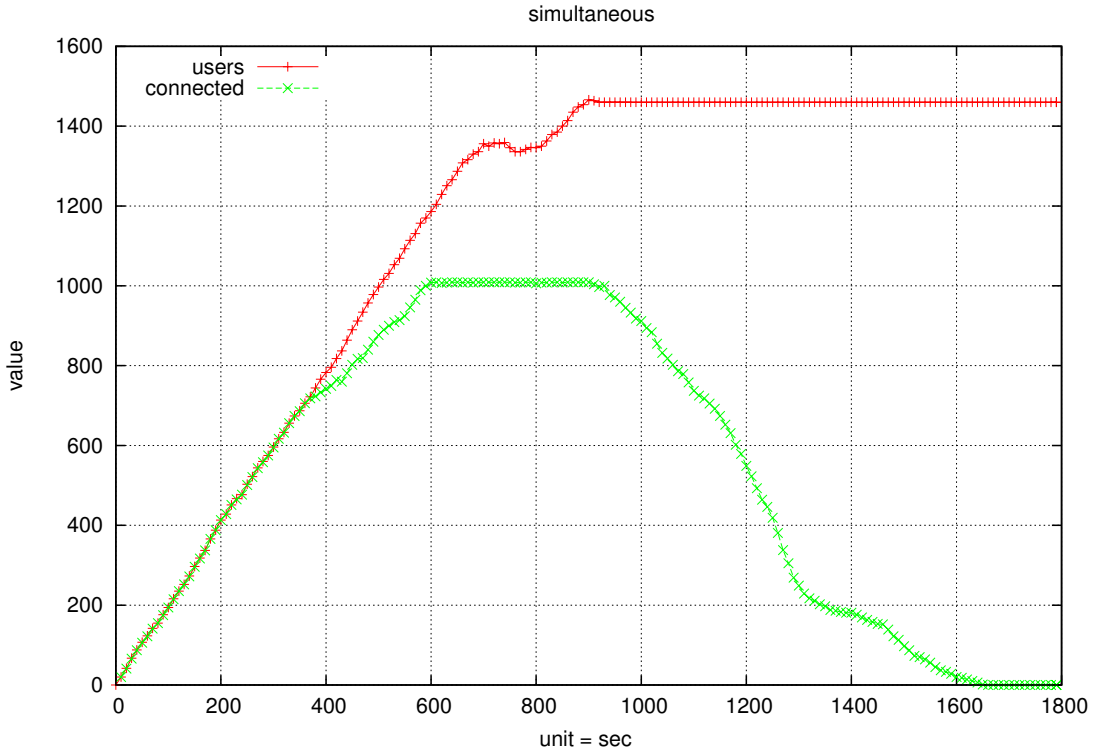


Figure 7.2: Simultaneous online members

employed here. In extreme cases, if the number of simultaneously online peers is n, there needs to be $\lceil \frac{n}{1000} \rceil$ communication peers on the Web to appropriately handle all the traffic. Note that this number was calculated in terms of the PC used in this test and may vary depending on the allocated computing resources on each communication peer.

The rates for the above five transactions are depicted in Figure 7.3, from which we can see that a large portion of time was spent on *page* (i.e., users browsing pages and corresponding thread(s) being idle) and other interaction-related transactions were finished comparatively more efficiently.

7.3 IM Execution in Browsers: Non-Blocking I/O vs Blocking I/O

eXtended Lightweight Coordination Calculus (XLCC) has extended Lightweight Coordination Calculus (LCC) by introducing a new operator to materialise the non-blocking I/O within the process of running IMs. In order to showcase the advantage of XLCC

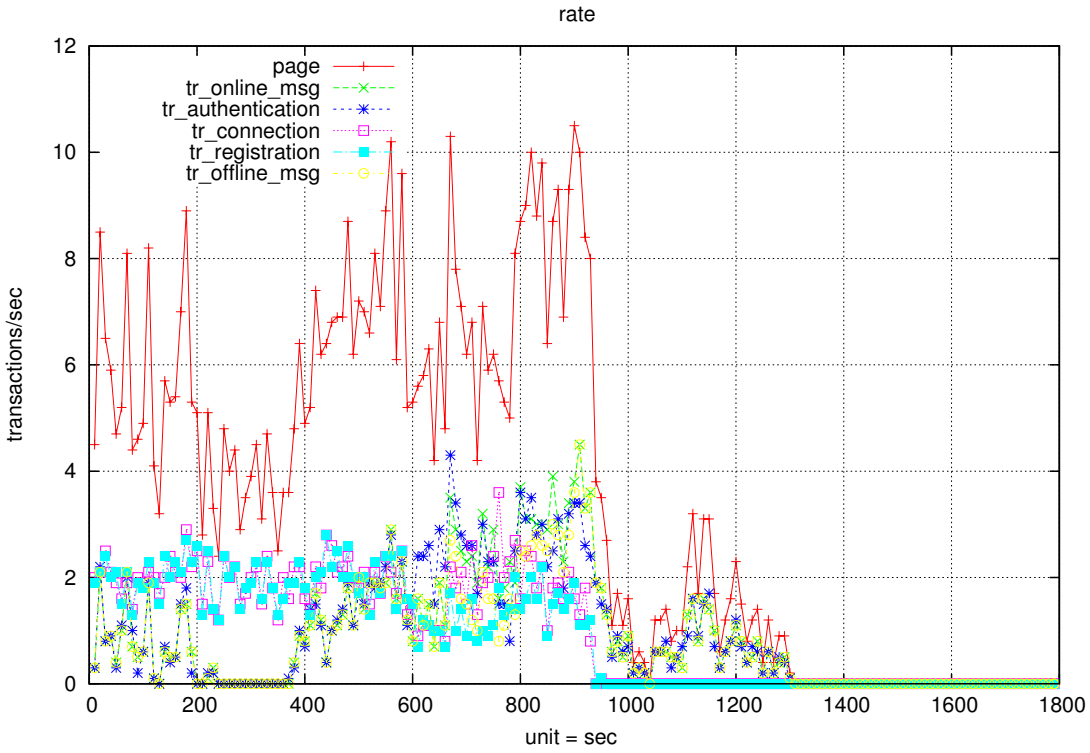


Figure 7.3: Five transactions for each peer

over the opposite blocking I/O design, our XLCC interpreter's performance on the IM execution with non-blocking I/O is compared with the performance on the I/O-blocking execution in this section. This comparison involved two peers, one of which triggers the interaction by sending a message to the other and receives responses later on. Each sending and receiving pair forms a basic request/response unit (RnR) in which message sending and message receiving should happen sequentially. Meanwhile, the operator `then` is used here to connect message sending and message receiving in each RnR and interactions, each of which containing only one unit, are not considered here since I/O will be blocked by `then` in this case. The lengths of messages which are passed during peer interactions could be different in the real world but in order to simplify the experiment, message lengths are uniform here. We conducted the experiment on the performance of the XLCC interpreter by calculating the costs of the time spent on running IMs with non-blocking I/O (RnRs are joined with `niobs`) and running IMs with blocking I/O (RnRs are joined with `thens`) respectively. After each calculation, the number of units was increased by one and the experiment was repeated. Figure 7.4 describes the result of calculations on the time costs of interactions between two peers according to the above experiment design.

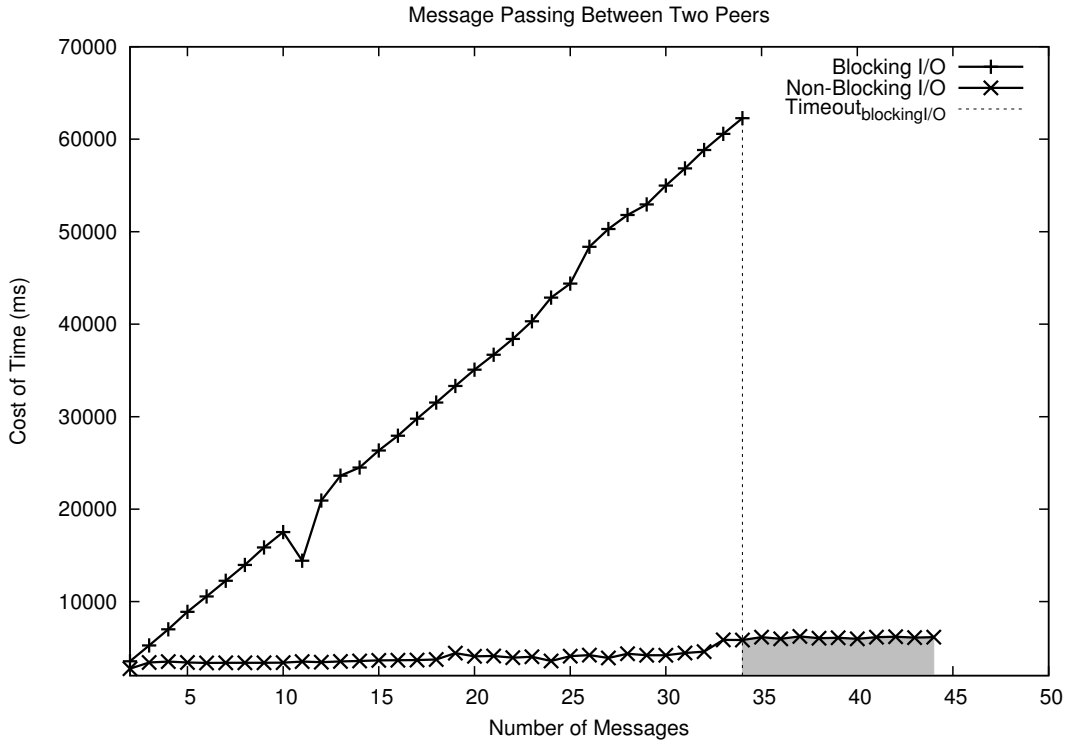


Figure 7.4: Comparison between interactions with non-blocking and blocking I/Os

From the above figure, with the increasing of the number of RnR units, there is a steep linear increase in the time cost of running IM with blocking I/O. However, for the IM executions with non-blocking I/O, time cost is almost constant. Moreover, the experiment on the running of IMs with blocking I/O stop progressing when the number of RnRs reached 35, which occurred due to the timeout of the employed Hypertext Transfer Protocol (HTTP) endpoint based on Bidirectional-streams Over Synchronous HTTP (BOSH). This could of course be improved by reconfiguring the BOSH property settings or employing another endpoint with better performance. Nevertheless, with the same timeout setting as shown in Figure 7.4, running IMs in a non-blocking-I/O manner can handle more RnR unites so our approach which supports non-blocking I/O scales to the peer-to-peer knowledge sharing environment with a large number of messages being passed around better than those approaches based on the traditional blocking-I/O manner.

Since our solution here conforms to HTML5, technically, the agent state can be persisted inside the local storage of each Web browser. As of writing of this thesis, large browser vendors including Mozilla Firefox, Google Chrome and Opera have provided 5 MB local storage space on average, and Microsoft IE has provided 10 MB for each

entire domain. Presumably, each IM has five LCC clauses, each of which has five variables and the average lengths of variable names and values are four and ten, respectively. With 5 MB local storage space, approximately 7,000 ($5,000,000 \div (5 \times 5 \times (4 \times 2 + 10 \times 2))$) interactions can be persistent for each IM. Note that this number was calculated based on the above presumption and may vary on a case-by-case basis. Nevertheless, that number of interactions is fairly enough for an agent (a peer) to persist and remember in practice at any one time and with improvement in both software and hardware, this capacity will, in any case, continue to increase.

7.4 Experiments and Case Study on IM Semantic Enhancement

This section experiments with the semantic enhancement strategy applied to IMs and gives case studies on IM annotation/consumption. We used a PC with Intel Pentium Duo 3.0*GHz* CPUs and 1 *GB* of RAM to consult the experiment. As mentioned above, one way of harnessing published IMs is to assist the Distributed Discovery Service (DDS) in discovering appropriate IMs that meet peers' requirements. In the peer-to-peer network, an atomic interaction occurs between two peers. We used two PCs as peers with the same performance, each of which has been installed with a communication server that can communicate with the other peer, and this makes each machine have both client and server capabilities. IMs owned by each peer have been published on the DDS and are also stored in its local IM repository. Suppose a peer, *P*, sends a query (note that both Uniform Resource Identifier (URI)-based queries and phrase-based queries are supported) to a DDS peer, *D*. By matching the query against with semantic annotations embedded in persisted IMs, *D* can select most relevant IMs which are likely to meet the requirement of the user who has logged on to *P* and made that query. Several matching and ranking strategies can be adopted as already discussed in Chapter 3. When a specific IM was selected and is currently being browsed by *P*, *P*'s Resource Description Framework in Attributes (RDFa) parser (e.g., this parser has been installed on *P*' as an add-on/plugin) can automatically parse the IM page into Resource Description Framework (RDF) triples. By querying these triples, *P* can check if it can play a specific role in this IM by investigating OpenKnowledge Components (OKCs) and constraints. If *P* can provide all the OKCs which this IM requires a spe-

cific role to provide, P is able to subscribe to this IM and play this role during the IM execution. On the other hand, P will also have a chance to query the above harvested triples to find out if this IM has belonged to some communities and then P can apply for a membership to join these communities at its own will. Suppose the URI for this IM is denoted by im_uri; the named graph containing P's profile is denoted by $IM_{triples}$; the URI for the graph containing harvested triples is denoted by IM. Then the following SPARQL Protocol and RDF Query Language (SPARQL) query will help the peer figure out which role it can play and which potential communities it may join:

```
SELECT ?role ?community
FROM ⟨IM⟩
FROM NAMED ⟨IM_triples⟩
WHERE {
  GRAPH ⟨IM_triples⟩ {?P a wsc:Peer. ?role a owlp:Role:
    Role. ?P wsc:plays ?role.}
  ?community a vook:P2PCommunity.
  ⟨im_uri⟩ owlp:hasRole ?role.
  ⟨im_uri⟩ vook:belongs_to ?community.
}
```

Another way of making use of published IMs occurs in their running processes. In order to analyse this usage, we created and published an IM targeting the job vacancy service on the UK Civil Service Website [2]. This IM can help people not only find out appropriate job information services but also look for desired jobs from these services. Since each job vacancy is published on a Web page with embedded semantic annotations (RDFa 1.0 on this Website), when the IM is executed, users looking for jobs will query the service and get the job pages back for further checking if they can play roles in this IM. Using the Atom feed published by this Website, we retrieved 338 pages corresponding to 338 jobs in total. Suppose each page is given a unique ID (e.g., 0 to 337) and then the time costs of page retrievals are described in Figure 7.5. Job vacancy pages from this site are based on a unified form actually forms a template by which the site administrator can create and update job information. On average, the time cost interval for retrieving job vacancy pages is [171ms, 188ms]. Then the embedded RDF triples are harvested from the above pages and the corresponding time cost for each harvest is described in Figure 7.6. In this figure, most of pages are parsed within the time cost interval [3ms, 5ms]. The first page and the last page cost 43ms and

[2] http://www.civilservice.gov.uk/jobs/

27*ms* respectively, which are extraordinarily long compared with the others and can be ignored in this experiment because much time was spent on the preparation before the loading of retrieved pages and resource disposal after the overall RDFa harvest. We checked extracted triples and found that some of them are duplicated due to the RDFa parser (still under active development at the time of writing this thesis) we used. However, duplicated triples are repeated with equal times for different pages so the duplication will not influence the performance on the comparison and can be ignored in our experiment.

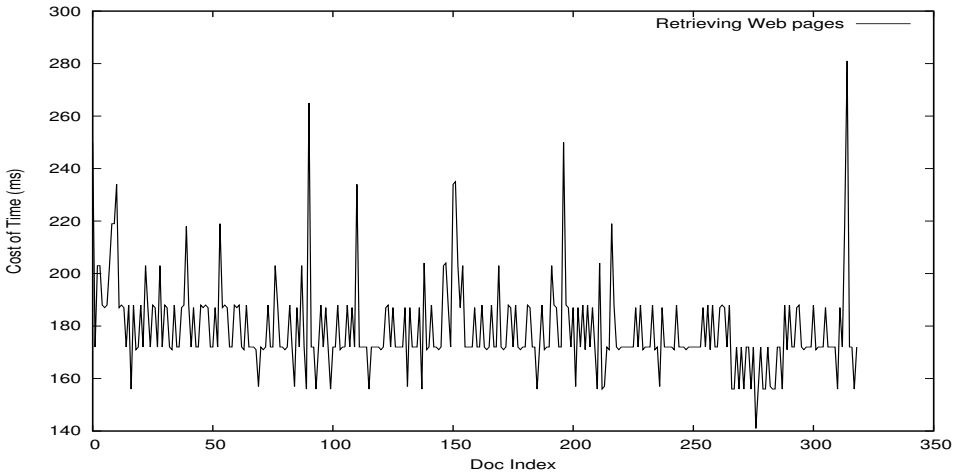


Figure 7.5: Time cost of retrieving pages

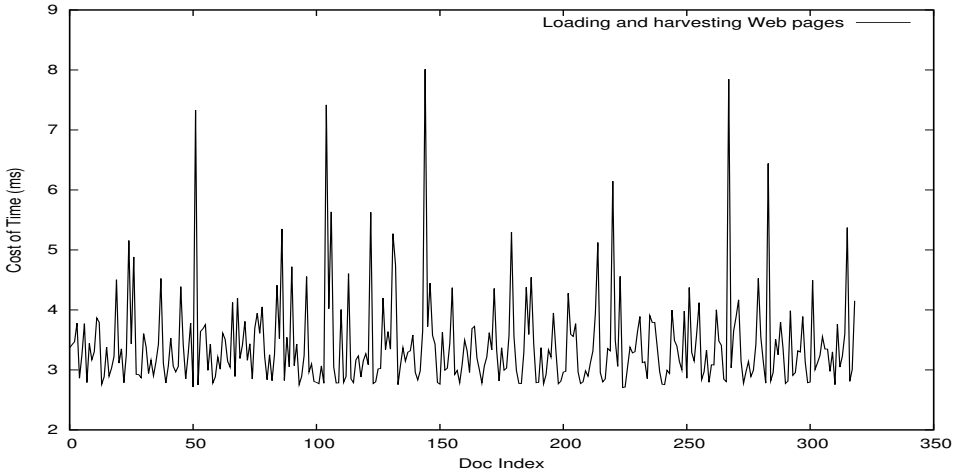


Figure 7.6: Time cost of harvesting RDFa data

By matching the user's query with semantic annotations embedded in published IMs, an IM is discovered as shown in Figure 7.7 (note that this IM is encoded with LCC

and *a(client, initial, 1, 1)* means the *client* role is the one that starts the interaction while *a(shop, necessary, 1)* means at least one peer needs to play the *shop* role during the interaction). Various peer side applications can be designed and implemented for digesting and reusing the semantically enhanced IMs in one way or another but further discussion of this is outside the scope of this thesis. Figure 7.7 depicts a peer-side consumer analyses the discovered IM and informs the peer of which roles it can play and which communities it may join in terms of its local profile.

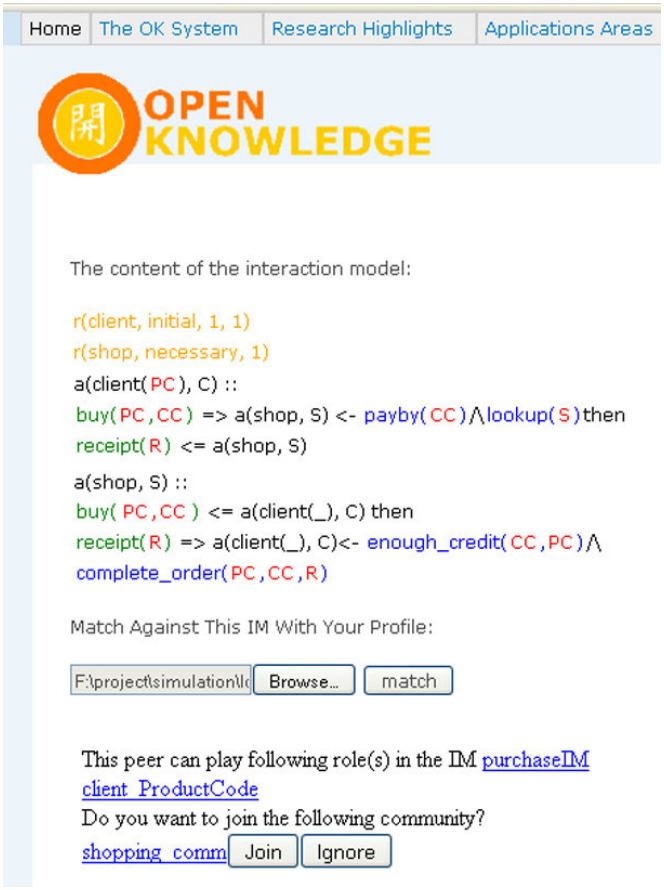


Figure 7.7: Snapshot of the user interface of a peer side consumer

Figure 7.8 gives a screenshot on a publisher publishing/annotating an IM in which a peer sends a greeting sentence to another and then the recipient displays this sentence. After being published, the Extensible HyperText Markup Language (XHTML) page of the IM will be persisted on the DDS and also stored in the publisher's local IM repository. As soon as the publisher receives acknowledgement of the successful publishing from the DDS, it will be asked whether or not to subscribe to the IM by choosing a role to play.

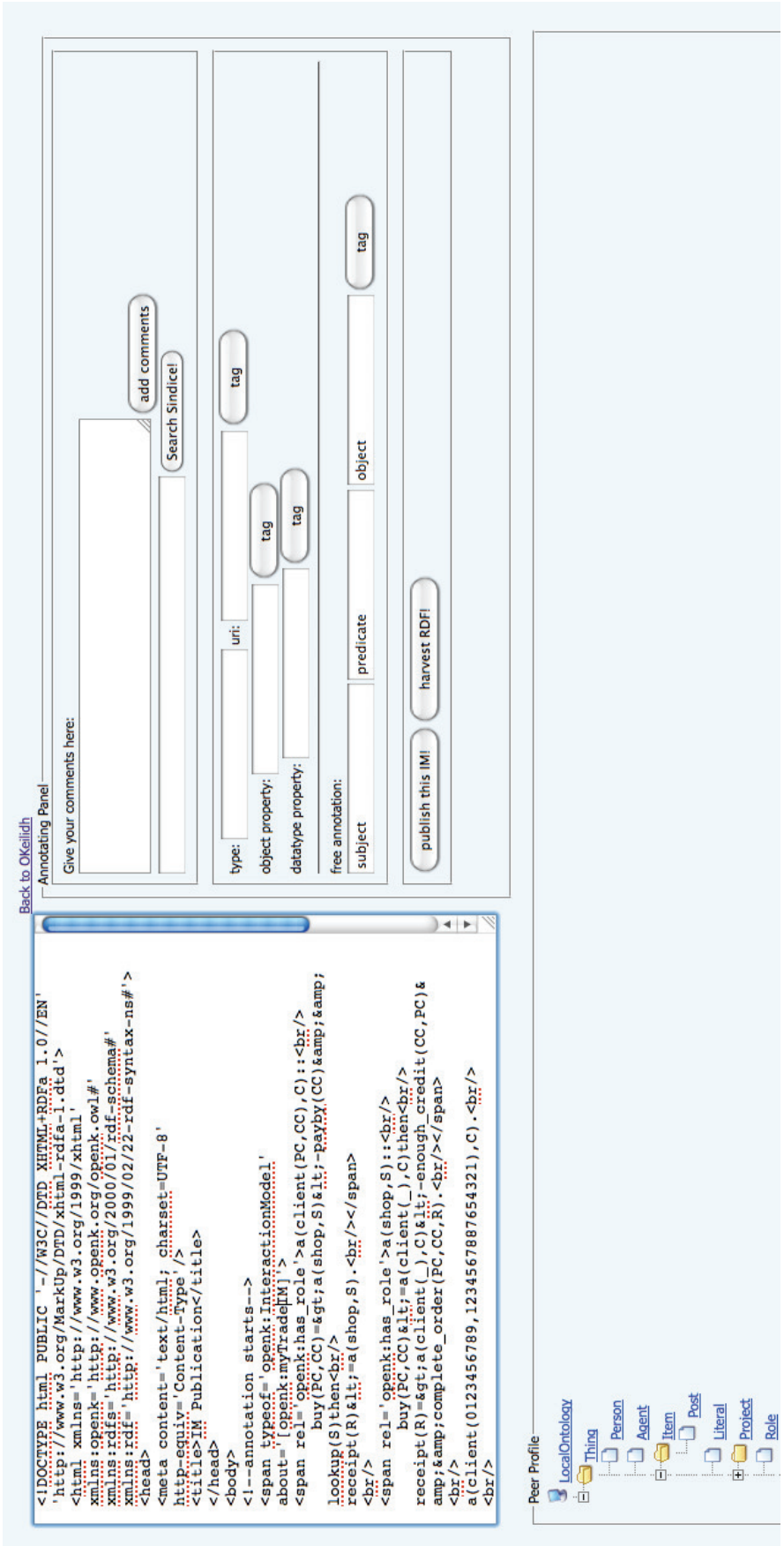


Figure 7.8: Screenshot on the user interface for annotating IMs

7.5 Usage Scenario

This section revisits the problems which have not been fully tackled due to the limitations on the traditional keyword-based search and service discovery techniques briefed in Section 1.3 and presents a usage scenarios for OKBook. Two additional case studies targeting those problems are also provided as a system walkthrough in Appendix B.

Several Social Networking Sites (SNSs) have emerged such as MySpace, Facebook and Twitter, and one of their common features is that they are all built on top of binary relations between nodes (a.k.a users) no matter if interlinks are unidirectional or bidirectional. They are based on a centralised server approach and users have to register on them in order to interact with other users. In the decentralised environment like the OpenKnowledge system, peers are more autonomous and expect more diverse social events. This requires that services provided by existing SNSs might be represented in a more generic way without any centralised server and users can interact with each other based on their own agreements. For instance, any peer that can play the role of a service provider in an IM, which defines a Facebook style of social interaction, could become a localised substitute for Facebook from the perspective of the client/server architecture depending on how constraints of this role will be satisfied.

In this thesis, the peer community is formed based on interaction logs and `foaf:knows` links between peers which are bidirectional. After an IM is successfully finished, the involved peers certainly neither have to to create `foaf:knows` links pointing to everybody nor have to create `foaf:knows` links pointing to others with which they have collaborated in that interaction. Peers are autonomous and egocentric so they can choose to know specific peers on different occasions. On the other hand, historical records of interactions especially which were successfully fulfilled can provide peers with more trustworthy collaborative peers and trigger the formation of groups of interests.

Thanks to IMs, functionalities owned by existing SNSs can be mapped to several OKCs. Suppose a peer expects to know which are its closest peers as observed by some other trusted peer and then it can subscribe to the IM described in Figure 7.9. Note that peers may have various algorithms to implement the OKC for solving the constraint $search\_neighbors(N,S)$ here. For example, users on Facebook are linked via undirected edges while nodes on Twitter are linked via directed ones. Suppose Peer A and Peer B both have registered on Facebook and Twitter respectively. Then

their relationships can be described using the following two formulas:

$$R_{facebook}(A,B) \leftrightarrow links\_to(A,B) \wedge links\_to(B,A)$$

$$R_{twitter}(A,B) \leftrightarrow links\_to(A,B) \vee links\_to(B,A)$$

Taking the peer-to-peer network as a directed social graph, discovery services from different social networks can be generalised and unified via a single IM, and any newly devised discovery algorithms can be therefore wrapped into OKCs capable of solving a corresponding constraint like $search\_neighbours(N,S)$.

/∗ An individual peer, *I*, sends out a message to a potential social network commissary, *C*, in order to retrieve *I*'s closest members. Then based on *I*'s neighbourhood information stored in *N* (a particular data model storing a graph taking peers as nodes and their relations as edges) and a specific algorithm derived from the graph theory, *C* find out *I*'s closest peers. Finally, *C* returns the discovered peer set *S* to *I*. There will be one peer playing the individual role and more than one peer playing the commissary role. ∗/

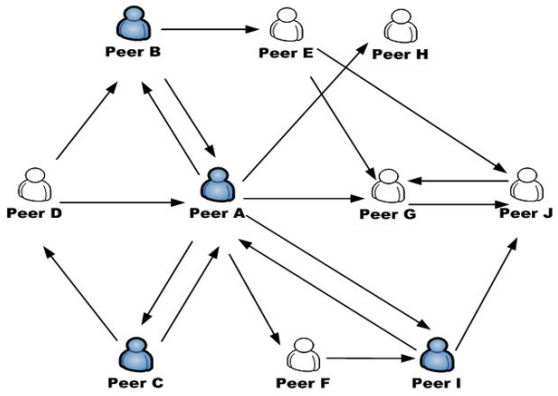
```
a(individual, I)::
   retrieve_closest_peers(N)⇒ a(commissary, C) then
   offer_peer_set(S)⇐ a(commissary, C).

a(commissary, C)::
   retrieve_closest_peers(N)⇐ a(individual, I) then
   offer_peer_set(S)⇒ a(individual, I)← not(isolate(N)) &&
                                       search_neighbours(N, S).
```

Figure 7.9: IM for retrieving the set of closest peers

As mentioned in Figure 7.9, *N* is a data model holding the information about all of a peer's neighbours. A peer could just work out the set of its closest peers by employing a particular algorithm but as mentioned above, there are a variety of algorithms for doing this job so it is necessary to separate each peer with those algorithms and wrap them into public services which can be invoked whenever needed. Arguably, peers should only send their personal data to others they trust so it is obviously more secure for them to keep their data local and perform as many tasks over those data locally as possible. On the other hand, manipulating data locally may bring lots of burdens on peers especially those which have limited computation capacities or limited accesses

to nearby computation resources. Thus, a trade-off needs to be found to balance data privacy and data sharing, which is however out of the scope of this thesis. Since the peer social graph is directed, N can be a two-columns table in which each row denotes an edge. One column denotes the direction of the edge and the other column denotes the connected peer via this edge. Based on diverse types of interactions, peers that want to play a social community commissary role can compose various IMs and employ diverse algorithms to implement OKCs for the purpose of meeting other peers' social requirements. For example, Figure 7.10 illustrates the peer A's closest peers B, C and I, which were discovered based on the breath-first search algorithm and bidirectional interlinks ($depth = 1$). This actually simulates a user's set of Facebook-like closest peers. However, if a peer wants a set of Twitter-like closest peers such as a set composed of all its followers (or followees), another OKC needs to be created based on single directional interlinks. These IMs will be also published on microdata-embedded pages, so our IM discovery service on OKBook will find the most suitable interaction(s) in which peers prefer being involved.



Edge	Peer	Edge	Peer
Incoming	B	Outgoing	F
Outgoing	B	incoming	I
Incoming	D	Outgoing	I
Outgoing	C	Outgoing	G
Incoming	C	Outgoing	F

Figure 7.10: Peer A's Facebook-like closest peer discovery ($depth = 1$)

On Facebook, users can interact with each other in several ways. The first and the most basic one is that users can post on their friends' Walls (serve as the primary asynchronous messaging boards) or their own, and probably their friends may reply to these posts later on. Users can contact with their friends via the email service and the instant messenger internally. Photo sharing is one of most important functionalities and another type of interaction is that users comment on photos uploaded by their friends or themselves. Users can also interact with others through applications such as games, online markets and blogs. Last but not least, Mini-Feeds can provide a continually refreshed list of all user events including creation of new social links, Wall posts, photo

comments and application notifications. So based on different types of interactions we can build corresponding IMs for Facebook. For example, Figure 7.11 describes an IM which helps peers to share photos with their subscribers in a peer-to-peer manner.

/∗ An individual peer, *P*, uploads a latest photo and send it to a peer community, *OKB*, in order to share the photo with other community members who are the subscribers of this peer. Then *OKB* finds all its subscribers and sends a message of wall update to each of them (including the publishing peer itself). ∗/

```
a(peer, P)::
   post(Pic)⇒ a(community, OKB)← upload(Pic) then
   update_wall(Pic, Pub)⇐ a(community, OKB).

a(community, OKB)::
   post(Pic)⇐ a(Peer, P) then
   a(community(Pic, P, S), OKB)← find_subscribers(P, S) niob
   update_wall(Pic, P)⇒ a(Peer, P).

a(community(Pic, Pub, S), OKB)::
   null← S = []
   or
   update_wall(Pic, Pub)⇒ a(peer, Sub)← S = [Sub|R] then
   a(community(Pic, Pub, R)).
```

Figure 7.11: IM for posting pictures to subscribers' walls

OKBook is dedicated to peers' interactions and it also provides functionalities of social networks but in a more interactive way. The aforementioned interactions on Facebook can be boiled down to several IMs, and the main difference between the friend graph derived from Facebook and the peer community derived from OKBook is that the former is formed by manually created links from one registered user to another and the latter is formed by automatically recorded interactions in which peers have been involved.

Summary

In this chapter, several experiments were carried out from different aspects of the OKBook system. The experimental results indicate that lightweight annotations can ben-

efit peers all through their interaction processes including discovery and data reuse, and at the same time they do not explicitly increase the payload at the consumer side. The stress tests showcase that the scalability of the system powered by OKBook can handle massive numbers of registered peers and passed messages in a decentralised environment. Blocking-I/O is sometimes not necessary and may even effect the system performance. So on that basis, our approach instead chooses non-blocking-I/O to obtain a communication layer with better efficiency. Finally, this chapter returns to the problems raised at the beginning of this thesis and gives basic case studies which describe how OKBook can tackle these problems based on the approaches and designs previously described in this thesis.

Chapter 8

Conclusions and Future Work

An approach has been proposed in this thesis to the formation of peer communities based on peer interaction protocols in the form of Interaction Models (IMs). A platform called OKBook has been created as a preliminary implementation of this approach taking OpenKnowledge as its inspiration. OKBook provides peers with a platform for publishing, discovering and (un)subscribing to IMs. Within the publishing process, linkable metadata is used for annotating elements inside IMs which will be finally published on Web pages. For the discovery process, the two mechanisms proposed are based on a meta search engine and a dynamic peer grouping algorithm.

Compared with Web Service (WS) orchestration, WS choreography provides an alternative model for representing how peers collaborate with one another in order to achieve their top-level goals. In this thesis, we also presented OKeilidh as a decentralised proof-of-concept system which encodes choreography as IMs and executes interactions within modern Web browsers. A demonstration screencast is available here[1] and more are coming up on the OKeilidh Website[2]. eXtended Lightweight Coordination Calculus (XLCC) extends Lightweight Coordination Calculus (LCC) as a lightweight and browser-focused language used for encoding choreography. In addition, we have developed a vocabulary that makes it convenient for service publishers to annotate their services and link those services to others in an interconnected manner. This in turn benefits and enriches the increasing number of resources (including services) published in conformity to Linked Data principles. In future work, we will further integrate OKeilidh into OKBook and thus make IM management and IM exe-

[1] `http://vimeo.com/user9076607/qna-with-okeilidh`

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

cution interoperate seamlessly together.

Our Web-oriented knowledge sharing platform has emerged from the *ad hoc* and peer-to-peer networks, and as discussed in this thesis, IM execution is a message-intensive process so tremendous traffic will occur, especially when a massive amount of interactions occur in a decentralised environment simultaneously. Therefore, we have introduced a new operator to materialise the non-blocking I/O within the process of running IMs, and the experimental results show that running IMs in a non-blocking-I/O manner is superior to running them in a blocking-I/O manner by being capable of handling more request/response units. So this operator design scales to the decentralised knowledge sharing environment with a large number of messages able to be passed around, and more than the traditional approaches based on the blocking-I/O manner. On the other hand, OKeilidh incorporates Extensible Messaging and Presence Protocol (XMPP) as the peer communication protocol in the communication layer. As discussed in Section 2.5, Advanced Message Queuing Protocol (AMQP) is a wire-level protocol based on Transmission Control Protocol (TCP) and various patterns can be designed and built on top of it. Therefore, considering the strengths and weaknesses of both XMPP and AMQP, we believe it is also important for OKeilidh to support AMQP in the future, and make this protocol complementary to and cooperate with XMPP in the communication layer.

Also, from the perspective of WS choreography, this thesis proposes a new approach for republishing the IMs associated with metadata in a peer-to-peer knowledge sharing environment. We add a semantic layer on top of the IM encoded in LCC by republishing it using the annotation-embedded Web page. These republished IMs can assist peers by discovering desired services and collaborative peers precisely thanks to the unambiguous Uniform Resource Identifiers (URIs) of online resources. IM publication complies with the Linked Data principals and will further contribute to and also benefit from the Web of data. Moreover, the IM publication provides a more secure and controllable way of transferring knowledge within the distributed environment. On the other hand, peers can be ranked in terms of their reputations in a community based on the algorithm discussed in (Besana et al., 2009). There is a possibility that the peer currently being targeted was subscribed to a specific IM in the past and the IM page will also inform this peer of those previous subscriptions with meta information such as how many subscriptions have been made by this peer before via which OKBook servers.

OKeilidh is currently at the stage of demonstration and further explorations into the aspect of its use in practice are necessary and can be outlined via several directions. It can be used as a distributed multi-agent system running in Web browsers and how to deal with argumentation and security problems which have been faced by existing multi-agent systems has not been thoroughly investigated in this thesis. We expect to develop OKeilidh into a full-fledged system comparable with contemporary systems (such as Jade, Jason and Jadex, etc.) though it may not have to be FIPA-complaint. Our long term goal is to create an ecosystem for generic task-driven peer collaborations/competitions. OKBook heavily relies on the Semantic Web as its underlying data/inference infrastructure and as discussed earlier, a number of tasks performed on this platform benefit from appropriately applied vocabularies/ontologies. To tackle the potential problem related to knowledge heterogeneity, this thesis introduces existing methods such as ontology mapping and light weight same-as services. However, as of writing this thesis, people from the Semantic Web community have not found a widely accepted solution to address the above problem. Therefore, how our approaches can help practitioners to eliminate the heterogeneity problem (for example, via negotiations between involved peers on a specific task) and by doing so, how our approaches can at the same time benefit from this (for example, via enriched semantics leading to better performance on IM/peer discovery) will be further explored in the near future.

Nowadays, more and more service providers begin to look for customers instead of waiting for customers to look for their services. So we believe that in the near future, we will not search for services but services will automatically find us in one way or another (e.g., through the peer community). Our approaches proposed in this thesis, accompanied with implemented proof-of-concept prototypes, are trying to achieve this goal within decentralised knowledge sharing environments such as *ad hoc* and peer-to-peer networks.

Bibliography

Aberer, K. and Despotovic, Z. (2001). Managing trust in a peer-2-peer information system. In *Proceedings of the Tenth International Conference on Information and Knowledge Management (CIKM 2001)*, pages 310–317. ACM Press.

Adida, B., Birbeck, M., McCarron, S., and Pemberton, S. (2008). RDFa in XHTML: Syntax and Processing, W3C Recommendation. `http://www.w3.org/TR/rdfa-syntax`.

Akkiraju, R. (2005). Web Service Semantics-WSDL-S (Version 1.0). `http://www.w3.org/Submission/WSDL-S/`.

Akram, A. and Allan, R. (2005). Virtual Peer Communities and the Community Coordinator. In *Proc. SKG '05*, pages 110–119.

Alexander, K., Cyganiak, R., Hausenblas, M., and Zhao, J. (2009). Describing Linked Datasets - On the Design and Usage of voiD, the 'Vocabulary of Interlinked Datasets'. In *Proc. WWW Workshop on LDOW '09*.

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., and Ives, Z. (2007). DBpedia: A Nucleus for a Web of Open Data. In *Proceedings of the 6th International Semantic Web Conference*, pages 11–15. Springer.

Auer, S., Dietzold, S., Lehmann, J., Hellmann, S., and Aumueller, D. (2009). Triplify: Light-Weight Linked Data Publication from Relational Databases. In *Proceedings of the 18th International Conference on World wide web*, pages 621–630. ACM.

Badra, F., Servant, F.-P., and Passant, A. (2011). A Semantic Web Representation of Product Range Specification based on Constraint Satisfaction Problem in the Automotive Industry. In *Proceedings of the ESWC Workshop on Ontology and Semantic Web for Manufacturing (OSEMA'11), Heraklion, Greece, 29 May, 2011*.

Bai, X. (2011). Addressing the RDFa Publishing Bottleneck. In *Proceedings of the 20th international conference companion on World wide web*, WWW '11, pages 331–336. ACM.

Bai, X., Cheng, B., and Robertson, D. (2009). Mobile Widget Sharing By Mining Peer Groups. In *Proceedings of the 1st Workshop on Inductive Reasoning and Machine Learning on the Semantic Web at ESWC 2009*. CEUR-WS.org.

Bai, X., Delbru, R., and Tummarello, G. (2008). RDF Snippets for Semantic Web Search Engines. In *Proc. OTM '07*, volume 5332, pages 1304–1318. Springer.

Bai, X., Klein, E., and Robertson, D. (2011). RDFa^2: Lightweight Semantic Enrichment for Hypertext Content. In Pan, J., Chen, H., Kim, H.-G., Li, J., Wu, Z., Horrocks, I., Mizoguchi, R., and Wu, Z., editors, *The Semantic Web*, volume 7185 of *Lecture Notes in Computer Science*, pages 318–333. Springer Berlin / Heidelberg.

Bai, X., Klein, E., and Robertson, D. (2012). Choreographing Web Services with Semantically Enhanced Scripting. In *IEEE/WIC/ACM International Conference on Web Intelligence (WI 2012 to appear)*.

Bai, X. and Robertson, D. (2010). Service Choreography Meets the Web of Data via Micro-Data. In *Proceedings of the AAAI Spring Symposium on Linked Data Meets Artificial Intelligence (LINKEDAI 2011)*, pages 8–13. AAAI Press.

Bai, X., Vasconcelos, W., and Robertson, D. (2010). OKBook: Peer-to-Peer Community Formation. In *Proceedings of the Extended Semantic Web Conference*, pages 106–120. Springer.

Barker, A., Walton, C., and Robertson, D. (2009). Choreographing Web Services. *Services Computing, IEEE Transactions on*, 2(2):152–166.

Beckett, D. and Berners-Lee, T. (2008). Turtle - Terse RDF Triple Language, W3C Team Submission. `http://www.w3.org/TeamSubmission/turtle`.

Beckett, D. and McBride, B. (2004). RDF/XML Syntax Specification (Revised), W3C Recommendation. `http://www.w3.org/TR/REC-rdf-syntax`.

Bellifemine, F., Poggi, A., and Rimassa, G. (1999). JADE - A FIPA-compliant agent framework. In *Proceedings of Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'99)*, pages 97–108.

Berners-Lee, T. (1998). Notation 3 Specification, W3C Design Issues. `http://www.w3.org/DesignIssues/Notation3.html`.

Berners-Lee, T. (2000). Semantic Web on XML. `http://www.w3.org/2000/Talks/1206-xml2k-tbl`.

Berners-Lee, T. (2006). Linked Data. `http://www.w3.org/DesignIssues/LinkedData.html`.

Berners-Lee, T., Fischetti, M., and Dertouzos, T. M. (1999). *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. HarperCollins.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The Semantic Web. *Scientific American*, 284(5):34–43.

Besana, P., Patkar, V., Barker, A., Robertson, D., and Glasspool, D. (2009). Sharing Choreographies in OpenKnowledge: A Novel Approach to Interoperability. *Journal of Software*, 4(8):833–842.

Birman, K. and Joseph, T. (1987). Exploiting Virtual Synchrony in Distributed Systems. *SIGOPS Oper. Syst. Rev.*, 21(5):123–138.

Bizer, C., Cyganiak, R., and Heath, T. (2007). How to Publish Linked Data on the Web. `http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial`.

Bradner, S. (1997). Key Words for Use in RFCs to Indicate Requirement Levels (IETF RFC 2119). `http://www.ietf.org/rfc/rfc2119`.

Brickley, D. and Miller, L. (2007). FOAF Vocabulary Specification. `http://xmlns.com/foaf/spec/`.

Busi, N., Gorrieri, R., Guidi, C., Lucchi, R., and Zavattaro, G. (2005). Choreography and Orchestration: A Synergic Approach for System Design. In *Proceedings of the 3rd International Conference of Service-Oriented Computing*, pages 228–240. Springer.

Carroll, J. J., Bizer, C., Hayes, P., and Stickler, P. (2005). Named graphs, provenance and trust. In *Proceedings of the 14th International Conference on World Wide Web*, pages 613–622. ACM.

Çelik, T. and Meyer, E. (2004). XHTML Friends Network. In *Proceedings of the ACM Hypertext*. ACM.

Cheng, G., Ge, W., and Qu, Y. (2008). Falcons: Searching and Browsing Entities on the Semantic Web. In *Proceedings of the 17th International Conference on World Wide Web*, pages 1101–1102. ACM.

Chirita, P. A., Alex, P., Chirita, R., Idreos, S., Koubarakis, M., and Nejdl, W. (2004). Publish/Subscribe for RDF-based P2P Networks. In *Proc. ESWS '04*, volume 3053, pages 182–197. Springer.

Choi, N., Song, I.-Y., and Han, H. (2006). A survey on ontology mapping. *SIGMOD Rec.*, 35(3):34–41.

Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. (2001). Web Services Description Language (WSDL) 1.1. `http://www.w3.org/TR/wsdl/`.

Cornelli, F., Damiani, E., di Vimercati, D. C., Paraboschi, S., and Samarati, P. (2002). Choosing Reputable Servents in a P2P Network. In *Proceedings of the Eleventh International Conference on World Wide Web (WWW 2002)*. ACM Press.

Crockford, D. (2006). JSON: The Fat-Free Alternative to XML. In *Proceedings of XML 2006*.

Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., and Weerawarana, S. (2002). Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI. *Internet Computing, IEEE*, 6(2):86–93.

Damiani, E., di Vimercati, D. C., Paraboschi, S., Samarati, P., and Violante, F. (2002). A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks. In *Proceedings of the 9th ACM conference on Computer and Communications Security (CCS 2002)*. ACM Press.

Davoust, A. and Esfandiari, B. (2008). Towards Semantically Enhanced Peer-to-Peer File-Sharing. In *Proc. OTM Workshops '08*, volume 5333, pages 937–946. Springer.

Davoust, A. and Esfandiari, B. (2009). Linking and Navigating Data in a P2P File-Sharing Network. In *Proc. WWW Workshop on LDOW '09*.

de Pinninck Bas, A. P., Dupplaw, D., Kotoulas, S., and Siebes, R. (2007). The Open-Knowledge Kernel. *International Journal of Applied Mathematics and Computer Sciences (IJAMCS)*, 4(3):162–167.

Dietze, S., Yu, H. Q., Pedrinaci, C., Liu, D., and Domingue, J. (2011). SmartLink: A Web-Based Editor and Search Environment for Linked Services. In *Proceedings of ESWC '01*, volume 6644, pages 436–440. Springer.

Ding, L., Finin, T., Joshi, A., Pan, R., Cost, R. S., Peng, Y., Reddivari, P., Doshi, V., and Sachs, J. (2004). Swoogle: A Search and Metadata Engine for the Semantic Web. In *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management*, pages 652–659. ACM.

Dodds, L. and Davis, I. (2010). Linked Data Patterns - A Pattern Catalogue for Modelling, Publishing, and Consuming Linked Data. `http://patterns.dataincubator.org/book`.

Farrell, J. and Lausen, H. (2007). Semantic Annotations for WSDL and XML Schema. `http://www.w3.org/TR/2007/REC-sawsdl-20070828/`.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University OF California, Irvine.

Finin, T., Fritzson, R., McKay, D., and McEntire, R. (1994). KQML as an agent communication language. In *Proceedings of the third international conference on Information and knowledge management*, pages 456–463. ACM.

Fischer, M., Purtell, T. J., and Lam, M. S. (2011). Email Clients as Decentralized Social Apps in Mr. Privacy .

Fitzpatrick, B., Slatkin, B., Atkins, M., and Genestoux, M. (2012). PubSubHubbub Core 0.4 — Working Draft. `https://superfeedr-misc.s3.amazonaws.com/pubsubhubbub-core-0.4.html`.

Grant, J., Beckett, D., and McBride, B. (2004). RDF Test Cases, W3C Recommendation. `http://www.w3.org/TR/rdf-testcases`.

Hammer-Lahav, E. (2010). The OAuth 1.0 Protocol (IETF RFC 5849). `http://tools.ietf.org/html/rfc5849`.

Hampton, K., Goulet, L. S., Rainie, L., and Purcell, K. (2011). Social networking sites and our lives. Technical report, Pew Internet and American Life Project.

Häsel, M. (2011). Opensocial: an enabler for social applications on the Web. *Commun. ACM*, 54(1):139–144.

Hendler, J. A. (2009). Tonight's Dessert: Semantic Web Layer Cakes. In *Proceedings of the 6th European Semantic Web Conference (ESWC 2009)*, volume 5554, page 1. Springer-Verlag.

He, A., Johnston, E., and Kushmerick, N. (2004). ASSAM: A Tool for Semi-Automatically Annotating Semantic Web Services. In *Proceedings of the 3rd International Semantic Web Conference (ISWC'04)*, pages 320–334. Springer-Verlag.

Hickson, I. (2012). HTML Microdata , W3C Working Draft. `http://www.w3.org/TR/microdata/`.

Idehen, K. and Erling, O. (2008). Linked Data Spaces & Data Portability.

Joachims, T. (1998). Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In *Machine Learning: ECML-98*, volume 1398, pages 137–142. Springer Berlin / Heidelberg.

Kamvar, S. D., Schlosser, M. T., and Garcia-Molina, H. (2003). The EigenTrust algorithm for reputation management in P2P networks. In *Proceedings of the 12th international conference on World Wide Web*, pages 640–651. ACM.

Kavantzas, N., Burdett, D., Ritzinger, G., and Lafon, Y. (2005). Web services Choreography Description Language Version 1.0. `http://www.w3.org/TR/ws-cdl-10/`.

Kaykova, O., Kononenko, O., Terziyan, V., and Zharko, A. (2004). Community Formation Scenarios in Peer-to-Peer Web Service Environments. In *Proc. IASTED on Databases and Applications '04*, pages 62–67. ACTA Press.

Kelsey, J., Schneier, B., and Wagner, D. (1998). Protocol Interactions and the Chosen Protocol Attack. In *Proceedings of the 5th International Workshop on Security Protocols*, pages 91–104. Springer-Verlag.

Khambatti, M., Ryu, K. D., and Dasgupta, P. (2004). Structuring Peer-to-Peer Networks Using Interest-Based Communities. In *Proc. VLDB workshop on DBISP2P '04*, volume 2944, pages 48–63. Springer.

Klusch, M., Fries, B., and Sycara, K. (2006). Semantic Web Service Selection with SAWSDL-MX. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2006)*, pages 915–922. ACM Press.

Klusch, M. and Kapahnke, P. (2008). Semantic Web Service Selection with SAWSDL-MX. In *Proceedings of the International Workshop on Service Matchmaking and*

Resource Retrieval in the Semantic Web (SMR^2 2008) at ISWC 2008, pages 3–18. CEUR-WS.org.

Kobilarov, G., Scott, T., Raimond, Y., Oliver, S., Sizemore, C., Smethurst, M., Bizer, C., and Lee, R. (2009). Media Meets Semantic Web — How the BBC Uses DBpedia and Linked Data to Make Connections. In *Proceedings of the 6th European Semantic Web Conference*, pages 723–737. Springer.

Kopecky, J., Gomadam, K., and Vitvar, T. (2009). hRESTS: An HTML Microformat for Describing RESTful Web Services. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT'08)*, pages 619–625. IEEE CS.

Kotoulas, S. and Siebes, R. (2007). Adaptive Routing in Structured Peer-to-Peer Overlays. In *Proceedings of the 3rd International IEEE Workshop on Collaborative Service-Oriented P2P Information Systems (WETICE 2007)*. IEEE Computer Society.

Kramer, J. (2009). Advanced Message Queuing Protocol (AMQP). *Linux Journal*, 2009(187):3.

Langridge, S. and Hickson, I. (2002). Pingback 1.0. Technical report.

Lara, R., Roman, D., Polleres, A., and Fensel, D. (2004). A Conceptual Comparison of WSMO and OWL-S. In *Proceedings of the European Conference on Web Service (ECOWS 2004)*, pages 254–269. Springer.

Lewis, R. (2007). Dereferencing HTTP URIs. `http://www.w3.org/2001/tag/doc/httpRange-14/HttpRange-14.html`.

Liu, K., Bhaduri, K., Das, K., Nguyen, P., and Kargupta, H. (2006). Client-Side Web Mining for Community Formation in Peer-to-Peer Environments. *ACM SIGKDD Explorations*, 8(2):11–20.

Mallya, A. U., Desai, N., Chopra, A. K., and Singh, M. P. (2005). OWL-P: OWL for Protocol and Processes. In *Proceedings of the fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, pages 139–140. ACM.

Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., and

Sycara, K. (2004). OWL-S: Semantic Markup for Web Services. W3C Member Submission. http://www.w3.org/Submission/OWL-S/.

McBride, B. (2004). RDF Primer, W3C Recommendation. http://www.w3.org/TR/2004/REC-rdf-primer-20040210.

Milner, R. (1999). *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, New York, NY, USA.

Mitra, N. and Lafon, Y. (2007). SOAP Version 1.2 Part 0: Primer (Second Edition), W3C Recommendation. http://www.w3.org/TR/2007/REC-soap12-part0-20070427/.

Moro, G., Ouksel, A. M., and Sartori, C. (2003). Agents and Peer-to-Peer Computing: A Promising Combination of Paradigms. In *Proceedings of the 1st International Conference on Agents and Peer-to-Peer Computing (AP2PC'02)*, pages 1–14. Springer-Verlag.

Nejdl, W., Siberski, W., and Sintek, M. (2003). Design Issues and Challenges for RDF- and Schema-Based Peer-to-Peer Systems. *ACM SIGMOD Record*, 32(3):41–46.

Nowack, B. (2008). SPARQL+, SPARQLScript, SPARQL Result Templates - SPARQL Extensions for the Mashup Developer. In *International Semantic Web Conference (Posters & Demos)*, volume 401. CEUR-WS.org.

O'Reilly, T. (2007). What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. *International Journal of Digital Economics*, (65):17–37.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999-66, Stanford InfoLab.

Palma, R., Haase, P., and Gómez-Pérez, A. (2005). Oyster - Sharing and Re-using Ontologies in a Peer-to-Peer Community. In *Proc. ISWC '05*, volume 3729, pages 1059–1062. Springer.

Paolucci, M., Kawamura, T., Payne, T. R., and Sycara, K. P. (2002). Semantic Matching of Web Services Capabilities. In *Proceedings of the First International Semantic Web Conference on The Semantic Web*, ISWC'02, pages 333–347. Springer-Verlag.

Papazoglou, M. P. (2003). Service-Oriented Computing: Concepts, Characteristics and

Directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003. WISE 2003*, pages 3–12. IEEE Computer Society.

Passant, A., Breslin, J., and Decker, S. (2010). Open, Distributed and Semantic Microblogging with SMOB. In *Web Engineering*, volume 6189, pages 494–497. Springer Berlin / Heidelberg.

Paterson, I., Smith, D., Saint-Andre, P., and Moffitt, J. (2010). Bidirectional-streams Over Synchronous HTTP (BOSH). `http://xmpp.org/extensions/xep-0124.html`.

Patil, A., Oundhakar, S., Sheth, A., and Verma, K. (2004). Meteor-S Web Service Annotation Framework. In *Proceedings of the 13th International Conference on the World Wide Web*, pages 553–562. ACM Press.

Pedrinaci, C. and Domingue, J. (2010). Toward the Next Wave of Services: Linked Services for the Web of Data. *J. UCS*, 16(13):1694–1719.

Peltz, C. (2003). Web Services Orchestration and Choreography. *Computer*, 36(10):46–52.

Pietriga, E., Bizer, C., Karger, D., and Lee, R. (2006). Fresnel: A Browser-Independent Presentation Vocabulary for RDF. In *Proceedings of the 5th International Semantic Web Conference (ISWC'06)*, volume 4273, pages 158–171. Springer.

Porter, C. E. (2004). A Typology of Virtual Communities: A Multi-Disciplinary Foundation for Future Research. *Journal of Computer-mediated Communication*, 10(1).

Prodromou, E., Vibber, B., Walker, J., and Copley, Z. (2010). OStatus 1.0 Draft 2. `http://ostatus.org/sites/default/files/ostatus-1.0-draft-2-specification.html`.

Recordon, D. and Hardt, D. (2012). The OAuth 2.0 Authorization Protocol (draft-ietf-oauth-v2-25). `http://tools.ietf.org/html/draft-ietf-oauth-v2-25`.

Recordon, D. and Reed, D. (2006). OpenID 2.0: A Platform for User-Centric Identity Management. In *Proceedings of the second ACM workshop on Digital Identity Management*, pages 11–16. ACM.

Reiter, R. (1991). Artificial Intelligence and Mathematical Theory of Computation. pages 359–380. Academic Press Professional, Inc.

Robertson, D. (2004). Multi-Agent Coordination As Distributed Logic Programming. In *Proceedings of the International Conference on Logic Programming*, pages 416–430. Springer.

Robertson, D. (2008). Webs of Interactions: Exploring Peer Ranking Via Simulation in OpenKnowledge. Technical Report, OpenKnowledge, University of Edinburgh.

Robertson, D., Walton, C., Barker, A., Besana, P., Chen-Burger, Y., Hassan, F., Lambert, D., Li, G., McGinnis, J., Osman, N., Bundy, A., McNeill, F., van Harmelen, F., Sierra, C., and Giunchiglia, F. (2009). Models of Interaction as A Grounding for Peer to Peer Knowledge Sharing. In *Advances in Web Semantics, vol 1*, pages 81–129. Springer.

Saint-Andre, P. (2004). Extensible Messaging and Presence Protocol (XMPP): Core (IETF RFC 3920). `http://www.ietf.org/rfc/rfc3920`.

Scott, D. (1970). Outline of a Mathematical Theory of Computation. *Fourth Annual Princeton Conference on Information Sciences and Systems*, pages 169–176.

Scott, M. L. (2009). *Programming Language Pragmatics*. Morgan Kaufmann, 3. edition.

Smarr, J. (2008). Portable Contacts 1.0 Draft C. `http://portablecontacts.net/draft-spec.html`.

Sporny, M., Inkster, T., Story, H., Harbulot, B., and Bachmann-Gmür, R. (2011). WebID 1.0 — Web Identification and Discovery, W3C Editor's Draft. `http://www.w3.org/2005/Incubator/webid/spec/`.

Suda, B. (2006). *Using Microformats*. O'Reilly press.

Tummarello, G., Delbru, R., and Oren, E. (2007). Sindice.com: Weaving the Open Linked Data. In *The Semantic Web*, volume 4825, pages 552–565. Springer Berlin / Heidelberg.

Verborgh, R., Steiner, T., Van Deursen, D., Van de Walle, R., and Gabarr Valls, J. (2011). Efficient Runtime Service Discovery and Consumption with Hyperlinked RESTdesc. In *Proceedings of the 7th International Conference on Next Generation Web Services Practices*. IEEE CS.

Videla, A. and Williams, J. J. (2012). *RabbitMQ in Action*. Manning.

Vitvar, T., Kopecký, J., Viskova, J., and Fensel, D. (2008). WSMO-Lite Annotations for Web Services. In *Proceedings of the 5th European Semantic Web Conference (ESWC'08)*, pages 674–689. Springer-Verlag.

Wang, F.-Y., Carley, K., Zeng, D., and Mao, W. (2007). Social Computing: From Social Informatics to Social Intelligence. *Intelligent Systems, IEEE*, 22(2):79–83.

Willmott, S., Vreeswijk, G., Chesevar, C., South, M., Mcginnis, J., Modgil, S., Rahwan, I., Reed, C., and Simari, G. (2006). Towards an Argument Interchange Format for Multiagent Systems. In *Proc. of the 3rd Int. Workshop on Argumentation in Multi-Agent Systems*.

Yao, W. and Julita, V. (2004). Trust-Based Community Formation in Peer-to-Peer File Sharing Networks. In *Proc. WI '04*, pages 416–430. IEEE CS.

Yeung, C. M. A., Liccardi, I., Lu, K., Seneviratne, O., and Berners-Lee, T. (2009). Decentralization: The Future of Online Social Networking. In *Proceedings of the W3C Workshop on the Future of Social Networking '09*.

Yolum, P. and Singh, M. P. (2002). Commitment Machines. In *Revised Papers from the 8th International Workshop on Intelligent Agents VIII*, ATAL '01, pages 235–247. Springer-Verlag.

Zhang, X., Cheng, G., and Qu, Y. (2007). Ontology Summarization Based on RDF Sentence Graph. In *Proc. WWW '07*, pages 707–716. ACM Press.

Zuckerberg, M. and Taylor, B. (2010). The Open Graph Protocol. `http://ogp.me`.

Appendix A

XLCC State Overview and Parse Tables

State Overview	
states[0].kernel Lookahead **$**	Production IM′ -> . IM
states[1].kernel Lookahead **$**	Production IM′ -> IM .
states[2].kernel Lookahead **$** **$**	Production IM -> Clause_List . BuiltIn_List IM -> Clause_List . **head (JSONLIST) .**
states[3].kernel Lookahead **$**	Production IM -> BuiltIn_List . Clause_List
states[4].kernel Lookahead **$** **$** **$**	Production IM -> **head** . **(JSONLIST) .** Clause_List IM -> **head** . **(JSONLIST) .** Clause_List BuiltIn_List IM -> **head** . **(JSONLIST) .** BuiltIn_List Clause_List
states[5].kernel Lookahead **plays knows iid head $** **plays knows iid head $**	Production Clause_List -> Clause . Clause_List -> Clause . Clause_List
states[6].kernel Lookahead **a $** **a $**	Production BuiltIn_List -> BuiltIn . BuiltIn_List -> BuiltIn . BuiltIn_List
states[7].kernel Lookahead **plays knows iid head a $** **plays knows iid head a $**	Production Clause -> Role . **::** Def . Clause -> Role . .
states[8].kernel Lookahead **a plays knows iid $**	Production BuiltIn -> **plays** . **(Constant , Constant) .**
states[9].kernel Lookahead **a plays knows iid $**	Production BuiltIn -> **knows** . **(Constant) .**
states[10].kernel Lookahead **a plays knows iid $**	Production BuiltIn -> **iid** . **(Constant) .**
states[11].kernel Lookahead **::** . **<- then or niob**}	Production Role -> **a** . **(** Type **,** Id **)**
states[12].kernel Lookahead **$**	Production IM -> Clause_List **head** . **(JSONLIST) .**
states[13].kernel Lookahead **$**	Production IM -> Clause_List BuiltIn_List .
states[14].kernel Lookahead **$**	Production IM -> BuiltIn_List Clause_List .
states[15].kernel Lookahead **$** **$** **$**	Production IM -> **head (. JSONLIST) .** BuiltIn_List Clause_List IM -> **head (. JSONLIST) .** Clause_List BuiltIn_List IM -> **head (. JSONLIST) .** Clause_List

states[16].kernel Lookahead **plays knows iid head $**	 Production Clause_List -> Clause Clause_List .
states[17].kernel Lookahead **a $**	 Production BuiltIn_List -> BuiltIn BuiltIn_List .
states[18].kernel Lookahead **plays knows iid head a $**	 Production Clause -> Role . .
states[19].kernel Lookahead **plays knows iid head a $**	 Production Clause -> Role **::** . Def .
states[20].kernel Lookahead **a plays knows iid $**	 Production BuiltIn -> **plays (** . **Constant , Constant)** .
states[21].kernel Lookahead **a plays knows iid $**	 Production BuiltIn -> **knows (** . **Constant)** .
states[22].kernel Lookahead **a plays knows iid $**	 Production BuiltIn -> **iid (** . **Constant)** .
states[23].kernel Lookahead **::** . **<- then or niob**}	 Production Role -> **a (** . Type **,** Id **)**
states[24].kernel Lookahead **$**	 Production IM -> Clause_List **head (** . **JSONLIST)** .
states[25].kernel Lookahead **$**	 Production IM -> **head (JSONLIST** . **)** . Clause_List
$** **$	IM -> **head (JSONLIST** . **)** . Clause_List BuiltIn_List IM -> **head (JSONLIST** . **)** . BuiltIn_List Clause_List
states[26].kernel Lookahead **plays knows iid head a $** **. then or niob** **. then or niob** **. then or niob**	 Production Clause -> Role **::** Def . . Def -> Def . **then** Def Def -> Def . **or** Def Def -> Def . **niob** Def
states[27].kernel Lookahead **. then or niob**}	 Production Def -> Interaction .
states[28].kernel Lookahead **. then or niob**}	 Production Def -> { . Def }
states[29].kernel Lookahead **. then or niob**} **. then or niob**} **. then or niob**}	 Production Interaction -> Message . **=>** Role Interaction -> Message . **=>** Role **<-** Constraint Interaction -> Message . **<=** Role
states[30].kernel Lookahead **. then or niob**} **<- && ||** **<- && ||**	 Production Interaction -> Constraint . **<-** Message **<=** Role Constraint -> Constraint . **&&** Constraint Constraint -> Constraint . **||** Constraint
states[31].kernel Lookahead **. then or niob**}	 Production Interaction -> **null** . **<-** Constraint
states[32].kernel Lookahead **. then or niob**} **. then or niob**}	 Production Interaction -> Role . Interaction -> Role . **<-** Constraint
states[33].kernel Lookahead **=> <=** **<- && ||** **<- && ||** **<- && ||** **== != > < >= =<**	 Production Message -> **Constant** . **(** Terms **)** Constraint -> **Constant** . Constraint -> **Constant** . **( )** Constraint -> **Constant** . **(** Terms **)** Id -> **Constant** .
states[34].kernel Lookahead **<- && || . then or niob**} **)**	 Production Constraint -> **not** . **(** Constraint **)**
states[35].kernel Lookahead **<- && || . then or niob**} **)** **<- && || . then or niob**} **)** **<- && || . then or niob**} **)** **<- && || . then or niob**} **)** **<- && || . then or niob**} **)** **<- && || . then or niob**} **)**	 Production Constraint -> Id . **==** Id Constraint -> Id . **!=** Id Constraint -> Id . **>** Id Constraint -> Id . **<** Id Constraint -> Id . **>=** Id Constraint -> Id . **=<** Id
states[36].kernel Lookahead **<- && || . then or niob**} **)** **== != > < >= =<**	 Production Constraint -> **Variable** . **=** Id Id -> **Variable** .
states[37].kernel Lookahead	 Production

<- && || . then or niob})	Constraint -> **list** . (Id , Id , Id)
states[38].kernel Lookahead == != > < >= =< <- && || . then or niob}) ,	Production Id -> **LIST** .
states[39].kernel Lookahead == != > < >= =< <- && || . then or niob}) ,	Production Id -> **String** .
states[40].kernel Lookahead **a plays knows iid $**	Production BuiltIn -> **plays (Constant** . , **Constant)** .
states[41].kernel Lookahead **a plays knows iid $**	Production BuiltIn -> **knows (Constant** .) .
states[42].kernel Lookahead **a plays knows iid $**	Production BuiltIn -> **iid (Constant** .) .
states[43].kernel Lookahead :: . <- then or niob}	Production Role -> **a (** Type . , Id **)**
states[44].kernel Lookahead ,	Production Type -> Term .
states[45].kernel Lookahead ,) ,)	Production Term -> **Constant** . Term -> **Constant** . (Terms)
states[46].kernel Lookahead ,)	Production Term -> **Variable** .
states[47].kernel Lookahead ,)	Production Term -> **LIST** .
states[48].kernel Lookahead ,)	Production Term -> **String** .
states[49].kernel Lookahead ,)	Production Term -> _ .
states[50].kernel Lookahead $	Production IM -> Clause_List **head (JSONLIST** . **)** .
states[51].kernel Lookahead $ $ $	Production IM -> **head (JSONLIST) .** . BuiltIn_List Clause_List IM -> **head (JSONLIST) .** . Clause_List BuiltIn_List IM -> **head (JSONLIST) .** . Clause_List
states[52].kernel Lookahead . then or niob}	Production Def -> Def **niob** . Def
states[53].kernel Lookahead . then or niob}	Production Def -> Def **or** . Def
states[54].kernel Lookahead . then or niob}	Production Def -> Def **then** . Def
states[55].kernel Lookahead **plays knows iid head a $**	Production Clause -> Role :: Def . .
states[56].kernel Lookahead . then or niob} } then or niob } then or niob } then or niob	Production Def -> { Def . } Def -> Def . **then** Def Def -> Def . **or** Def Def -> Def . **niob** Def
states[57].kernel Lookahead . then or niob}	Production Interaction -> Message <= . Role
states[58].kernel Lookahead . then or niob} . then or niob}	Production Interaction -> Message => . Role <- Constraint Interaction -> Message => . Role
states[59].kernel Lookahead <- && || . then or niob})	Production Constraint -> Constraint || . Constraint
states[60].kernel Lookahead <- && || . then or niob})	Production Constraint -> Constraint && . Constraint
states[61].kernel Lookahead . then or niob}	Production Interaction -> Constraint <- . Message <= Role
states[62].kernel Lookahead . then or niob}	Production Interaction -> **null** <- . Constraint

states[63].kernel Lookahead **. then or niob**}	 Production Interaction -> Role **<-** . Constraint
states[64].kernel Lookahead **<- && ||** **<- && ||** **=> <=**	 Production Constraint -> **Constant (** . Terms **)** Constraint -> **Constant (** . **)** Message -> **Constant (** . Terms **)**
states[65].kernel Lookahead **<- && || . then or niob**} **)**	 Production Constraint -> **not (** . Constraint **)**
states[66].kernel Lookahead **<- && || . then or niob**} **)**	 Production Constraint -> Id **=<** . Id
states[67].kernel Lookahead **<- && || . then or niob**} **)**	 Production Constraint -> Id **>=** . Id
states[68].kernel Lookahead **<- && || . then or niob**} **)**	 Production Constraint -> Id < . Id
states[69].kernel Lookahead **<- && || . then or niob**} **)**	 Production Constraint -> Id > . Id
states[70].kernel Lookahead **<- && || . then or niob**} **)**	 Production Constraint -> Id **!=** . Id
states[71].kernel Lookahead **<- && || . then or niob**} **)**	 Production Constraint -> Id **==** . Id
states[72].kernel Lookahead **<- && || . then or niob**} **)**	 Production Constraint -> **Variable =** . Id
states[73].kernel Lookahead **<- && || . then or niob**} **)**	 Production Constraint -> **list (** . Id , Id , Id **)**
states[74].kernel Lookahead **a plays knows iid $**	 Production BuiltIn -> **plays (Constant ,** . **Constant) .**
states[75].kernel Lookahead **a plays knows iid $**	 Production BuiltIn -> **knows (Constant) .** .
states[76].kernel Lookahead **a plays knows iid $**	 Production BuiltIn -> **iid (Constant) .** .
states[77].kernel Lookahead **:: . <- then or niob**}	 Production Role -> **a (** Type **,** . Id **)**
states[78].kernel Lookahead **,)**	 Production Term -> **Constant (** . Terms **)**
states[79].kernel Lookahead **$**	 Production IM -> Clause_List **head (JSONLIST) . .**
states[80].kernel Lookahead **$** **$** **$**	 Production IM -> **head (JSONLIST) . .** Clause_List IM -> **head (JSONLIST) . .** Clause_List BuiltIn_List IM -> **head (JSONLIST) . .** BuiltIn_List Clause_List
states[81].kernel Lookahead **. then or niob**} **. then or niob**} **. then or niob**} **. then or niob**}	 Production Def -> Def **niob** Def . Def -> Def . **then** Def Def -> Def . **or** Def Def -> Def . **niob** Def
states[82].kernel Lookahead **. then or niob**} **. then or niob**} **. then or niob**} **. then or niob**}	 Production Def -> Def **or** Def . Def -> Def . **then** Def Def -> Def . **or** Def Def -> Def . **niob** Def
states[83].kernel Lookahead **. then or niob**} **. then or niob**} **. then or niob**} **. then or niob**}	 Production Def -> Def **then** Def . Def -> Def . **then** Def Def -> Def . **or** Def Def -> Def . **niob** Def
states[84].kernel Lookahead **. then or niob**}	 Production Def -> { Def } .
states[85].kernel Lookahead **. then or niob**}	 Production Interaction -> Message **<=** Role .
states[86].kernel Lookahead **. then or niob**}	 Production Interaction -> Message **=>** Role .

. then or niob}	Interaction -> Message **=>** Role . **<-** Constraint
states[87].kernel Lookahead **<- && || . then or niob})** **<- && || . then or niob})** **<- && || . then or niob})**	 Production Constraint -> Constraint **||** Constraint . Constraint -> Constraint . **&&** Constraint Constraint -> Constraint . **||** Constraint
states[88].kernel Lookahead **<- && || . then or niob})** **<- && || . then or niob})** **<- && || . then or niob})** **== != > < >= =<**	 Production Constraint -> **Constant** . Constraint -> **Constant** . **( )** Constraint -> **Constant** . **(** Terms **)** Id -> **Constant** .
states[89].kernel Lookahead **<- && || . then or niob})** **<- && || . then or niob})** **<- && || . then or niob})**	 Production Constraint -> Constraint **&&** Constraint . Constraint -> Constraint . **&&** Constraint Constraint -> Constraint . **||** Constraint
states[90].kernel Lookahead **. then or niob}**	 Production Interaction -> Constraint **<-** Message . **<=** Role
states[91].kernel Lookahead **<=**	 Production Message -> **Constant** . **(** Terms **)**
states[92].kernel Lookahead **. then or niob}** **. then or niob} && ||** **. then or niob} && ||**	 Production Interaction -> **null** **<-** Constraint . Constraint -> Constraint . **&&** Constraint Constraint -> Constraint . **||** Constraint
states[93].kernel Lookahead **. then or niob}** **. then or niob} && ||** **. then or niob} && ||**	 Production Interaction -> Role **<-** Constraint . Constraint -> Constraint . **&&** Constraint Constraint -> Constraint . **||** Constraint
states[94].kernel Lookahead **=> <=** **<- && ||** **) ,**	 Production Message -> **Constant (** Terms . **)** Constraint -> **Constant (** Terms . **)** Terms -> Terms . **,** Term
states[95].kernel Lookahead **<- && || . then or niob})**	 Production Constraint -> **Constant ( )** .
states[96].kernel Lookahead **) ,**	 Production Terms -> Term .
states[97].kernel Lookahead **<- && || . then or niob})** **) && ||** **) && ||**	 Production Constraint -> **not (** Constraint . **)** Constraint -> Constraint . **&&** Constraint Constraint -> Constraint . **||** Constraint
states[98].kernel Lookahead **<- && || . then or niob})**	 Production Constraint -> Id **=<** Id .
states[99].kernel Lookahead **<- && || . then or niob}) ,**	 Production Id -> **Constant** .
states[100].kernel Lookahead **<- && || . then or niob}) ,**	 Production Id -> **Variable** .
states[101].kernel Lookahead **<- && || . then or niob})**	 Production Constraint -> Id **>=** Id .
states[102].kernel Lookahead **<- && || . then or niob})**	 Production Constraint -> Id < Id .
states[103].kernel Lookahead **<- && || . then or niob})**	 Production Constraint -> Id > Id .
states[104].kernel Lookahead **<- && || . then or niob})**	 Production Constraint -> Id **!=** Id .
states[105].kernel Lookahead **<- && || . then or niob})**	 Production Constraint -> Id **==** Id .
states[106].kernel Lookahead **<- && || . then or niob})**	 Production Constraint -> **Variable =** Id .
states[107].kernel Lookahead **<- && || . then or niob})**	 Production Constraint -> **list (** Id . **,** Id **,** Id **)**
states[108].kernel Lookahead **a plays knows iid $**	 Production BuiltIn -> **plays (Constant , Constant .)** .
states[109].kernel Lookahead **a plays knows iid $**	 Production BuiltIn -> **knows (Constant)** . .
states[110].kernel	

Lookahead	Production
a plays knows iid $	BuiltIn -> **iid (Constant) .** .

states[111].kernel Lookahead	
:: . <- then or niob}	Production Role -> **a (** Type , Id . **)**

states[112].kernel Lookahead	
,) **) ,**	Production Term -> **Constant (** Terms . **)** Terms -> Terms . , Term

states[113].kernel Lookahead	
$	Production IM -> Clause_List **head (JSONLIST)** . .

states[114].kernel Lookahead	
$	Production IM -> **head (JSONLIST)** . BuiltIn_List . Clause_List

states[115].kernel Lookahead	
$** **$	Production IM -> **head (JSONLIST)** . Clause_List . BuiltIn_List IM -> **head (JSONLIST)** . Clause_List .

states[116].kernel Lookahead	
. then or niob}	Production Interaction -> Message **=>** Role **<-** . Constraint

states[117].kernel Lookahead	
<- && || . then or niob} **)** **<- && || . then or niob**} **)**	Production Constraint -> **Constant (** . Terms **)** Constraint -> **Constant (** . **)**

states[118].kernel Lookahead	
. then or niob}	Production Interaction -> Constraint **<-** Message **<=** . Role

states[119].kernel Lookahead	
<=	Production Message -> **Constant (** . Terms **)**

states[120].kernel Lookahead	
) ,	Production Terms -> Terms , . Term

states[121].kernel Lookahead	
<- && || **=> <=**	Production Constraint -> **Constant (** Terms **)** . Message -> **Constant (** Terms **)** .

states[122].kernel Lookahead	
<- && || . then or niob} **)**	Production Constraint -> **not (** Constraint **)** .

states[123].kernel Lookahead	
<- && || . then or niob} **)**	Production Constraint -> **list (** Id , . Id , Id **)**

states[124].kernel Lookahead	
a plays knows iid $	Production BuiltIn -> **plays (Constant , Constant)** . .

states[125].kernel Lookahead	
:: . <- then or niob}	Production Role -> **a (** Type , Id **)** .

states[126].kernel Lookahead	
,)	Production Term -> **Constant (** Terms **)** .

states[127].kernel Lookahead	
$	Production IM -> **head (JSONLIST)** . BuiltIn_List Clause_List .

states[128].kernel Lookahead	
$	Production IM -> **head (JSONLIST)** . Clause_List BuiltIn_List .

states[129].kernel Lookahead	
. then or niob} **. then or niob**} **&& ||** **. then or niob**} **&& ||**	Production Interaction -> Message **=>** Role **<-** Constraint . Constraint -> Constraint . **&&** Constraint Constraint -> Constraint . **||** Constraint

states[130].kernel Lookahead	
<- && || . then or niob} **)** **) ,**	Production Constraint -> **Constant (** Terms . **)** Terms -> Terms . , Term

states[131].kernel Lookahead	
. then or niob}	Production Interaction -> Constraint **<-** Message **<=** Role .

states[132].kernel Lookahead	
<= **) ,**	Production Message -> **Constant (** Terms . **)** Terms -> Terms . , Term

states[133].kernel Lookahead	
) ,	Production Terms -> Terms , Term .

states[134].kernel Lookahead	
<- && || . then or niob} **)**	Production Constraint -> **list (** Id , Id . , Id **)**

states[135].kernel Lookahead	
a plays knows iid $	Production BuiltIn -> **plays (Constant , Constant)** . .

states[136].kernel Lookahead <**- && || . then or niob**} **)**	 Production Constraint -> **Constant (** Terms **) .**
states[137].kernel Lookahead <**=**	 Production Message -> **Constant (** Terms **) .**
states[138].kernel Lookahead <**- && || . then or niob**} **)**	 Production Constraint -> **list (** Id , Id , . Id **)**
states[139].kernel Lookahead <**- && || . then or niob**} **)**	 Production Constraint -> **list (** Id , Id , Id . **)**
states[140].kernel Lookahead <**- && || . then or niob**} **)**	 Production Constraint -> **list (** Id , Id , Id **) .**

Pop Table	
Left-hand side	Number of symbols to pop
0	1
44	2
44	2
44	6
44	6
44	7
44	7
42	1
42	2
45	4
45	2
43	1
43	2
48	7
48	5
48	5
46	6
49	1
47	1
47	3
47	3
47	3
47	3
52	3
52	5
52	3
52	5
52	3
52	1
52	3
54	1
54	3
54	4
54	4
54	3
54	3
54	3
54	3
54	3
54	3
54	3
54	3
54	8
55	3
55	1
51	1
51	1
51	1
51	4
51	1
51	1
50	1
50	1
50	1
50	1
53	4

Action Table

State	WHITESPACE	head	a	or	then	niob	knows	plays	iid	list	null	not	::	,	.	(	)	[	]	{	}	==	!=	>	<	>=	=<	=>	<=	->	<-	=	_	|	Variable	Constant	JSONLIST	LIST	String	||	&&	$
0		s4	s11				s9	s8	s10																																		
1																																										r0
2		s12					s9	s8	s10																																		
3			s11																																							
4																s15																										
5		r7	s11				r7	r7	r7																																	r7
6			r11				s9	s8	s10																																		r11
7													s19		s18																											
8																s20																										
9																s21																										
10																s22																										
11																s23																										
12																s24																										
13																																									r1
14																																									r2
15																																					s25					
16		r8					r8	r8	r8																																	r8
17			r12																																							r12
18		r10	r10				r10	r10	r10												s28																					r10
19			s11							s37	s31	s34									s28														s36		s33			s38	s39			
20																																					s40							
21																																					s41							
22																																					s42							
23																																	s49		s46		s45			s47	s48			
24																																						s50						
25																	s51																										
26				s53	s54	s52									s55																												
27				r18	r18	r18									r18						r18																						
28			s11							s37	s31	s34									s28														s36		s33			s38	s39			
29																												s58	s57														
30																															s61											s59	s60	
31																															s62												
32				r28	r28	r28									r28						r28											s63											
33																s64						r52	r52	r52	r52	r52	r52	r52			r30									r30	r30	
34																s65																										
35																						s71	s70	s69	s68	s67	s66															
36																						r53	r53	r53	r53	r53	r53					s72										
37																s73																										
38				r54	r54	r54								r54	r54		r54				r54	r54	r54	r54	r54	r54	r54	r54			r54									r54	r54	
39				r55	r55	r55								r55	r55		r55				r55	r55	r55	r55	r55	r55	r55	r55			r55									r55	r55	
40														s74																												
41																	s75																										
42																	s76																										
43														s77																												
44														r17																												
45														r46		s78	r46																										
46														r47			r47																										
47														r48			r48																										
48														r50			r50																										
49														r51			r51																										
50																	s79																										
51															s80																											
52			s11							s37	s31	s34									s28														s36		s33			s38	s39			
53			s11							s37	s31	s34									s28														s36		s33			s38	s39			
54			s11							s37	s31	s34									s28														s36		s33			s38	s39			

55		r9	r9				r9	r9	r9																																	r9
56				s53	s54	s52															s84																					
57			s11																																							
58			s11																																							
59										s37		s34																							s36	s88		s38	s39			
60										s37		s34																							s36	s88		s38	s39			
61																																				s91						
62										s37		s34																							s36	s88		s38	s39			
63										s37		s34																							s36	s88		s38	s39			
64																	s95																s49		s46	s45		s47	s48			
65										s37		s34																							s36	s88		s38	s39			
66																																			s100	s99		s38	s39			
67																																			s100	s99		s38	s39			
68																																			s100	s99		s38	s39			
69																																			s100	s99		s38	s39			
70																																			s100	s99		s38	s39			
71																																			s100	s99		s38	s39			
72																																			s100	s99		s38	s39			
73																																			s100	s99		s38	s39			
74																																				s108						
75															s109																											
76															s110																											
77																																			s100	s99		s38	s39			
78																																	s49		s46	s45		s47	s48			
79															s113																											
80			s11				s9	s8	s10																																	
81				s53	s54	s52									r21						r21																					
82				s53	s54	r20									r20						r20																					
83				r19	s54	r19									r19						r19																					
84				r22	r22	r22									r22						r22																					
85				r25	r25	r25									r25						r25																					
86				r23	r23	r23									r23						r23										s116											
87				r42	r42	r42									r42		r42				r42										r42									s59	s60	
88				r30	r30	r30									r30	s117	r30				r30	r52	r52	r52	r52	r52	r52				r30									r30	r30	
89				r41	r41	r41									r41		r41				r41										r41									r41	s60	
90																													s118													
91																s119																										
92				r27	r27	r27									r27						r27																			s59	s60	
93				r29	r29	r29									r29						r29																			s59	s60	
94														s120			s121																									
95				r31	r31	r31									r31		r31				r31										r31									r31	r31	
96														r45			r45																									
97																	s122																							s59	s60	
98				r39	r39	r39									r39		r39				r39										r39									r39	r39	
99				r52	r52	r52								r52	r52		r52				r52										r52									r52	r52	
100				r53	r53	r53								r53	r53		r53				r53										r53									r53	r53	
101				r38	r38	r38									r38		r38				r38										r38									r38	r38	
102				r37	r37	r37									r37		r37				r37										r37									r37	r37	
103				r36	r36	r36									r36		r36				r36										r36									r36	r36	
104				r35	r35	r35									r35		r35				r35										r35									r35	r35	
105				r34	r34	r34									r34		r34				r34										r34									r34	r34	
106				r40	r40	r40									r40		r40				r40										r40									r40	r40	
107														s123																												
108																	s124																									
109			r14				r14	r14	r14																																	r14
110			r15				r15	r15	r15																																	r15
111																	s125																									
112														s120			s126																									

113																																										r4
114			s11																																							
115							s9	s8	s10																																	r3
116										s37		s34																							s36	s88		s38	s39			
117																	s95																s49		s46	s45		s47	s48			
118			s11																																							
119																																	s49		s46	s45		s47	s48			
120																																	s49		s46	s45		s47	s48			
121																												r56	r56		r32									r32	r32	
122				r33	r33	r33									r33		r33				r33										r33									r33	r33	
123																																			s100	s99		s38	s39			
124															s135																											
125				r16	r16	r16							r16		r16						r16										r16											
126														r49			r49																									
127																																										r6
128																																										r5
129				r24	r24	r24									r24						r24																			s59	s60	
130														s120			s136																									
131				r26	r26	r26									r26						r26																					
132														s120			s137																									
133														r44			r44																									
134														s138																												
135			r13				r13	r13	r13																																	r13
136				r32	r32	r32									r32		r32				r32										r32									r32	r32	
137																													r56													
138																																			s100	s99		s38	s39			
139																	s140																									
140				r43	r43	r43									r43		r43				r43										r43									r43	r43	

Goto Table

State	IM′	Clause_List	BuiltIn_List	IM	Clause	Role	Def	BuiltIn	Type	Id	Term	Interaction	Message	Constraint	Terms
0		2	3	1	5	7		6							
1															
2			13					6							
3		14			5	7									
4															
5		16			5	7									
6			17					6							
7															
8															
9															
10															
11															
12															
13															
14															
15															
16															
17															
18															
19						32	26			35		27	29	30	
20															
21															
22															
23									43		44				
24															
25															
26															
27															
28						32	56			35		27	29	30	
29															
30															
31															
32															
33															
34															
35															
36															
37															
38															
39															
40															
41															
42															
43															
44															
45															
46															
47															
48															
49															
50															
51															
52						32	81			35		27	29	30	
53						32	82			35		27	29	30	
54						32	83			35		27	29	30	
55															
56															
57						85									
58						86									
59										35				87	
60										35				89	
61													90		
62										35				92	
63										35				93	
64											96				94
65										35				97	
66										98					
67										101					
68										102					
69										103					
70										104					
71										105					
72										106					
73										107					
74															
75															
76															
77										111					
78											96				112
79															
80		115	114		5	7		6							
81															
82															
83															
84															
85															
86															
87															
88															
89															
90															
91															

92																
93																
94																
95																
96																
97																
98																
99																
100																
101																
102																
103																
104																
105																
106																
107																
108																
109																
110																
111																
112																
113																
114		127			5	7										
115			128					6								
116										35				129		
117											96				130	
118						131										
119											96				132	
120											133					
121																
122																
123										134						
124																
125																
126																
127																
128																
129																
130																
131																
132																
133																
134																
135																
136																
137																
138										139						
139																
140																

Appendix B

Case Studies on OKBook

Case Study I

OKBook provides users with not only interactions in which they were involved but also new interactions in which their collaborators were involved. The latter kind of interactions can form seeds for expansion of social groups via which users are likely to interact with more users whom they are difficult if not impossible to know only based on the searching functionality offered by Web sites like eBay. Here, let's revisit the discovery problem briefed in Section 1.3. Suppose with the help of OKBook, Alice bought a product from Bob via the trade IM described in Figure 3.8. This IM depicts an interaction in which a client purchases a product referenced by a product code from a shop catalogue using his or her credit card. Suppose Bob is a retailer who bought the same product from another peer Carol (the original manufacturer of this product) using cash via another IM in the past. By logging on to OKBook, Alice may reach and subscribe to this IM via the automatically discovered peer groups. Therefore, when Alice intends to buy the same product next time, she has a chance to interact with other peers such as Carol via those newly discovered IM instead of with Bob via the former one and Alice will be likely to get a lower price this time. Meanwhile, from Carol's perspective, OKBook has assisted her in discovering a new customer. Once their interaction is finished, Alice's group will be enlarged by absorbing a new group member (Alice) as well as a new IM. The group-driven interaction was summarised in a sequence diagram previously detailed in Figure 3.9.

Case Study II

Conventionally, when a user accesses to shopping Web sites such as eBay, he/she searches a desired product by typing in relevant keywords via the front-end search User Interface (UI). This is based on the precondition that providers have already registered on this Web site and published adverts for that product on the server. So it is difficult if not impossible for those Web sites to find services providers who had not yet registered based on customers' requirements. On the other hand, keyword-based search has its natural limitations caused by the synonymity and the ambiguity of phrases. Taken as annotations, URIs can provide less ambiguous identifiers to concepts that convey meanings users want search interfaces to be truly knowledgeable about. In the peer-to-peer network, peers are more autonomous and there is no central server having an overall control in this distributed environment with the help from the OKBook platform, all a user has to do is search for an appropriate IM (recommended by OKBook) and subscribe to it no matter if collaborative peers exist or not at that time. Then OKBook will try to automatically find other peers who can collaborate with this user and fulfil the interaction afterwards. Even if there is no peer yet providing the expected product for the time being, the subscription of this user will be still valid for a period of time (each subscription has an expiry time). But for most conventional Web sites, this temporarily "no result" is likely to end up with a page giving a Hypertext Transfer Protocol (HTTP) 404 error. As soon as enough collaborative peers subscribe to an IM, this user will be informed that this IM goes into the execution process. In Figure B.1, the sequence diagram illustrates on-line shopping cases on eBay and OKBook described based on the above analysis. Here, if the requester searches for a product P on eBay which actually does not have a suitable service provider logged on itself, then the service provider group $S = \phi$. At the same time, the interaction finishes without the requester obtaining the desired product. However, on OKBook, if the requester subscribes to a specific IM and there is no decent service provider for the time being, then as mentioned earlier, the subscription will still be valid for a period of time. When one or more sellers shows up and the requester's subscription is still valid by chance, the IM along with the subscription information will be sent to both the requester peer and the provider peer for running.

On the other hand, the annotating strategy employed by OKBook can improve the user experience on the peer side. Metadata-embedded Web pages provide a hybrid way

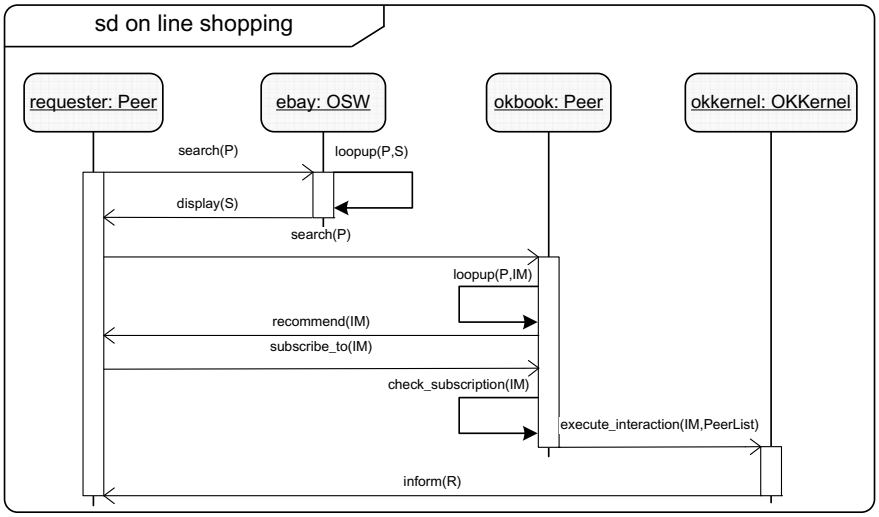


Figure B.1: Sequence diagram for on-line shopping using eBay and OKBook

of publishing human-readable data and machine-readable data via the same medium. Therefore, when users are browsing this kind of Web pages, peer-side applications can harvest and further digest the metadata and after that, some information of users' interests may be digger out. For example, suppose all items on eBay are published with metadata, if a user does a keyword search for a specific item, the peer-side application can display those appropriate items (e.g., clothes fitting into the user's own gender) at the top of the result list by analysing the user's profile against with the search result originally returned by eBay. Or, when a user looks at a page about a restaurant and based on the embedded location information (e.g., latitude/longitude or postcode), the browser will automatically calculate how long it is going to take for him/her to drive/walk to this restaurant. The above tasks are difficult for conventional server-side applications to perform due to their lack of ability to consume data carried by either embedded annotations or user profiles stored in distribute.

Glossary

communication ID A peer's *communication ID* is an email-address-like Jabber ID that contains a peer's XMPP account and this XMPP server's domain name, which are separated by @. Literally, a peer's community ID and communication ID can be the same when the registered server is installed with both OKBook and XMPP so as to be a community and communication peer.

communication peer A *communication peer* is a server which is installed with XMPP. It is recommended that the community peers are also installed with XMPP can become a communication peer at the same time.

community ID A peer's *community ID* is an email-address-like ID that contains a peer's OKBook account and this OKBook server's domain name, which are separated by @.

community peer A *community peer* is a server which is installed with OKBook and curate the group data.

hub A *hub* is a server which curates subscribers' callback functions and subscribed peers registered on itself (e.g., in hash tables). When a peer publishes/updates its profile or an IM and pings a hub, the hub will push the updated content to all of its subscribers.

interaction ID An*interaction ID* is a unique identifier generated by a particular peer community which trigger this interaction.

LCC clause A clause is a self-contained definition of a role, with message passing being the only means of transferring information between roles.

message A *message* is a piece of content which is wrapped in a specific format and passed from one peer to another via a particular socket in the peer-to-peer network.

peer A *peer* is a computation network node (e.g., a PC or a mobile terminal, etc.) which is able to delegate users to interact with other peers and access local resources or external resources. Most users have their own peers dedicated to work for themselves.

peer group A *group* is a set of peers which share common interests.

peer profile A *peer profile* accommodates its owners public information such as which roles it can play and also proprietary information such as how it can solve particular constraints and corresponding methods wrapped in OpenKnowledge Components.

publisher A *publisher* is a peer who publishes both its profile and IMs on the peer-to-peer network.

PubSubHubbub An open protocol for distributed publish/subscribe communication on the Internet.

subscriber A *subscriber* is a community peer which has subscribed to its members' profiles and certain IMs.

super peer A *super peer* is either a community peer or a communication peer which is always online, changes less frequently and provides trustworthy services continuously.

user A *user* is a client who is a service provider or a service stakeholder who can play a specific role defined in an IM or is configured to access certain peers which delegate users to play specific roles defined in IMs. A user is also a peer.

Acronyms

(X)HTML HTML or XHTML. 38, 98, 105, 108–110, 113–115, 117, 118

ACL Agent Communication Language. 23

AMQP Advanced Message Queuing Protocol. 25, 26, 154

AMS Agent Management System. 22, 23

AP Agent Platform. 23

API Application Programming Interface. 18, 24, 26, 41

BBS Bulletin Board Service. 10, 12

BNF Backus-Naur Form. 72

BNode Blank Node. 117

BOSH Bidirectional-streams Over Synchronous HTTP. 71, 72, 89, 141

CMS Content Management System. 13, 32

CRUD Create, Read, Update and Delete. 124

CSP Constraint Satisfaction Problems. 96, 98, 99, 102

CSV Comma-Separated Value. 38

CURIE Compact URI. 117

DDS Distributed Discovery Service. 45, 106, 142, 145

DF Directory Facilitator. 22, 23

EAP Event-based Asynchronous Pattern. 81

EOGP Extended Open Graph Protocol. 48–51

eRDF Embedded RDF. 19, 98

FIPA Foundation for Intelligent Physical Agents. 22, 23

FOAF Friend of a Friend. 32, 42, 102, 109, 111, 113, 115, 116

GUI Graphical User Interface. 16, 17

hRESTS HTML for RESTful Service. 18

HTML HyperText Markup Language. 19, 116

HTTP Hypertext Transfer Protocol. 2, 19, 25, 34, 39, 59, 62, 67, 70–72, 104, 108, 120–122, 125, 141, 184

IFAI Interactions From An Interaction. 43, 44, 135

IM Interaction Model. iii, 2–5, 12, 13, 16, 17, 19, 20, 27, 29–32, 35, 36, 38–52, 54–59, 62, 63, 65–74, 76–78, 80–84, 86–89, 91, 93, 94, 96, 98–109, 118, 120–128, 133, 135–137, 139–145, 147–150, 153–155, 183, 184

IMAP Internet Message Access Protocol. 25

JID Jabber ID. 62, 76, 78

JSON JavaScript Object Notation. 23, 72, 73, 77, 78

KB Knowledge Base. 51, 103, 104, 106

KQML Knowledge Query and Manipulation Language. 23

LCC Lightweight Coordination Calculus. 2, 3, 5, 16, 22, 29, 44, 63, 68, 69, 71–73, 76, 77, 81–83, 85, 86, 91, 93, 94, 96, 98, 101, 103, 104, 108, 121, 139, 153, 154

LCCI LCC Interpreter. 67–70, 76, 77, 84, 87, 91, 121, 123

LOD Linking Open Data. 7, 117

MathML Mathematical Markup Language. 99

MSM Minimal Service Model. 19

MTS Message Transport Service. 22, 23

MWSAF METEOR-S Web Service Annotation Framework. 18

NAT Network address translation. 67

NS Name Space. 110, 113, 114

ODS OpenLink Data Space. 26

OGP Open Graph Protocol. 49

OKC OpenKnowledge Component. 16, 32, 36, 68, 73, 76, 81, 86, 89, 96, 103, 123–125, 142, 147–149

OPENK OpenKnowledge in the peer community. 101

OWL Web Ontology Language. 96

OWL-P OWL for Processes and Protocols. 96, 98, 102, 107

OWLS-MX OWL-S Matchmaker. 18

QName Qualified Name. 114

RDF Resource Description Framework. 8, 11, 19–21, 32–34, 38–41, 51, 56, 93, 98, 99, 104–118, 122, 142

RDFa Resource Description Framework in Attributes. 7, 19, 20, 30, 31, 34, 38, 41, 49, 98, 105, 106, 108, 109, 113–115, 117, 118, 122, 142–144

REST REpresentational State Transfer. 18, 61, 122, 123, 125

RPC Remote Procedure Call. 41

RSS RDF Site Summary or Really Simple Syndication. 36

SA-REST Semantic Annotations for REST. 18

SASL Simple Authentication and Security Layer. 80

SAWSDL Semantic Annotations for WSDL and XML Schema. 18

SAWSDL-MX SAWSDL Matchmaker. 18

SMOB Semantic MicroBlogging. 26

SMS Short Message Service. 24

SMTP Simple Mail Transfer Protocol. 25

SNS Social Networking Site. 10, 12, 15, 24–27, 119, 126, 147

SOAP Simple Object Access Protocol. 122, 123, 125

SPARQL SPARQL Protocol and RDF Query Language. 7, 31, 33, 39, 51–56, 70, 104, 116, 121, 122, 125, 143

SRV Service Record. 72

STOMP Simple (or Streaming) Text Orientated Messaging Protocol. 25

SW Semantic Web. 17, 20

SWS Semantic Web Service. 7, 18

SWSE Semantic Web Search Engine. 30, 39, 41, 42, 46, 136

TCP Transmission Control Protocol. 2, 25, 67, 70–72, 122, 154

TCP/IP Transmission Control Protocol/Internet Protocol. 70, 121

TLS Transport Layer Security. 80

UI User Interface. 5, 6, 13, 59, 184

URI Uniform Resource Identifier. 3, 7, 8, 18–20, 26, 30, 31, 38–42, 44, 49, 50, 59, 61, 62, 74, 104–108, 111, 113, 116–118, 123–125, 142, 143, 154, 184

URL Uniform Resource Locator. 36, 38, 41, 46, 61, 78, 86, 87, 114

voiD Vocabulary of Interlinked Datasets. 117

VOOK Vocabulary Of OpenKnowledge. 102, 103

WP Winning Proportion. 136, 137

WS Web Service. 4, 7, 10, 15, 17–19, 27, 31, 87, 93, 108, 109, 122, 125, 153, 154

WS-CDL Web Services Choreography Description Language. 18

WSCAIM Web Service Choreography As Interaction Models. 96, 101–103, 108

WSDL Web Services Description Language. 17

WSMO Web Service Modeling Ontology. 18, 109

WWW World Wide Web. 1, 6, 10–12, 15, 27, 38

WYSIWYG What-You-See-Is-What-You-Get. 115

XCI ID cross-Community Interaction ID. 75, 76

XFN XHTML Friends Network. 32

XHTML Extensible HyperText Markup Language. 19, 145

XLCC eXtended Lightweight Coordination Calculus. 2, 72, 77, 83, 85–87, 89, 94, 108, 139, 140, 153

XMPP Extensible Messaging and Presence Protocol. 2, 23, 25, 26, 66–68, 70–72, 77, 80, 120, 122, 154

XPath XML Path Language. 116