INFORMATICA, 2014 Vol. x, No. y, 1–22

EVENT-BASED AWARENESS SERVICES FOR P2P GROUPWARE SYSTEMS

Alex POULOVASSILIS*, Fatos XHAFA†, Thomas O'HAGAN*

*London Knowledge Lab, Birkbeck, University of London, UK
E-mail: ap@dcs.bbk.ac.uk,trohaga@outlook.com
†Universitat Politécnica de Catalunya, Spain
E-mail: fatos@lsi.upc.edu

Abstract. P2P systems enable decentralised applications for supporting collaborating groups and communities, where the collaboration may involve both sharing of data and sharing of group processes among group members. In such applications, monitoring and awareness are critical functionalities required for an effective collaboration. However, to date there has been little research into providing generic, application-independent awareness in P2P groupware systems. We present a distributed event-based awareness approach for such systems that provides different forms of awareness through a set of interoperating, low-level awareness services. The user and technical requirements for the approach are motivated with reference to Project-Based Learning in a P2P environment. We describe the implementation of a superpeer P2P network on a Cloud platform and the provision of reliable awareness services (AaaS — *Awareness as a Service*) from the Cloud. We report on the outcomes of an empirical evaluation of the performance and scalability of the approach.

Key words: P2P Systems, Collaboration, Awareness, Services, Cloud Computing, Awareness as a Service (AaaS).

1. Introduction. Most classical educational and learning models have been teacher-centred: they have been built around the teacher as the main actor and leader while students play the role of followers and secondary actors. Due to the long tradition, experience and educational institutions' inertia, the teacher-centric approach also became the basis for online learning and teaching platforms developed using web technologies. However, the teacher-centric approach has been questioned over time as it does not fully support active learning and students' engagement. Alternative models such as *Project-Based Learning* (PBL) have been proposed to overcome its limitations and to give a more active role to students (Zumbach *et al.*, 2003). In the PBL model, the roles of teacher and student are inverted: the student is a primary actor and PBL can be considered a student-centric approach.

The emergence of new computational paradigms and technologies has, on the one hand, prompted questions such as which technology is best suited for implementing which learning model and, on the other hand, is inspiring the development of new learning models and the enhancement of existing ones, based on features of

the computational paradigms. One such computational paradigm is the *Peer-to-Peer* (P2P) one. The P2P paradigm has several features that have the potential to benefit computer-supported online learning processes:

Symmetry of network nodes: From a computational viewpoint, P2P network nodes are symmetric, i.e. there is no distinction between them. This feature is relevant for supporting learning communities, where participants may be at the same time learners and teachers.

Direct communication: In P2P systems, peers can communicate directly with each other, without server mediation, and network barriers (e.g. NATs and firewalls) can be overcome. This can allow peers to establish direct connections at any time, either synchronously or asynchronously, fostering peer interaction through sharing of knowledge and resources and provision of mutual support. P2P systems can also facilitate transitions between synchronous to asynchronous communication.

Information sharing: P2P systems enable the sharing of many kinds of information. Each peer can maintain its own repository and can also share its content with other peers. This can increase the availability of documents, especially when replication techniques are supported by the system, allowing multiple copies of the same document to be stored at different network sites. This feature can also be helpful to peers with more limited resources; for example, a mobile peer having limited storage capacity can use a repository hosted at another peer or can run software installed at other peers.

Group and community building: P2P networks can grow naturally with the addition of new peers and can enable grouping of peers, thus supporting the building of learner groups and communities. Social networking features can be readily implemented in P2P systems and are particularly relevant for scaffolding and emotional support in online learning processes.

Context: In P2P networks, context can be defined along several dimensions, for example context of individual peers, context of peer-groups, context of workspaces, and context of resources. Capturing and exploiting information about context can be very useful for learning purposes.

Ubiquity: P2P networks can support not only fixed peers (desktop or wired computers) but also mobile peers. This makes possible the support of mobile learning, with its premise of "anytime, anywhere" learning.

While P2P systems can support learning in general, in this paper we are concerned with their use for collaborative group work. P2P technologies can potentially provide more support for collaboration than centralised approaches since group members can interact directly with their peers in order to provide additional scaffolding and social support (You and Pekkola, 2001, Bentley *et al.*, 1995). We consider in this paper requirements and approaches for endowing P2P systems with *awareness* mechanisms in order to fully support groups working collaboratively on common projects. In its basic form, awareness refers to the system's ability to notify the members of a group of changes occurring in the group's

workspace. More generally, awareness refers to knowledge provided by the system to group members about the current and past actions and status of other group members (Gutwin *et al.*, 1996). Provision of awareness enables more efficient information and knowledge sharing among group members, and more timely support and decision making.

The paper is organised as follows. We review previous work on the use of P2P technologies for supporting group collaboration in Section 2. In Section 3 we analyse the user and technical requirements relating to building P2P systems to support PBL and we identify some major types of group awareness relevant to this setting. We describe in Section 4 a structured P2P network model for meeting these requirements. In Section 5 we present a set of awareness services and show how these interoperate over the P2P network to provide the various types of awareness. In Section 6 we give an overview of a proof-of-concept implementation of our approach. In Section 7 we report on the results of an empirical investigation of performance and scalability. We give our concluding remarks and directions of future work in Section 8. This paper extends (Poulovassilis and Xhafa, 2013) to give more detailed analysis of the user requirements for building P2P networks to support collaborative group work, and how a set of low-level awareness services interoperate to meet these requirements. We also describe here an implementation of our approach and the results of an empirical evaluation of its performance and scalability.

2. Related work. Several works have discussed the benefits of using P2P technologies to support learning communities. Two key benefits are sharing of learning resources and support for collaboration (Nejdl *et al.*, 2002, Simon *et al.*, 2003, Bulkowski *et al.*, 2006). The work in (Papamarkos *et al.*, 2006) explores the use of Event-Condition-Action (ECA) rules to provide the change propagation and notification services required by a distributed learning community. The work in (Jin *et al.*, 2004, Fakas *et al.*, 2004) examines how P2P technologies may be useful for enabling e-learning environments to be more efficient, scalable and versatile. Other work identifies support for group cognition processes (Dou and Wang, 2004) and integration of P2P technologies with personal knowledge management (Berman and Annexstein, 2003).

Approaches using P2P technologies to support collaborating groups include (Busseta and Merzi, 2003, Rossi and Busetta, 2004), which provide agent-based, pervasive support for group work; (Menchaca-Mendez *et al.*, 2004), which describes a JXTA-based P2P system supporting opportunistic collaboration in editing of shared documents; (Parker *et al.*, 2005), which presents a P2P framework allowing applications to collaborate over a JXTA-based architecture; (Xhafa *et al.*, 2010) which supports management of collaborating peer groups using JXTA, focussing on group monitoring, autonomy, confidentiality and security; and (Kurmanowytsch *et al.*, 2003) which presents a P2P middleware providing services for mobile teamwork.

The two main modes of collaboration in distributed environments are *asynchronous* and *synchronous* (Preguia *et al.*, 2005, Qu and Nejdl, 2011). Examples of the former are interaction through asynchronous discussion forums, and asynchronous document updates made in shared workspaces. Examples of the latter are synchronous interactions in chat, audio and video-conference systems, and synchronous document updates made in shared workspaces. With asynchronous collaboration, the interaction spans longer periods of time and information relating to the collaboration is not immediate. In contrast, synchronous collaboration enables immediate knowledge about the ongoing collaboration and strongly-coupled interaction between group members.

The above mentioned proposals are concerned mainly with asynchronous collaboration and generally provide simple awareness mechanisms, e.g. through messages and alerts. P2P-based applications targeting synchronous collaboration include (Kawashima and Ma, 2004, Li *et al.*, 2004, Ma *et al.*, 2004, Margaritis *et al.*, 2004, Oasis *et al.*, 2006). However, these do not consider the provision of awareness information to group members. In our work, we aim to support not only notifications about the activities of group members but also group processes so as to enhance the group's abilities to carry out a common group project effectively and efficiently. Moreover, we develop a generic set of awareness services provided within P2P middleware, on top of which groupware applications can build specific awareness functionalities.

Finally, the work in (Fenkam *et al.*, 2002) proposes a publish-subscribe approach to event notification and awareness in P2P collaborative systems. However, this does not support all the forms of awareness provision that we have identified. Recent service-oriented approaches address interoperability, groupware features and *QoS* (Chan *et al.*, 2007, Galatopoullos *et al.*, 2008, Mason and Ellis, 2010). However, these provide publish/subscribe models of awareness and do not support a full-featured awareness model.

3. Motivating setting and requirements. Our motivating setting is *Project-Based Learning* (PBL) (Zumbach *et al.*, 2003), although our approach is intended to apply more generally to project-based collaboration in other sectors beyond education, such as business, science and healthcare. In PBL, learning is seen as a process-oriented activity through which learners build their knowledge by solving problems, accomplishing tasks and projects. PBL has been implemented not only in traditional face-to-face teaching but also in online university curricula. The Open University of Catalonia[1] (OUC) is an example where distributed PBL is used to achieve learning goals. One such module is Software Development Techniques, which is part of Software Engineering degree programme at OUC. In this module, the students begin by forming themselves into groups and then engage in the

[1] http://www.uoc.edu/portal/en/index.html

development of a specified distributed software development project. Examples of tasks to be undertaken within the project are:

- Undertaking a critical analysis of alternative software development techniques and technologies to be used in the software development project.
- Deciding on the design approach and the architectural model.
- Configuring the software development tools to be used for the project.
- Implementing the components of the software system being developed and carrying out unit testing.
- Integrating the components of the system, and carrying out integration testing and regression testing.
- Software quality assessment of the software developed.
- Deployment of the system developed in a real networking infrastructure.

Considering the above setting, a number of general user requirements emerge for building P2P networks to support collaborative group work:

R1: Definition of a project as a workflow of tasks and precedence relationships between these relating to their order of completion.
R2: Deciding on, and revising as necessary, task deadlines.
R3: Assigning, or re-assigning, a task to one or more group members.
R4: Checking if inter-task dependencies are being met.
R5: Discovering and assigning the resources needed for accomplishing a task.
R6: Updating the status of each task, moving through different states such as "pending assignment", "assigned", "in progress", "delayed", "waiting for another task to be completed", "feedback needed", "completed".
R7: Tracking of task completion.
R8: Tracking of progress with respect to the project workflow.
R9: Tracking the availability of group members to participate in discussions and undertake assigned tasks.

These user requirements imply the need for several different types of awareness, listed below (we refer the reader to (Xhafa and Poulovassilis, 2010) for a general discussion of awareness requirements in P2P groupware systems and to (Gutwin *et al.*, 1996, You and Pekkola, 2001) in the context of web-based groupware systems):

Activity awareness provides information about the progress of the group on the accomplishment of project tasks. It comprises knowing about actions taken by members of the group according to the tasks assigned to them. As part of activity awareness, we also consider information about group members' actions on artefacts created by the group. (Relates to requirements R2, R3, R6, R7)

Process awareness provides group members with information about their progress with respect to the project workflow — both individually and as a group. It enables the identification of past, current and next states of the project workflow, with the aim of supporting the group in moving the project forward. (Relates to requirements R1, R4, R8)

Communication awareness relates to message exchange, and information about the creation and usage of synchronous and asynchronous discussion forums. It allows peers to establish links with each other, share ideas, provide feedback on each other's work, and conduct negotiations about task assignment, deliverables and deadlines. (Relates to requirements R1, R2, R3)

Context awareness provides information about the time, location and environment in which group members perform actions, and the profiles and preferences of group members. (Relates to requirements R2, R3, R5, R9)

Availability awareness provides information about the availability of group members and of their resources for project-related work. The former is necessary for establishing synchronous communication. The latter is useful for supporting group members' requirements for specific resources in undertaking tasks assigned to them (e.g. availability of a machine for running a program). (Relates to requirements R5, R9)

A number of technical requirements also arise for providing generic awareness services in this P2P setting (we refer the reader to (Xhafa and Poulovassilis, 2010) for a general discussion of technical requirements for awareness provision in P2P groupware systems):

Dynamic P2P network conditions: awareness provision needs to be able to handle peers joining and leaving the system at any time. Distribution and replication of information is needed in order to ensure provision of awareness under dynamic conditions. Replication increases processing efficiency and improves the availability of information; both full and partial replication of objects is possible. Updates made to a local copy of an object need to be propagated and applied to all its replicas; this needs to be undertaken under the dynamic conditions of the P2P network using reliable message passing mechanisms.

Genericity of events, e.g. using an XML or RDF-based representation, since we are aiming at application-independent awareness services.

Lightweight mechanisms, to reduce the overhead of event notification and processing, especially for peers with limited computational resources.

Multiple granularities of awareness presentation, allowing awareness provision to be adapted to group members' short, mid and long-term objectives. For instance, for a latecomer to the group or a peer who has been disconnected for some time, it is more useful to receive a summary of the group's activities rather than a fully detailed report.

Multiple modes of awareness delivery, allowing awareness information to be delivered either 'passive' or 'active' mode. The former does not require any specific actions by group members while the latter allows group members to request specific awareness information.

4. Computational model. We now present our computational model for provision of awareness in P2P groupware systems, to the level of detail necessary for this paper (we refer the interested reader to (Poulovassilis and Xhafa, 2013)

for further details). Our model is a superpeer network model, c.f. (Nejdl *et al.*, 2002, Simon *et al.*, 2003), in which the network consists of several, possibly overlapping, *peergroups*. The peers of each peergroup are connected to a single superpeer. There is frequent communication between peers within a peergroup, and less frequent communication between superpeers. Superpeers serve as coordinators of the overall network while other peers represent group members at the network edge. Superpeers will typically be wired networked computers while peers may be wired computers or mobile, more resource-constrained, devices.

Peergroups form in order to undertake group projects, and peers may join or leave a peergroup at any time. Each group project is coordinated by one superpeer (superpeers may coordinate multiple projects). Peers contact the relevant superpeer in order to join a project — we term the set of peers working on a project a *project group*, or just *group*.

Information about the group's work on a project is distributed between the peers of the group and stored in local repositories at the peers. Peers' actions are notified by them to their superpeer, which manages the distribution, replication and consistency of information across the group. Each superpeer supports Event-Condition-Action (ECA) rule processing capabilities c.f. (Papamarkos *et al.*, 2006). A rule's Event part matches events occurring at a peer or at the superpeer, and its Condition part is evaluated with respect to the superpeer's repository. There is 'immediate' coupling between the event part and the condition part (see (Paton, 1999) for discussion of ECA rule coupling modes). A rule fires when its condition evaluates to true.

In general, several rules may fire as a result of some event occurrence in the peergroup. We assume that in such cases the rules that fire are ordered by a precedence relationship, or if not that they commute (see, e.g. (Papamarkos *et al.*, 2006) for discussion of techniques for determining the confleunce properties of sets of ECA rules). When a rule fires, one or more instances of its Actions part are scheduled for execution within the peergroup, with each instance executing at a single peer. There is 'detached' coupling between the Condition and the Action i.e. the superpeer does not wait to receive acknowledgement from peers that they have received and executed these actions. We assume the provision of reliable message passing services between the superpeer and peers, so that actions are eventually propagated to a disconnected peer and executed there when it reconnects to the network.

These ECA rule capabilities at the superpeer are used for a number of purposes:

- to encode the workflow of each project as a finite state machine (FSM) and maintain the project state;
- to encode the replication policies and consistency requirements of the project: peers will notify the superpeer of updates on their local copies of group artefacts, which it can propagate to peers holding replicas;

Table 1. Peer Core Services.

Event Notification Services Detect event occurrences at a peer. Notify the superpeer's Event Handler Service, sending the type of the event, event parameters, and any data changes (insertions/deletions/updates).
Repository Connection Services Manage connection with the local peer repository. They include an *Update Manager* that submits data update requests to the repository, and a *Query Manager* that communicates with the repository's query processing engine.
Messaging Services Responsible for message exchange between peers. Wrap/unwrap outgoing/incoming messages and pass them to the appropriate service. Provide reliable message passing between peers in the face of network dynamicity.
Resource Information Services Provide information about the peer's computational resources (e.g. data storage, CPU, bandwidth), their state, and their availability.
Object Sharing Services Send documents and other group artefacts to other peers. Metadata describing the artefacts is also sent.
Synchronous Forum Services Create a room for an online synchronous session; notify other group members of a room's creation; request group members to join a room; request the superpeer to create a room.

- to automate the passive mode delivery of awareness information to peers according to their current status, the status of the projects and tasks they are participating in, their preferences, and their context;
- to automate the generation and delivery of global summaries from detailed information and local summaries received from individual peers.

Table 2. Superpeer Core Services.

Routing Services Keep a list of the peers comprising the superpeer's peergroup and its neighbouring superpeers, in order to maintain the communication paths in the network.
ECA Rule Processing Services Include *Event Handler*, *Condition Evaluator* and *Action Scheduler* services, as described earlier.
ECA Rule Management Services Maintain the superpeer's Rule Base, including indexing its contents and providing query and update functionalities over it.
Synchronous Forum Management Services Allow the superpeer to satisfy peers' requests for room creation, notify the group of ongoing synchronous sessions, and support latecomers.

5. Awareness services. We now identify a set of services that peers and superpeers need to support in order to provide the types of group awareness identified in Section 3. Each peer (and superpeer) implements a set of core services that are necessary to support the group awareness services: Event Notification, Repository Connection, Messaging, Resource Information, Object Sharing, and Synchronous Discussion Forum services, as listed in Table 1. Superpeers in addition support ECA Rule Management and Processing, Routing, and Synchronous Discussion Forum Management services, as listed in Table 2.

Tables 3 and 4 list a set of services to be implemented at peers in order to support "passive" mode and "active" mode delivery of awareness, respectively. We indicate in each case for which type or types of awareness each service is needed. Table 5 lists a set of services to be implemented at superpeers. Figure 1 illustrates the interactions between the services at a peer and a superpeer, showing both the core services and the awareness services listed in Tables 3, 4 and 5.

Referring to the user requirements identified in Section 3, we see that these can be met as follows by the interoperating peer and superpeer services:

- A group's superpeer has knowledge of the group members, the project tasks and project workflow, and the responsibilities assigned to each group member through services $joinProject$, $leaveProject$, $acceptTask$ and $relinquishTask$, thus meeting requirements R1, R4.
- Peers can notify their superpeer regarding their status, their progress on tasks, and the time, location and environment in which they undertake their project-related actions through services $notifyAction$ and $notifyActionSummary$; the superpeer can inform group members of the group's progress on tasks and the project as a whole through services $provideProject/Task/PeerSummary$; thus meeting requirements R6, R7, R8.
- Peers can communicate with each other to redefine deadlines and reassign tasks through the messaging and synchronous forum services, thus meeting requirements R2, R3. Peers that are not online while such changes take place can be updated by the superpeer when they rejoin the network by the $provideAwarenessSummary$ service.
- Peers can notify their superpeer regarding their own availability and the availability of their resources through services $notifyCollaborationStatus$ and $notifyAvailability$; the superpeer can propagate this information to other relevant peers through the $provideAvailabilitySummary$ service; thus meeting requirements R5, R9.

6. Implementation. While there are no specific technical requirements for peers in our model — which could be PCs, laptops, smart devices etc. — superpeers need to be reliable. We therefore envision the implementation of superpeers on a Cloud infrastructure — which could be public or private. The virtualisation of resources in the Cloud can be used to instantiate superpeers whenever needed

Table 3. Peer Awareness Services - Passive Mode (these all call the Event Notification Service)

$joinProject(superpeer, project, context, profile, resources, capabilities)$ Peer requests to join a project. Needed for: Activity and process awareness.
$leaveProject(superpeer, project)$ Peer requests to leave a project. For: Activity and process awareness.
$acceptTask(superpeer, project, task)$ Peer accepts responsibility for a task in a project. For: Activity and process awareness.
$relinquishTask(superpeer, project, task)$ Peer relinquishes responsibility for a task in a project. For: Activity and process awareness.
$notifyAction(superpeer, project, task, action, context, status)$ Peer notifies superpeer of an action in relation to a project task (e.g. upload/download/view/ update of a document, creation/contribution to a forum) as well as their current context (e.g. time, location, environment, workspace involved) and their current status in relation to the task (e.g. pending acceptance, accepted, resource request pending, task being undertaken, task completion delayed, task completed). Allows the superpeer to track the state of the project workflow and to provide partial and complete views of this to group members. For: Activity, process, context and communication awareness.
$notifyActionSummary(superpeer, project, summary, context, status)$ Peer sends summary information about its actions to the superpeer, e.g. after a period of disconnection from the network. For: Activity, process, context and communication awareness.
$notifyCollaborationStatus(superpeer, project, fromTime, context, status)$ Peer notifies superpeer of its status in relation to being available to participate in synchronous collaboration with other group members. For: Availability awareness.
$notifyAvailability(superpeer, project, availabilityMetadata)$ Peer notifies superpeer of its availability for working towards a project and the availability of its resources that are relevant for the project. For: Availability awareness.

in order to ensure high availability. Moreover, Cloud Computing has become a key computing paradigm for service provisioning and through Cloud-based superpeers we envisage the provision of *awareness as a service* (AaaS), to be consumed by peers.

We have implemented a prototype that supports a subset of the services described earlier, as a proof-of-concept of our approach and also so to undertake an empirical study investigating its performance and scalability (see Section 7).

The implemented peer services are event notification, repository connection, messaging, resource information and object sharing. The implemented superpeers also support ECA rule management, rule processing, and routing services. We refer the reader to (O'Hagan, 2014) for full implementation details of the prototype and we give here an overview of its salient features.

In order to run superpeers in the Cloud we needed the flexibility of an infrastructure-as-a-service (IaaS) Cloud service rather than a higher-level service

Table 4. Peer Awareness Services - Active Mode

$requestAvailability(superpeer, project)$ Peer requests information from the superpeer about the availability of other group members and their resources. Needed for: Availability awareness.
$requestProjectSummary(superpeer, project, fromTime)$ Peer requests from the superpeer an overview of the actions occurring within the project after a specified time. For: Activity and process awareness.
$requestTaskSummary(superpeer, project, task, fromTime)$ Peer requests from the superpeer an overview of the actions occurring in respect of a specific task of the project after a specified time. For: Activity and process awareness.
$requestPeerSummary(superpeer, project, peer, fromTime)$ Peer requests from the superpeer an overview of the actions undertaken by a particular group member on the project after a specified time. For: Activity and process awareness.
$requestAwarenessSummary(superpeer, project, fromTime, awarenessType)$ Peer requests from the superpeer a summary of the activity, process, communication, context, or availability awareness information in respect of the project. More specific similar services can request information as relating to a specific task, peer, resource etc. For: All awareness types.

such as the Platform-as-a-Service Google App Engine[2]. We chose Amazon Elastic Compute Cloud (AWS EC2) as a cost-effective and well-documented IaaS service[3].

Table 5. Superpeer Awareness Services.

$assignProject(project)$ Handles the assignment of a new project to the superpeer. The superpeer stores information about the project and its constituent tasks in its local repository.
$provideProjectSummary(peer, project, fromTime)$ Sends a global summary of the project to a peer in response to a request it receives from the $requestProjectSummary$ peer service. A project summary may also be sent to a peer if specific events occur at the peer or superpeer and specified conditions hold, via appropriate ECA rules hosted at the superpeer (i.e. in "passive" mode).
$provideTaskSummary/providePeerSummary$ Behave similarly, in response to the $requestTaskSummary/requestPeerSummary$ peer services. Task/peer summaries can also be sent to peers in "passive" mode via appropriate ECA rules.
$provideAwarenessSummary(peer, project, fromTime, awarenessType)$ Sends awareness information to a peer in response to a request from the $requestAwarenessSummary$ peer service, or in "passive" mode via ECA rules.
$provideAvailabilitySummary(peer, project)$ Sends availability information to a peer in response to a request from the $requestAvailability$ peer service, or in "passive" mode via ECA rules.

In the prototype, the peer and superpeer functionalities are implemented in

[2]https://developers.google.com/appengine/
[3]http://aws.amazon.com/ec2/

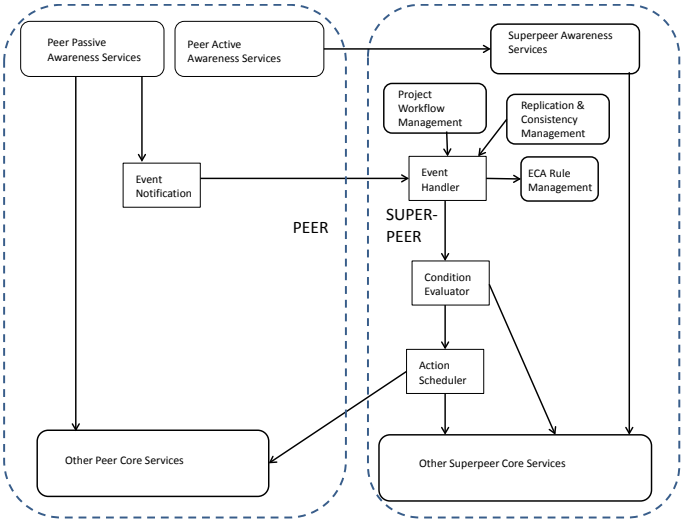

Fig. 1. Interaction between Peer and Superpeer Services for Awareness Provision

Java for reasons of portability and reusability — in particular, we aimed to reuse as much as possible classes between the peer and superpeer implementations. We use Apache Jena[4] for the RDF processing, specifically, its RDF/XML serialization/deserialization and RDF persistence functionalities. To persist the RDF graphs we use the Jena TDB triplestore.

6.1. Implementation approach. Our initial approach to building the P2P network functionality was to investigate existing P2P libraries. We identified JXSE, a Java implementation of the JXTA protocols, as a promising starting-point[5]. The motivation for considering this kind of functionality was that a major problem for P2P networks is the difficulty of enabling heterogeneous peers to access a network (Shen *et al.*, 2010). Peers may be located behind firewalls and Network Address Translation (NAT) gateways, which restrict incoming and outgoing packets in several ways, making P2P connections difficult. JXTA is an open-source set of P2P protocols that enable P2P networks to allow access to nodes situated behind NAT gateways or firewalls. Of additional relevance is the fact that JXTA works by forming an overlay network with a superpeer architecture, which is the form of network we wanted to create.

However, in early attempts at prototyping with JXTA, we found the JXTA system to be rather complex. The JXSE project had been discontinued in 2011[6], meaning that there was no online community to correspond with about the open-

[4]http://jena.apache.org/
[5]https://jxse.kenai.com
[6]https://kenai.com/projects/jxse/pages/LatestNews

source code. This is in contrast to the other third-party libraries that we used for the prototype; for instance, Jena has a large and active online community. Also, the latest (and last) version of JXSE had little supporting documentation due to the project being discontinued.

The performance evaluation that we planned would be run using a superpeer deployed in the Cloud and a group of peers on a Local Area Network (LAN). Therefore, NAT traversal capability was not a core requirement for the evaluation study. So, in this first prototype, we have left it for later iterations to incorporate this functionality. Instead, we focused on building a system that would support the evaluation and that would be able to flexibly support the implementation of additional awareness services.

Since the peers' and superpeers' data is stored as RDF, the event notifications (sent from peer to superpeer) describing local peer events are also encoded in RDF. Each connection to each peer is handled by a different Runnable at the superpeer, which creates a subthread for handling the event stream being received from the peer. When a peer first connects to a superpeer, the superpeer sends it the network identities of each member of the peergroup, and the peer subsequently creates continuous connections with the rest of the peergroup. Therefore, while a member of the peergroup is connected to the superpeer, it also has a single continuous connection for each member of the peergroup.

A superpeer receiving an event notification passes it to its ECA rule processor. The processor identifies rules that match the event type of the notification, which is a property of the event described by the RDF. Rules may have conditions that are tested on the contents of the superpeer's local RDF repository (containing the relevant project-related data). If the condition is satisfied, then the action specified in the rule is executed locally or remotely, depending on the type of rule. For this prototype, we built custom ECA rule-processing capabilities, without the use of a third-party rules engine, so as to easily and rapidly tailor the prototype to our specific requirements. For instance, the majority of actions that result from a superpeer's rule processing are a form of remote procedure call (RPC) whereby the superpeer sends a message (and perhaps some accompanying data) to a peer and the message specifies an action that should be performed at the remote peer node.

In light of this, the action parts of our ECA rules are templates of RPCs, encoded in XML. The condition parts of our rules are expressed as unique identifiers for Java instances of a specific class, called condition handlers, which are indexed at the superpeer. These condition handler objects are able to query the RDF repository in order to check that a certain condition holds or to retrieve specific information in order to complete the RPC template. If a specified condition does not hold, the rule processing sequence is terminated and no action is executed. Information that may be retrieved by a condition handler includes the identities of the peers who should execute the action (which may include the superpeer itself) and the parameters supplied to the RPC template. We have implemented a rule

processor API through which an application programmer is able to add rules to the rule base and to obtain lists of rules that match particular events.

Although we refer to the actions that result from a superpeer's rule processing as a form of RPC, these actions differ from usual RPC calls due to the 'detached' coupling mode between each ECA rule's condition and its action. For instance, XML-RPC requests are followed by an XML-RPC response which contains the result of the XML-RPC. This is not the case in our prototype system: a superpeer may send a message to a peer specifying a particular action that should be executed at that peer; however, the superpeer does not expect a response detailing the result of the remote action execution.

6.2. Communication protocols. We have implemented two different communication protocols between peers and superpeers. Here we describe these protocols in the example context that we used for the performance evaluation. This context is that a peer makes an update to a document stored locally; other peers in the peergroup may also hold copies of this document; the system must now initiate a sequence of steps that lead to all copies of the document in the peergroup being synchronised in order to reflect the update. We chose this example for the performance evaluation as it fully exemplifies the general sequence of steps relating to the 'passive' delivery of awareness information to a peergroup (see Fig. 1), namely: an event occurring at a peer; the event being notified by the peer to the superpeer; the superpeer determining which rules are triggered by the event, evaluating their conditions (if any) and generating actions to send to relevant peers of the peergroup; and these actions being executed by these peers. Although we describe the two communication protocols in this example context, the underlying functionality of these protocols could be used to provide all other passively delivered awareness services discussed earlier.

In the definitions of the two protocols in Algorithms 1 and 2, SP is a superpeer coordinating a group of peers, $\{P_1, \ldots, P_n\}$; P_{init} is a member of $\{P_1, \ldots, P_n\}$ which performs an update on its local copy of a document, *doc*; P_{init} produces an RDF description, ev, of this event and some associated data, *dat*; specifically, *dat* is a 'patch' that can be applied to copies of *doc* in order to update them too; P_{target} is the subset of $\{P_1, \ldots, P_n\}$ that hold a copy of *doc*.

Algorithm 1 Protocol 1

1: P_{init} sends the message (ev, dat) to SP.
2: SP executes the appropriate ECA rule(s) in order to determine P_{target}.
3: SP sends to each peer in P_{target} a message (in unicast mode) containing instructions on how to update their copies of *doc* and the accompanying data for this purpose, *dat*.
4: Each peer in P_{target} executes the requested update on their local copy of *doc* using *dat*.

Algorithm 2 Protocol 2

1: P_{init} sends ev to SP.
2: Same as step 2 in Protocol 1.
3: SP sends to P_{init} an action message. This message contains the identities of the members of P_{target} and instructions for how P_{init} should carry out step 4.
4: P_{init} sends a message to each peer in P_{target} (in unicast mode). This message contains instructions to update their copies of *doc* and the accompanying data for this purpose, *dat*.
5: Same as step 4 in Protocol 1.

As mentioned earlier, the prototype currently does not support NAT traversal and firewall hole-punching. However, the functionality of the communication protocols provides an alternative in certain circumstances. For instance, if a peer cannot communicate directly with another peer, it can relay any messages through its superpeer assuming that both peers have direct connections to the superpeer, which would have to be the case at some point by virtue of them being members of its peergroup.

Reliable message passing between the superpeer and peers is provided in the prototype, so that actions are eventually propagated to a peer and executed there when it reconnects to the network. This is achieved by peers and superpeers storing locally messages and data that have failed to be sent, due to the intended recipient not being connected to the network. When the intended recipient reconnects, the stored messages and data are then forwarded. Messages and data that have not yet been sucessfully forwarded to their intended recipient are also persisted in the case that the sender disconnects from the network; the sender can then resend those messages when it reconnects if the intended recipient is also connected to the network at that time.

7. Performance evaluation. Our performance evaluation investigated the following: Given an occurrence of an update to *doc* at P_{init}, what is the average time t taken for the members of P_{target} to update their copies of *doc*?

We examined the effect of several different variables on the size of t:

- The value of n, i.e. the size of the peergroup.
- The size of P_{target} as a fraction of n.
- The size of the document *doc* and of the 'patch' *dat*.
- The communication protocol used by the system to propagate the update (i.e. Protocol 1 or Protocol 2).

We conducted two evaluations, one with the peers running on a local LAN and a larger-scale one with the peers running in the Cloud. For both evaluations, the superpeer maintains a continuous connection to every peer in its peergroup and likewise the peers maintain a continuous connection to each other. The work

required to establish these connections (creating a new Thread and establishing a Socket connection) is done prior to the running of each experimental instance. In all cases, unicast rather than multicast communication is used. The RDF event notifications are sent as binary strings and the 'patch' objects used for *dat* are sent serialized using ObjectOutputStreams. The 'patch' objects are created using a modified version of the google-diff-match-patch library[7]: we modified the source so that the 'patches' could be serialized.

7.1. First evaluation. For the first evaluation, a single peer was run on each of 32 machines connected by a LAN in one of our institution's computer labs. Each machine was running Windows 7 on an Intel Core i5 3.20 GHz processor with 8GB RAM. The superpeer was run on an m1.large AWS EC2 instance with a 64-bit Linux AMI[8]. Since logging is an archetypal crosscutting concern, we used AspectJ to log the timings data[9]. Each experimental instance was run 10 times. For each of these runs, the mean time taken for each member of P_{target} to synchronise its copy of *doc* with P_{init} was computed (in milliseconds), as well as the standard deviation. We refer the reader to (O'Hagan, 2014) for the full set of results.

Figs. 2–5 show the behaviour of the two protocols under different conditions. Each graph plots four lines, representing respectively a P_{target} size of 25%, 50%, 75% and 100% of n. The x axis shows the number of peers as the size of the peergroup, n, increases (from 8 up to a maximum of 32 in steps of 4). The y axis shows the mean time taken for each member of P_{target} to synchronise its copy of *doc* with P_{init} (in milliseconds). In all cases the size of *dat* is fixed to be 25% of the size of *doc*.

Figs. 2 and 3 relate to a *doc* size of 100Kb. A good scalability for both Protocol 1 and Protocol 2 can be observed as n increases. As the size of the peergroup increases, the average time taken for the members of P_{target} to synchronise their documents remains, from the user's perspective, within an acceptable range. It can be seen that this average time is greater when Protocol 2 is used in comparison to Protocol 1. This is because Protocol 2 requires an extra round of data transfer between machines (peer to superpeer, superpeer to peer, and finally, peer to peer). Additionally, the graphs for Protocol 2 are generally less smooth than those for Protocol 1. This effect can be attributed to the additional 'noise' created by the additional round of data transfer. Figs. 4 and 5 relate to a larger *doc* size of 1Mb. Similar scalability and relative performance between the two protocols is observed as with the *doc* size of 100kb. The average time taken for the members of of P_{target} to synchronise their documents is higher with this larger *doc* size but still remains, from the user's perspective, within an acceptable range. Similar results are obtained with larger sizes of *dat* as a proportion of *doc* for both protocols and both document sizes.

[7] http://code.google.com/p/google-diff-match-patch/
[8] http://aws.amazon.com/ec2/instance-types/
[9] http://eclipse.org/aspectj/

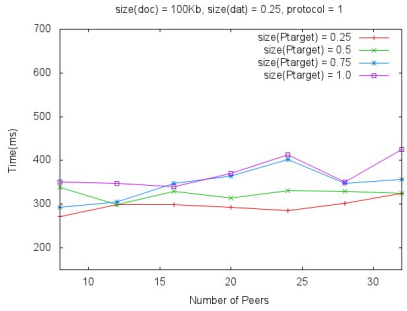

Fig. 2. Lab, Protocol 1, 100Kb doc

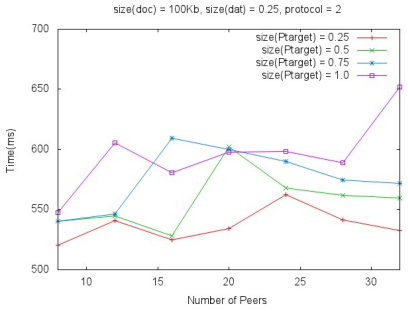

Fig. 3. Lab, Protocol 2, 100Kb doc

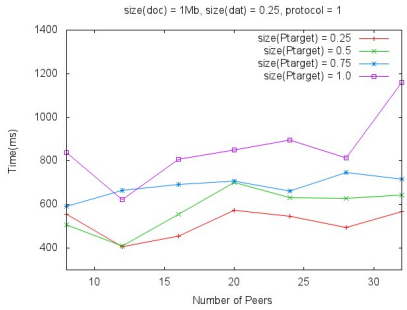

Fig. 4. Lab, Protocol 1, 1Mb doc

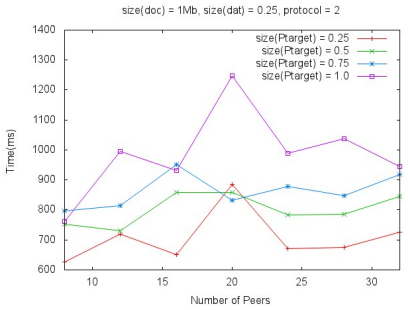

Fig. 5. Lab, Protocol 2, 1Mb doc

7.2. Second evaluation. Our second evaluation aimed to explore further the scalability of our approach for larger peergroups than was possible in the lab, up to a maximum of $n = 60$ which is pragmatically at the upper limit for collaborating groups in PBL. For this second evaluation, we have a single superpeer node and a group of peer nodes all of which are AWS EC2 virtual machines. We also have an EC2 virtual machine that functions as a server node, supporting the execution of the experiment but not itself part of the system. The superpeer is run on an m1.large instance, as before, and the peers on lower-spec t1.micro instances; 64-bit Linux AMIs are used for both types of instance. The virtual machines are launched from the server node using the AWS Java API. Through the API, peers are configured to have access to Strings of 'user data' when they are launching. The user data is a script that makes various necessary changes to the virtual machine so that it can operate as a peer node (update the OS, set permissions and classpaths, download the correct JRE version, etc); the script then downloads required information (binaries and jar dependencies) from the server; and finally it runs the peer program, connecting to the superpeer.

We examined the effect of same set of variables on the size of t as in the first

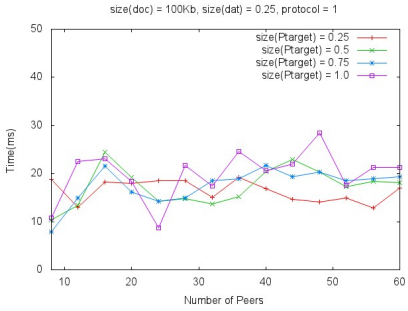

Fig. 6. Cloud, Protocol 1, 100Kb doc

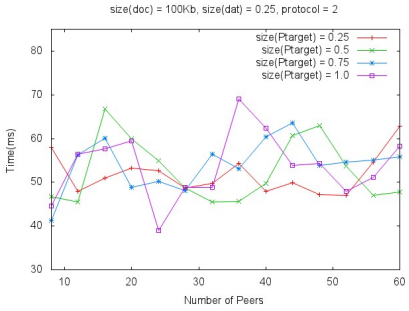

Fig. 7. Cloud, Protocol 2, 100Kb doc

evaluation. Figs. 6 – 9 show the behaviour of the two protocols under different conditions. Figs. 6 and 7 relate to a *doc* size of 100Kb, and Figs. 8 and 9 to a *doc* size of 1Mb. We observe again good scalability for both protocols and both document sizes as n increases. Again the average time taken to synchronise all copies of *doc* is generally greater when Protocol 2 is used. Similar results are obtained with larger sizes of *dat* as a proportion of *doc* for both protocols and both document sizes. One outlier is the value for Protocol 1 with a 1Mb *doc* size and $n = 8$ in Fig. 8. We conjecture this may be due to initialisation effects in using the AWS as this was the first set of experiments run in the Cloud; but further investigation is needed to verify this.

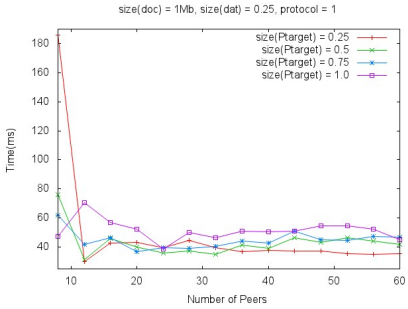

Fig. 8. Cloud, Protocol 1, 1Mb doc

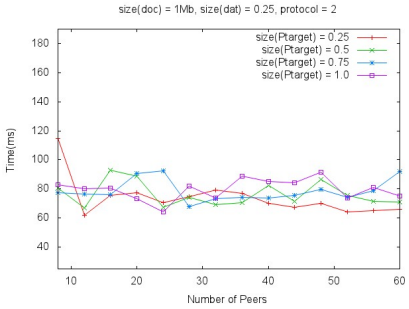

Fig. 9. Cloud, Protocol 2, 1Mb doc

Each of the data points plotted in Figs. 2–9 is an average of 10 timings. The average of the standard deviations (SDs) of the data points in Fig. 2 is 57.1. Likewise, the average SDs for the data points in Figs. 3 – 9 are 49.8, 206.6, 165.9, 12.8, 15.0, 22.0, 14.6 respectively. The consistency and size of these SDs points to the robustness of the experimental results.

As would be predicted theoretically given the parallelisation of the SP to P_{target} interactions in Protocol 1 and the P_{init} to P_{target} interactions in Protocol

2, the average time taken for the members of P_{target} to synchronise their documents remains roughly constant as n increases. The acceptable performance and scalability of both protocols means that, even though slower, Protocol 2 is an acceptable alternative to Protocol 1 at times when the superpeer may be under a high communications load, and applications could switch over to Protocol 2 at such times allowing the peers to take on some of this communications burden.

8. Conclusions. The provision of awareness information to group members engaged in collaborative project-based work enables more efficient sharing of knowledge and more timely support and decision making. While mature proposals have been reported in the literature for web-based groupware systems, there has been little work for P2P groupware systems and proposals have only partially addressed the requirements for awareness provision to collaborating groups. Furthermore, most previous approaches to awareness provision in P2P systems have implemented the awareness mechanisms as part of applications.

In contrast, we propose a generic set of primitive awareness functionalities and services to be provided within the P2P middleware, on top of which specific awareness functionalities for groupware applications can be developed. We envisage not only simple awareness information about group members' activities, e.g. in the form of notifications, but also support for group processes, aiming to enhance the group's abilities to undertake a project effectively and efficiently.

We have discussed the challenges arising in P2P systems when addressing awareness provision, have reviewed the user requirements relating to generic awareness functionalities, and have described how our primitive services meet these requirements. Services at superpeers and peers can be composed to build more complex services, at varying levels of abstraction, in order to provide the required awareness functionality for a particular P2P groupware application.

We have developed a prototype implementation supporting a subset of the proposed services as a proof-of-concept. Using this prototype, we have undertaken a performance evaluation considering factors such as sizes of peergroups, message sizes, and replication policies for peers' data, the results of which points to good performance and scalability of our approach.

Future work includes investigation of the performance of the prototype under more dynamic network conditions, where the size and make-up of the group are continuously changing. This would stress-test the reliable message passing services, possibly identifying areas for improvement when dealing with more realistic network conditions. In the same vein, we would like to study multi-cast communication and compare the results to the unicast communication used in this work. An interesting extension of our prototype is to handle mobile peers using smartphones. This would focus on the advantages of using mobile devices, such as greater opportunity to leverage context awareness; and also on dealing with some of the challenges posed by mobile peers, such as resource constraints and

poor connectivity. Finally, implementation of P2P groupware applications using our prototype would enable its performance to be evaluated for real workloads.

Acknowledgment. This work has been partially supported by Spanish Research Project TIN2013-46181-C2-1-R COMMAS (Computational Models and Methods for Massive Structured Data).

REFERENCES

R. Bentley, T. Horstmann, K. Sikkel and J. Trevor. Supporting Collaborative Information Sharing with the WWW. The BSCW Shared Workspace System, *The World Wide Web Journal*, 63-73, 1995.

K.A. Berman and F.S. Annexstein. An Educational Tool for the 21st Century: Peer-to-peer Computing. In Proc. Ohio Learning Network Conference, 2003.

A. Bulkowski, E. Nawarecki, and A. Duda. Peer-to-Peer: an Enabling Technology for Next-Generation E-learning. Fourth EDEN Research Workshop, Spain, 2006.

P. Busetta and M. Merzi. Approach to the Integration of Peer-to-Peer Systems with Active Environments, In Proc. Workshop from Objects to Agents (WOA 2003), 42-48, 2003.

L. Chan, S. Karunasekera, A. Harwood, and E. Tanin. CAESAR: middleware for complex service-oriented peer-to-peer applications. In Proc. 2nd Workshop on Middleware for Service Oriented Computing, at Int. Middleware Conference (MW4SOC '07), New York, 12-17, 2007.

W. Dou, J. Wang, P2P-Based Knowledge Grid Oriented Toward Cooperative Cognition, In Proc. 2nd Int. Workshop on Knowledge Grid and Grid Intelligence (KGGI04), 63-71, 2004.

G. Fakas, B. Karakostas. A Peer to Peer Architecture for Dynamic Workflow Management Using Web Services. *Information and Software Technology Journal*, Elsevier, 46(6), 423-431, 2004.

P. Fenkam, E. Kirda, S. Dustdar, H. Gall, and G. Reif. Evaluation of a Publish/Subscribe System for Collaborative and Mobile Working. In Proc. 11th IEEE Int. Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '02), Washington, DC, USA, 23-28, 2002.

D.G. Galatopoullos, D.N. Kalofonos, and E.S. Manolakos. A P2P SOA enabling group collaboration through service composition. In Proc. 5th Int. Conference on Pervasive Services, 111-120, 2008.

C. Gutwin, S. Greenberg, and M. Roseman. Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation. *BCS HCI* 1996, pp 281-298.

H. Jin, Z. Yin, X. Yang, W. Fang, J. Ma, H. Wang, J. Yin. APPLE: A Novel P2P Based e-Learning Environment. Distributed Computing, Springer LNCS vol. 3326, 52-62, 2004.

T. Kawashima, J. Ma. TOMSCOPA synchronous P2P collaboration platform over JXTA. In Proc. Int. Workshop on Multimedia Network Systems and Applications, 85-90, 2004.

E. Kirda, H. Gall, P. Fenkam, G. Reif. MOTION: A Peer-To-Peer Platform for Mobile Teamwork Support. In Proc. 26th Int. Computer Software and Applications Conference (COMPSAC02), 1115 - 1117, 2002.

R. Kurmanowytsch, E. Kirda, C. Kerer, S. Dustdar. OMNIX: A topology-independent P2P middleware. In Proc. CAiSE Workshops 2003, Springer LNCS vol. 75, 47-56.

Y. Li, J. Bu, C. Chen, X. Xu. Reliable Communication Based on P2P Architecture on Real-Time Collaborative Editing System. In Proc. 8th Int. Conference on CSCW in Design, 244 - 249, 2004.

J. Ma, M. Shizuka, J.Lee, R Huang. A P2P Groupware System with Decentralized Topology for Supporting Synchronous Collaborations. In Proc. Int. Conference on Cyberworlds, 54-61, 2003.

M. Margaritis, C. Fidas, N. Avouris, V. Komis. A Peer-to-Peer Architecture for Synchronous Collaboration over Low-Bandwidth Networks. University of Patras, TechRep 26500, Greece.

R.T. Mason and T.J. Ellis. A Recommendation for the Use of Service Oriented Architecture (SOA) to Bridge the LMS to LOR Data Movement Interoperability Gap for Education. In Proc. of Informing Science & IT Education Conference (InSITE), 43-56, 2010.

R. Menchaca-Mendez, E. Gutierrez-Arias, J. Favela. Opportunistic Interaction in P2P Ubiquitous Environments. In Proc. CRIWG'04, Springer LNCS vol. 3198, 349-362, 2004.

W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmr, and T. Risch. Edutella: A P2P Networking Infrastructure Based on RDF. In Proc. 11th World Wide Web Conference, 604-615, 2002.

T. O'Hagan. Development of Awareness as a Service (AaaS) in P2P Groupware using Cloud Computing. Birkbeck College, UK, 2014. Available at http://www.dcs.bbk.ac.uk/research/techreps/2014/bbkcs-14-01.pdf

Y. Oasis, S. Abdala, A. Matrawy. A Multilayer P2P Framework for Distributed Synchronous Collaboration. *IEEE Internet Computing*, 33-41, 2006.

G. Papamarkos, A. Poulovassilis, and P.T. Wood. Event-Condition-Action Rules on RDF Metadata in P2P Environments. *Computer Networks*, 50(10), 1513-1532, 2006.

D. Parker, D. Cleary. Building Richer JXTA Applications with Collaborative Spaces in a Peer-to-Peer Environment. In Proc. 38th Hawaii Int. Conference on System Sciences, vol. 9, pp 301b, 2005.

N. W. Paton. *Active Rules in Database Systems*. Springer, 1999.

A. Poulovassilis and F. Xhafa. Building Event-Based Services for Awareness in P2P Groupware systems. In Proc. 8th Int. Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC 2013), 200-207, 2013.

N. Preguia, J.L. Martins, H. Domingos and S. Duarte. Integrating Synchronous and Asynchronous Interactions in Groupware Applications. Groupware: Design, Implementation, and Use. Springer LNCS vol. 3706, 89-104, 2005.

Ch. Qu and W. Nejdl. Constructing a web-based asynchronous and synchronous collaboration environment using WebDAV and Lotus Sametime. In Proc. 29th Annual ACM SIGUCCS Conference on User services, 142-149, 2011.

S. Rossi, P. Busetta: Towards Monitoring of Group Interactions and Social Roles via Overhearing. Cooperative Information Agents VIII, Springer LNCS vol. 3191, 47-61, 2004

X. Shen, H. Yu, J. Buford (Eds). *Handbook of Peer-to-Peer Networking*, Springer, 2010.

B. Simon, Z. Mikls, W. Nejdl, M. Sintek, J. Salvachua. Smart Space for Learning: A Mediation Infrastructure for Learning Services. In Proc. 12th World Wide Web Conference, Hungary, 2003.

F. Xhafa, L. Barolli, S. Caballé, R. Fernandez. Efficient peerGroup management in JXTA-Overlay P2P system for developing groupware tools. *The Journal of Supercomputing*, 53(1), 45-65, 2010.

F. Xhafa, A. Poulovassilis. Requirements for Distributed Event-Based Awareness in P2P Groupware Systems. In Proc. AINA-2010 Workshops, 220-225, Australia.

You, Y. and Pekkola, S. Meeting others – supporting situation awareness on the WWW. *Decis. Support Syst.* 32(1), 71-82, 2001.

J. Zumbach, A. Hillers, and P. Reimann. Supporting Distributed Problem-based Learning: The Use of Feedback in Online Learning. In T. Roberts (Ed.), *Online Collaborative Learning: Theory and Practice*. Idea, 2003.

Received February 2014

A. Poulovassilis has an MA in Mathematics from Cambridge University and an MSc and PhD in Computer Science from Birkbeck, University of London. Her research interests center on information management, integration, and personalization. Since 2003, she has been Co-Director of the London Knowledge Lab, a multidisciplinary research institution which aims to explore the future of knowledge and learning with digital technologies.

F. Xhafa holds a PhD in Computer Science from the Department of Languages and Informatics Systems (LSI) of the Technical University of Catalonia (UPC). Currently he is *Professor Titular d'Universitat* at LSI/UPC. His research interests include parallel and distributed algorithms, security, optimization, networking and distributed computing. More details can be found at http://www.lsi.upc.edu/∼fatos/

T. O'Hagan has a BA Honours in Philosophy from Warwick University and an MSc in Computer Science from Birkbeck, University of London. He undertook his dissertation at the London Knowledge Lab in the area of Awareness-As-A-Service for P2P systems. Currently he is a software developer in the commercial sector.