An Agent-based Approach to Assist Collaborations in A Framework for Mobile P2P Applications

Mengqiu Wang, Heiko Wolf, Martin Purvis, and Maryam Purvis

University of Otago, Dunedin, New Zealand,
`[mwang, hwolf1, mpurvis, tehrany]@infoscience.otago.ac.nz`

Abstract. The design of ad-hoc, wireless, P2P applications for small mobile devices raises a number of challenges for the developer, with object synchronisation, network failure and device limitations being the most significant. In this paper, we introduce the FRAGme2004 framework for mobile P2P application development. To address data availability and stability problems, we devised an agent-based fostering mechanism to protect applications against data losses in cases of peer dropping out. In contrast to most current literature, we focus on small scale P2P applications, especially gaming applications.

1 Introduction

P2P architecture is the next-generation network paradigm to replace the traditional client-server architecture. Typical P2P systems are characterized by the decentralized control, scalability and robustness. Despite the popularity of internet-based large scale P2P systems, small scale P2P applications that run on small, mobile devices are becoming increasingly popular as well. The absence of centralized control and thus no single failure-point, extreme dynamism in structure, full mobility and flexibility are all desired features in many application domains, which can be achieved via mutual exchange of information and services over ad-hoc, wireless, P2P networks. Challenging problems arise in the development of such applications. Firstly, wireless, ad-hoc networks face problems such as stability, data integrity, routing, notification of joining and leaving peers, and in case of peer failure, fault tolerance. In such networks, the connections of the devices may be highly variable as devices may hop from online to offline unpredictably, and thus not reliable. Secondly, since the peers exist in a collaborative environment without central control, synchronisation of peers and distribution of resources become big issues. Thirdly, for the applications to run smoothly on the small devices we are aiming for, efficient management of local computing resources is a necessity. Those problems have been addressed in other research works, but mostly in an ad-hoc fashion [5], [12], [13].

When developing new applications, these issues have to be treated repeatedly as there exists no generic infrastructure which addresses all aforementioned problems. In this paper, we present an agent-based mechanism which addresses collaboration issues among peers, together with a framework called “FRAGme2004” which is designed for mobile P2P application development.

The rest of the paper is structured as follows: In Section 2 we will review related literature, followed by an introduction to the FRAGme2004 framework in Section 3. In Section 4 we will present an agent-based fostering-mechanism for assisting collaborative relationship building among peers. Finally, we will show some mobile gaming applications developed using FRAGme2004 in Section 5, and evaluate the effectiveness of the fostering-mechanism.

2 Literature Review

With the take-off of vast mobile communication networks and technology, Peer-to-peer (P2P) computing has established itself as the next-generation distributed computing. A good introduction on P2P systems is provided by Milojicic et al. [14], which comprehensively reviews the field of P2P computing and applications. Besides giving a general overview, case studies of prominent P2P systems such as Groove, Gnutella and Freenet outline the major challenges in networks. This review also shows the distinct differences between large-scale P2P applications such as file-sharing and instant messaging, which have been the main research interest in many papers, and smaller scale P2P applications such as P2P gaming. FRAGme2004 is designed to facilitate the development of small-scale P2P applications that are typically hosted in a wireless ad-hoc network environment.

A topic discussed often is data availability. Cooper et al. [5] argue that “a complete system must ensure that important data remain preserved even if the creator or publisher leaves the system” (p.2). They developed a reliable preservation service built on a P2P architecture, on top of which digital library applications could be built. Another strategy, the “dissemination tree”, is used by Chen et al. [4] to reduce the number of replicas and the bandwidth needed for updates. Finally, Lin et al. [12] introduce a protocol to handle the loss and rebuilt of replicas. All those approaches are aimed at applications such as file sharing, where data loss and node failure harm the performance, but not the overall functioning of the system. Data availability and reliability is one of the design goals of FRAGme2004. In particular, we aim at small-scale P2P networks where every node failure can be a fatal threat to the functioning of the system. Our approach to this problem will be presented in more detail in Section 4.

FRAGme2004 is not the first effort to develop a framework for P2P applications. There have been different approaches in developing frameworks to allow the easy construction of P2P applications. Akehurst et al. [1] proposed a framework which includes group management and multicasting support to assist the design of complex applicactions. Another architecture incorporating a variety of devices that communicate in a P2P network is introduced by Kato et al. [11]. While most P2P applications to date can be narrowed to either the file sharing or instant messaging categories, Gerke et al. [8] introduce a framework that supports P2P services of any kind. Different from all these frameworks, FRAGme2004 aims at highly interactive small-scale P2P applications like gaming over wireless ad-hoc networks. Therefore, we tackle some challenges related specifically to small-scale systems, which we found not well covered in previous research.

Recently, researchers have been taking various approaches in exploring the use of agent technology in P2P systems. Dasgupta [6], [7] proposed an P2P architecture in which mobile agents are used as peer mediators, replacing traditional message-based protocols to achieve higher efficiency, robustness and scalability. Babaoglu et al. [2] designed and implemented the Anthill framework, which supports the design, implementation and evaluation of P2P applications based on agent concepts. Similar to Dasgupta's approach, Anthill uses mobile agents to accomplish distributed tasks such as resource discovery and file downloading. In the P2P architecture proposed by Homayounfar et al. [9], peers are modelled as agents with some intelligence (e.g., calculating the success probability of data search to perform faster searching) to enhance the capability and autonomy of peers. The agent-based fostering mechanism in FRAGme2004 is similar to the concepts in Homayounfar et al.'s work. Peers in a FRAGme2004 application are equipped with on-host service agents which are autonomous and have the intelligence to organize and reconstruct the collaboration relationship among peers. These collaboration relationships are crucial for data availability purpose, and will be explained in more detail in Section 4. The agents are also in charge of other collaboration tasks such as data exchange and synchronization.

3 The FRAGme2004 System

The FRAGme2004 framework is written in Java, and has a three-layer architecture. The layers communicate to each other via interfaces in order to achieve clear separation. The bottom layer is the Infrastructure Layer. This layer consists of the basic building blocks that address the communication requirements. A layer higher is the Object Layer. Object is the smallest entity that is distributed among the peers. The information and data that needs to be shared in the applications is encapsulated into objects, and the agents associated with each peer take care of the delivery, synchronization and life-cycle management of objects. The top layer is the Application Layer. A clearly defined API is provided to the developers for easy application development.

The FRAGme2004 framework frees application developers from networking and resource management. This includes the establishment of the underlying network infrastructure, notification of joining and leaving peers, communication and object exchange. Rather, they can focus on the higher layers of the application. To make applications run reliably at all times, the agents that take care of object synchronization and distribution also use a novel fostering mechanism that makes use of data redundancy to achieve the overall integrity and robustness even in cases of node failure. We will describe the three layers in detail in the next subsections.

3.1 Communication Layer

A basic requirement for an effective communication layer is to provide reliable networking services. Currently, the communication layer of FRAGme2004 is based on a middleware called JGroups [10], which is also being used in many other P2P projects to provide reliable multicasting. There are several reasons why we chose to use JGroups. First of all, JGroups provides reliable unicasting and multicasting, which frees us from having to worry about those low level details. Almost all P2P applications require efficient communication. This is especially true with complex applications like games, which FRAGme2004 is targeted at. Unicasting (point to point communication), although used in many places like file sharing applications, does not suffice here. In circumstances where peers in a group share the same resource or need to be notified at the same time for synchronization purposes, multicasting is called in.

Originally, we tested two approaches for FRAGme2004. One based on simple Remote Method Invocation (RMI) unicasting and one based on Multicasting, to determine the more efficient way of communication. It was shown that multicasting communication is more efficient than RMI calls in group communications, an effect that is aggrevated with an increasing number of peers. It is to be noticed that RMI and JGroups are not totally comparable due to their different marshalling mechanisms. However, the differences found can still be partially attributed to the fact that in the case of RMI, a sender has to contact every other peer one by one to share information with his group; while in multicasting he only needs to send one message for the information to be disseminated to all peers of his group.

The second reason for using JGroups is that JGroups gives us all the group management functions that we need. For example, creation and deletion of groups, notification of change of membership (joined/left/crashed peers) and so on. The importance of group management is obvious. While group management appears as a simple task, it conceals a lot more low level details, for example the establishment of the network structure and communication channels and the identification for groups.

Finally, we use JGroups because it provides support, albeit limited, for guaranteed delivery. In wireless communication, virtually no routing protocols can guarantee one hundred percent success of delivery. But with some failure detection and retransmission mechanisms as included in JGroups, we can assume the guarantee of delivery to a certain degree. Such a guarantee is very important in terms of object-level protocol design. It allowed us to implement high level protocols (for example the interaction protocols used by agents in Object Layer) with less overhead and therefore higher efficiency. One of the tradeoffs of having such guarantee of delivery is that the scalability of the P2P system is greatly restricted. In order to maintain reasonable efficiency under this guaranteed delivery policy, the network is constrained to be tightly-coupled. But since the target applications of FRAGme2004 are small-scale gaming applications, such a tradeoff is acceptable.

3.2 Object Layer

The Object Layer comprises functionalities for object sharing, exchange and synchronization of change. In case of peer failure, protection mechanisms are put in place to avoid data loss and the malfunctioning of other peers. To achieve this, several on-host service agents are created for each peer. Currently there are 3 kinds of agents that provide simple services in the framework. They are the object managing agent, synchronizing agent and fostering agent.

The object managing agent is in charge of the life-cycle management of any shared objects. When a new object(resource) is requested from the application layer, the object managing agent will locate the appropriate factory for creating instances of this object. It will gather all the required information for creating such an object, issue commands to the factory for creating the object, and return the object to the application layer. Upon receiving signals from the application layer indicating that an object is not used anymore, the object managing agent is responsible for checking the memory status. Depending on the amount of free memory available, it makes decisions of whether to cache the object for future use (by returning it to the factory) or to garbage collect it.

Once objects are created and handed back to the application layer, changes may be invoked on these objects both locally and remotely. In FRAGme2004, objects are assigned ownership, and by default objects are always owned by the peer that created them. To avoid cases where multiple peers invoke changes to the same object at the same time, we rely on synchronizing agents to synchronize the events. The change request is treated as a service request and is handled by the local synchronizing agent. The agent will first try to identify the ownership of the objects. If the object is owned locally, then it will simply invoke the change on the object, and then locate synchronizing agents residing on other peers and inform them. If the object is not owned by the current peer, the synchronizing agent will locate the owner of the object and request for a object-changing service on the synchronizing agent of the owning peer. The synchronizing agents talk in a specific protocol. A typical packet format used by this protocol looks like this:

Performative	Sender Agent ID	Receiving Agent ID	Message Type	Content

The fostering agents are agents that serve for a special purpose, namely “Peer Fostering”. Due to the high volatility of wireless ad-hoc networks, special care needs to be taken to prevent applications from information loss and potential problems that evolve from it. Therefore, we have devised a “Peer Fostering” mechanism to increase the fault-tolerance of applications in case of peer failure, by introducing dependence relationship among peers. When a peer leaves the group intentionally or accidentally, the peer fostering mechanism comes into effect. As every peer’s objects are fostered by another peer’s fostering agent, they will not be lost. If the dropping out peer was holding any objects that are important to other peers, for example, the ball in a sports game, the fostering agents will negotiate ownership transfer and delegate the ownership of such objects to a new peer. In the case where the previously dropped out peer rejoins, fostering agents will negotiate for the ownership to be transferred back to the rejoining agent. When a new peer joins, the existing fostering agents will negotiate among themselves to elect the agent with the lightest workload, and assign the new peer to be the chosen agent's fosteree. We will describe this in more detail in Section 4.

3.3 Application Layer

Because the bottom two layers take care of all communication and resource management, application programmers can focus solely on the domain-specific aspects of the application without worrying about generic problems that come with P2P networks. A clearly defined API is provided to the programmers at this layer to interact with the framework. The interaction mainly goes through an access point called "Control Center". Factory design patterns are used to connect applications to the framework. Programmers are responsible for implementing appropriate factories for resources (objects) that are going to be shared across the network.

4 Agent-Based Peer Fostering

Wireless communication poses some additional challenges compared to cable-bound communications. Reliability and packet loss are the major issues. At the low level, we rely on JGroups to provide us with reliable uni- and multicasting, but it doesn't save us from the higher level problem that applications can suffer from a high degree of peer failure. Temporary disconnection of devices frequently occurs, especially in dynamic real-world environments.

This problem has serious impact on the basic usability of applications. In games, for instance, it is generally not tolerable if a player is not able to continue the game just because of some temporary loss of connection (e.g., he walks into a lift). The problem would become more serious if not only that player suffered from the temporary disconnection, but also other peers were affected. Therefore, it is important that some mechanisms are in place to ensure the integrity of data and continuity of the execution in case of peer failure. Furthermore, it is also desirable to have the disconnected peer able to rejoin the application without loss of his previous data.

4.1 Introduction to "Peer Fostering" mechanism

To address the problems as per discussed above, we introduced a so-called "Agent-Based Peer Fostering" mechanism into FRAGme2004. Similar to the schemes used in some well-known P2P applications (Gnutella, Napster), peer fostering is built on the basic idea that there exists some degree of data redundancy in all P2P systems. Most of the other systems don't have any special scheme that optimizes the degree of redundancy, but rather leave the highly redundant data in the system. This works for systems where peers are highly capable terminals like PCs. But in our case, all devices are very limited in terms of available memory. Therefore, leaving such a high degree of redundency could significantly constrain the performance of applications. As the number of peer failure instances increases, the accretion of redundant data could bring the system to crash.

In order to enable all peers to make sensible decisions about what to do in case of other peers' failure, we need to have all peers collaborate. This process is easily modelled with agents. Each fostering agent is autonomously acting on behalf of its owner peer. Through negotiation and collaboration with other fostering agents, this agent could balance the workload (the number of fosterees) with other agents to make better overall decisions.

4.2 Peer fostering relationship

We define the set of existing peers in a system $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$. a_i is the fostering agent of p_i, and $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ is the set of vertices in a graph. If $n > 1$, the Peer Fostering State (PFS) is defined as:

Definition 1 *the peer fostering state of a system is a simple directed circle $\mathcal{C}_{pfs}$*

$$\mathcal{C}_{pfs} = \{\langle a_{i_1}, a_{i_2}\rangle, \ldots, \langle a_{i_{n-1}}, a_{i_n}\rangle, \langle a_{i_n}, a_{i_1}\rangle\}$$

where each occurrence of $\langle a_{i_a}, a_{i_b}\rangle$ is a directed edge in the circle from vertex a_{i_a} to vertex a_{i_b}. For any particular $\mathcal{C}_{pfs}$, the set $\{a_{i_1}, a_{i_2}, \ldots, a_{i_n}\}$ is a permutation of the set $\{a_1, a_2, \ldots, a_n\}$.

Each edge $\langle a_{i_a}, a_{i_b}\rangle$ also denotes a "fostering" relationship between the fostering agent of peer p_{i_a} and the agent of peer p_{i_b}. In such a relationship, we say a_{i_a} is "fostering" a_{i_b}, with a_{i_a} being the "Fosterer" and a_{i_b} being the "Fosteree".

When a peer drops out and the objects that he owns are not needed by the others, these objects still have to be sustained to allow him to rejoin. In our system, instead of having these objects taking up memory storage on every peer, only the fosterer of the dropped out peer needs to store them. The "Fosterer"-"Fosteree" relationship is illustrated in Figure 1.

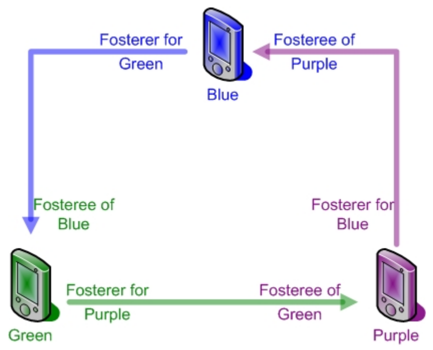


Fig. 1. Illustration of Fosterer-Fosteree Relationship

In the "Peer Fostering" scheme, each participating fostering agent not only knows who its "Fosteree" is, but also knows whom its "Fosteree" is fostering. The local knowledge of a fostering agent can be expressed as:

Definition 2 *the knowledge of peer a_i is*

$$K(a_i) = \langle F(a_i, a_j), F(a_j, a_k), Adopt(\{a_{x_1}, \ldots, a_{x_m}\})\rangle$$

where the set $\{a_{x_1}, a_{x_2}, \ldots, a_{x_m}\}$ contains the agents that a_i has temporarily "adopted", and where F denotes a fostering relationship.

The reason for storing this extra information will be clear when we explain the drop-out scenario in Section 4.3.

It is to be noticed that such relationships among fostering agents are updated every time a new peer joins, or an existing peer drops out. In the case of a new peer joining, the existing fostering agents will negotiate among themselves to determine which agent gets to foster the new agent. The fostering agents follow a set of interaction protocols that allow the agents to make conversations to build the relationships. These conversations are initiated by the "active" agent — the "Fosterers". Although the reader will notice that every single fostering agent is a "Fosterer" in some relationship, the agents don't have such global knowledge, and such knowledge is not needed either. Each agent only makes sure that the relationship in which it is the "active" party is properly pursued. It is not hard to see that when all agents finish building their own relationships, each agent will be fostered by some other agent. In order to correctly build the fostering relationships, we make the assumption that new peers are joining the network one at a time.

Each agent has its own thread of execution to secure its autonomy, thus the acquision of information and negotiation among agents happens behind the scene of the main gaming thread, and thread-safety measures are taken in our implementation. This relationship building phase is essential, but it generates a very small amount of traffic in the network and therefore its impact is negligible.

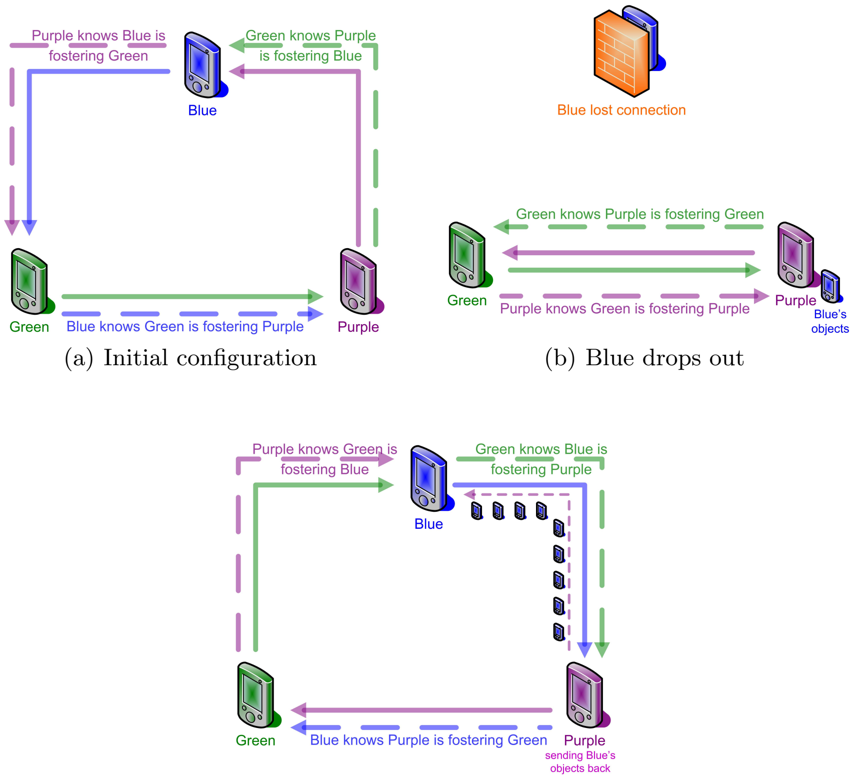
4.3 Peer drop-out handling

In this section, we will explain the sequence of actions that fostering agents take in case of other peer's drop-out. We will use Figure 2 as illustration. In the initial configuration, the local knowledge of a_{purple} is:

$$K(a_{purple}) = \langle F(a_{purple}, a_{blue}),$$
$$F(a_{blue}, a_{green}), Adopt(\emptyset)\rangle$$

and the local knowledge of a_{green} is:

$$K(a_{green}) = \langle F(a_{green}, a_{purple}),$$
$$F(a_{purple}, a_{blue}), Adopt(\emptyset)\rangle$$


Purple knows Blue is fostering Green
Green knows Purple is fostering Blue
Blue
Green
Blue knows Green is fostering Purple
Purple
Blue lost connection
Green knows Purple is fostering Green
Green
Purple knows Green is fostering Purple
Purple
Blue's objects
Purple knows Green is fostering Blue
Green knows Blue is fostering Purple
Blue
Green
Blue knows Purple is fostering Green
Purple
sending Blue's objects back

(a) Initial configuration

(b) Blue drops out

(c) Blue rejoins

Fig. 2. Peer dropping out and rejoining

When a peer (p_{blue}) drops out, all other peers will be notified of the peer dropping out event. a_{purple} will notice that a_{blue} matches the fosteree in one of the relationships ($F(a_{purple}, a_{blue})$) that it knows it's engaged in. And because the fosterer in that relationship is a_{purple} itself, a_{purple} will first store all a_{blue}'s objects, updating the adopting set to be $Adopt(\{a_{blue}\})$, then notify other fostering agents that the ownership of a_{blue}'s objects has been changed to a_{purple}. Knowing that a_{blue} was fostering a_{green}, a_{purple} will then take the initiative of reconstructing relationships by sending a_{green} a fostering request. Upon receiving such a request, a_{green} will send information about its own fosteree, which in this case is a_{purple}. The local knowledge of a_{purple} will now be updated to be:

$$\begin{aligned} K(a_{purple}) = \langle & F(a_{purple}, a_{green}), \\ & F(a_{green}, a_{purple}), Adopt(\{a_{blue}\})\rangle \end{aligned}$$

On the other hand, when a_{green} was notified that a_{blue} dropped out, it will notice that a_{blue} matches the fosteree in one of the relationships ($F(a_{purple}, a_{blue})$) that it knows locally. And because a_{purple} is its current fosteree, it foresees that

a_{purple} will be fostering some other peer after the relationship has been reconstructed, and therefore it sends a request to a_{purple} to get the updated fosteree of a_{purple}. After a_{purple} sends back the reply, the local knowledge of a_{green} will be updated to be:

$$K(a_{green}) = \langle F(a_{green}, a_{purple}),\\ F(a_{purple}, a_{green}), Adopt(\emptyset)\rangle$$

When a_{blue} rejoins, all peers will be notified of the joining event, and a_{purple} will notice that it is adopting a_{blue}'s previous objects. If a_{blue} requests to have its previous data back (it has the alternative option to rejoin as a completely new peer), a_{purple} will send a_{blue}'s objects back, and transfer the ownership of these objects back to a_{blue}. This scheme can be scaled up to an arbitrary number of peers.

5 The Games

As a proof-of-concept, three networked games that run on the Zaurus were developed based on FRAGme2004: a space shooter game called "SpaceBattle", a strategic tank game "BOOM!" and the Bomberman-like arcade game "RoboJoust". A screenshot of RoboJoust can be seen in Figure 3. They show that memory and communication bandwidth constraints are handled well enough by FRAGme2004 to allow fast action games on a limited device such as the Sharp Zaurus. Also, minimal knowledge of the framework was required, which allowed novice developers to focus on the gameplay design. "BOOM" and "RoboJoust" were developed from scratch by a group of eight fourth year students in under 50 hours.

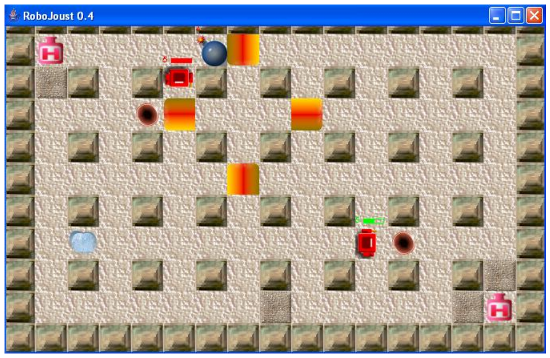


Fig. 3. Screenshot of RoboJoust

6 Evaluation of Agent-Based Fostering Mechanism

Based on the three games, we performed two experiments for the purpose of evaluating the fostering mechanism. Both experiments were performed on both

PC with LAN connections and Zauruses with WiFi connections. In the first experiment setup, we had a game in with five players are participating. Then we let randomly chosen players drop out one after another until there was only one player left. It was shown that each time after a player droped out, the remaining player could continue without any problem or noticeable lag. All the game-critical objects were kept intact by the framework, without the application programmer having to write any additional code. Then we let the formerly dropped out players join the game. It was shown that players could rejoin without experiencing problems, and each time when a player rejoined, the gameplay flow was not affected. In the second experiment, we took the first experiment one step further. Before all the peers who dropped out in the first round rejoin, we chose some random peers to drop out. Again, there wasn not a sign of performance impact on the game. In both experiments, the workload (number of fostees) of fostering agents were balanced nicely overtime. Since quantitative experiments are not really representing in this evaluation, we conclude through our qualitative experiments that the fostering mechanism works in the case of one peer dropping out at a time. Our solution will not work as nice if two peers are dropping at the same time. The circle will not be closed and will result in a path. The two peers at the ends of the path are not protected by the fostering mechanism, but fostering will still function for the peers on the rest of the path.

7 Conclusion

As we showed in this paper, the development of mobile P2P applications poses a number of obstacles, with object synchronisation, network failure and device limitations being the most significant. With FRAGme2004, we developed a system that tackles those problems using agent technologies and offers a reliable framework for P2P application development. By separating the application layer strictly from the framework infrastructure, FRAGme2004 allows developers to implement applications with minimal knowledge of the framework in a relatively short development period. The impact of network failure and peer dropout is now efficiently reduced by our agent-based peer fostering mechanism.

For future development, the range of FRAGme2004 enabled devices can be further expanded. The development took place on the Sharp Zaurus SL-C700, which runs Java Personal Profile. However, not too many mobile devices support Java Personal Profile. To make FRAGme2004 more widely useable, it needs to be ported to the Java Mobile Information Device Profile (MIDP). Also, Zaurus devices communicate via WiFi. But since the network traffic generated by most FRAGme2004 applications is not unmanageable, it is possible to make FRAGme2004 incorporate other types of less-capable wireless connections. The agent-based fostering mechanism can be improved to handle cases like two peers dropping out at the same time. The agents can also be enhanced to have more capabilities, for example, to take security and trust measures when communicating with other agents.

References

1. Akehurst, D.H., Waters, A.G., and Derrick, J. (2004). “A Viewpoints Approach to Designing Group Based Applications”, In Herwig Unger, editor, *Design, Analysis and Simulation of Distributed Systems 2004*, Advanced Simulation Technologies Conference, pp. 83-93, Arlington, Virginia, April 2004.
2. Babaoglu, O., Meling, H., and Montresor, A. (2002). “Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems”, *Proceedings of the 22nd International Conference on Distributed Computing Systems(ICDCS)*, pp. 15-22, Vienna, Austria, 2002.
3. Bruegge, B., and Dutoit, A.H. (2004). *Object-oriented software engineering: using UML, patterns, and Java*. Upper Saddle River, NJ, USA: Prentice Hall.
4. Chen, Y., Katz, R. H., and Kubiatowicz, J. (2002). “Dynamic Replica Placement for Scalable Content Delivery”, *IPTPS ’01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, Springer-Verlag, pp. 306 - 318, March 2002.
5. Cooper, B., Bawa, M., Daswani, N., Marti, S., and Garcia-Molina, H. (2003). “Authenticity and Availability in PIPE Networks”, *Future Generation of Computer Systems*.
6. P. Dasgupta (2003). “A Peer-to-Peer System Architecture for Multi-agent Collaboration”, *Advances in Soft Computing, (Proceedings of the 3rd International Conference on Intelligent Systems and Design Automation, Tulsa, OK)*, Springer-Verlag, pp. 483-492, August 2003.
7. P. Dasgupta (2003). “Improving Peer-to-Peer Resource Discovery Using Mobile Agent Based Referrals”, *Proceedings of the 2nd Workshop on Agent Enabled P2P Computing (co-located with AAMAS)*, pp. 41-54, Melbourne, Australia, July 2003,
8. Gerke, J., Hausheer, D., Mischke, J., and Stiller, B. (2003). “An Architecture for a Service Oriented Peer-to-Peer System (SOPPS)”, *Praxis der Informationsverarbeitung und Kommunikation (PIK)*, 2/03, pp. 90-95, April 2003
9. H. Homayounfar, F. Wang, S. Areibi (2002). “ Advanced P2P Architecture Using Autonomous Agents”, *CAINE*, San Diego California, pp. 115-118, Nov 2002.
10. JGroups Project, http://www.jgroups.org
11. Kato, T. et al. (2003) “A platform and applications for mobile peer-to-peer communications”,
    http://www.research.att.com/ rjana/Takeshi_Kato.pdf.
12. Lin, S.-D., Lian, Q., Chen, M., and Zhang, Z. (2004). “A Practical Distributed Mutual Exclusion Protocol in Dynamic Peer-to-Peer Systems”, *IPTPS04*, Springer-Verlag.
13. Margaritis, M., Fidas, C., Avouris, N., and Komis, V. (2003). “A Peer-To-Peer Architecture for Synchronous Collaboration over Low-Bandwidth Networks”, in K. Margaritis, I Pitas (ed.) *Proc 9th PCI 2003*, Thessaloniki.
14. Milojicic, D. S. et al.(2002). “Peer-to-peer computing”, *Technical Report HPL-2002-57*, HP Lab, 2002.
15. Pang, X., Catania, B. and Tan K. (2003). “Securing Your Data in Agent-Based P2P Systems”, *Eighth International Conference on Database Systems for Advanced Applications (DASFAA ’03)*, Kyoto, Japan, p. 55, March 26 - 28, 2003.