Dynamic Ad Hoc Coordination of Distributed Tasks using Micro-agents

Christopher Frantz, Mariusz Nowostawski, Martin K. Purvis

Department of Information Science, University of Otago, New Zealand

Abstract. The notion of μ-agents to develop complex software applications has been under active research interest for some time. Through improved organisational modelling μ-agents provide stronger support for decomposition and abstraction in decentralized applications. With the advent of the mobile application platform Android – which exhibits strong analogies to multi-agent system principles – we strongly believe that μ-agent-based modelling has become an increasingly attractive alternative. It can combine decentralized application development with the wide-ranging set of sensors and communication channels to foster both context-sensitivity and flexibility of applications. By integrating Android with the μ-agent concept mobile applications can put stronger emphasis on coordination of task-oriented agent organisations. As an example how this can facilitate the development of distributed applications, we describe an application for the field of "Unconferences" to dynamically schedule informal talks in an ad hoc manner. We model the central aspects of the application and show the advantages of our μ-agent-based approach. Finally, we contrast our approach to existing work in this field and suggest the consideration of μ-agents as an alternative to conventional object-oriented software development.

Keywords: Multi-agent Systems, Micro-agents, μ-agents, Unconferences, Android, Agent-based Modelling, Distributed Information Systems

1 Introduction

The increasing penetration of smartphones into consumer markets has made ubiquitous computing a reality. Beyond the traditional communication role, smartphone capabilities are increasingly used for context-sensitive coordination of tasks and location-based services, supported by a number of sensors (e.g. GPS, accelerometer, gyroscope) and a wide range of communication channels (e.g. NFC, WiFi, Bluetooth).

As a result, many of the applications running on those devices are inherently complex. They must integrate multiple sensors and communication channels with different characteristics and behaviour (e.g. blocking/non-blocking communication) and often rely on concurrent and asynchronous communication patterns within the application.

The modern mobile application platform Android, reflects this understanding and allows flexible modelling of mobile applications from application components, structuring applications from asynchronously interacting front-end, back-end, content-related and context-related elements. Although the Android architecture simplifies complex application development, the application components do not provide enough flexibility to model more fine-grained functionality beyond the high-level application layout enforced by the Android architecture.

To improve the modelling capabilities and also allow a flexible concurrent layout of applications, we present our integrated approach of agent-based technology and the Android platform, which we call 'μ-agents on Android' (MOA). We see μ-agents as a lightweight approach to model concurrent and potentially distributed applications from autonomous entities. The micro-agent concepts that we introduce also provide a comprehensive organisational meta-model to allow the strong level of decomposition that Android application components cannot offer "out of the box".

We begin by introducing the notion of *application components* in Android and suggest some improvements to establish the development of a more flexible Android application framework. Following this, we introduce the concept of μ-agents and elaborate some of their key modelling features. After this, we describe how both technologies have been integrated as MOA.

To clarify the potential of our approach for dynamic information systems, we describe an application which benefits from those capabilities, and demonstrates well how to address coordination concerns for ad hoc talks at "Unconferences" [10]. Unconferences represents a comparatively young phenomenon and rely on a strong degree of decentralized organisation.

2 Android and μ-agents

2.1 Android application components

The application platform Android [4], developed by the Open Handset Alliance under the lead of Google, is a Linux-based software stack providing a uniform approach towards writing smartphone applications. The Android application design principles make even elementary applications and their components interchangeable by the phone user. Android's software stack includes the Dalvik virtual runtime environment, which allows for application development using the widely known Java syntax. On higher levels of the software stack, capabilities are organized in a task-oriented manner using the notion of managers. For example the *LocationManager* provides support for location awareness and location-based event handling.

To model applications, Android supports the notion of *application components*, which can be used to model a basic template of an application by its functional characteristics. Android comes with four different application component types: *Activities* run in the foreground, directly interact with the user, and are of rather short-running nature. *Services* run in the background, are long-running and often maintain the core application state, as their functionality is

least likely to be interrupted. *Broadcast receivers* represent an event subscription mechanism used both for system events (such as receiving text messages), and *Content providers* act as an abstraction layer for any persistent data source.

All these components (apart from Content providers) are linked by so-called Android *intents*, which represent a message structure encapsulating abstract request specification plus extra key-value pairs (extras). Intents are asynchronously executed and allow dynamic binding of application components at runtime. As a consequence, applications are constituted by the combination of application components and the intents registered with particular components, which are documented as an application manifest. The application manifest is a XML file describing application components and permissions, among other project-specific details.

2.2 Modelling constraints of Android application components

Although Android provides a number of building blocks for modular application composition, we feel that the framework still constrains application development, in particular, the structural decomposition: Android applications often result in a rather static structure consisting of the four provided coarse application component types. From a modelling perspective, the provided modelling artefacts are considerably limited and only allow simple application templates, structuring applications by foreground and background activity, as well as context (e.g. events) and content storage. Applications thus show uniform structure and are typically modelled using conventional object-oriented programming.

This is not a fundamental flaw, but Android does not take the full step to ease the task-oriented modelling aspects such as a more fine-grained decomposition of functionality into more primitive and specialized application components while dealing with low-level aspects such as concurrency. To provide further decomposition of application functionality in Android, application developers need to fall back to Object-orientation. Along with the weaker abstractions developers also need to deal with aspects such as thread handling and loose the built-in capability for asynchronous message passing on this level.

In order to provide a better and also more uniform modelling support, we provide more fine-grained application-oriented modelling mechanisms based on software agents [11]. This way, application architects and developers can use the same conceptual and modelling tools on higher and lower levels of abstraction.

2.3 μ-agents

The idea of using small-size agents to compose application functionality goes back to the late 1990's (e.g. [3], [8]) and is motivated by the suggestion to engineer comprehensive applications with numerous entities having narrowly specialised functionality in a cascading, hierarchical structure. The use of μ-agents, in the approach taken here, is similar to the one of object-oriented programming in that μ-agents can be recursively constructed from μ-agents. This allows effective decomposition in fine-grained entities and also allows abstraction from

lower level functionality while constantly thinking in an agent metaphor. What makes μ-agents different from objects is that they pursue their own objectives (e.g. composing their functionality from other agents) and they do not only act in a reactive manner. Agents communicate asynchronously instead of blocking message calls. Internally however, the application developer is not tied to a particular interaction mechanism. μ-agents can thus largely differ in their internals but commit to common communication mechanisms. As a result, applications developed with this μ-agent concept in mind effectively exploit concurrency without the need to deal with thread-related aspects (such as in many object-oriented languages). At the same time, they deliver a performance which is comparable to existing object-oriented systems. With this motivation in mind, we implemented a μ-agent platform we call μ^2.

The μ-agent meta-model applied here is shown in Figure 1 and discussed in the following paragraphs.

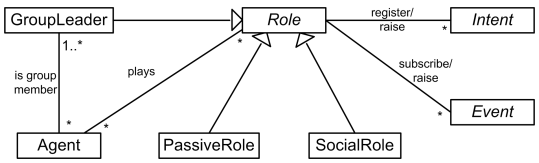

Fig. 1. Core relationships in μ^2

μ-agents are lightweight autonomous persistent entities which show reactive – and, potentially, proactive – behaviour and can efficiently interact in synchronous or asynchronous manner. Organisational aspects come into play when considering that μ-agents are designed to play one or more user-defined roles which are dedicated to fulfil *applicable intents*. Roles represent a set of potentially interrelated behaviours and may require necessary capabilities by a μ-agent in order to be played.

μ-agent *intents* (as opposed to *Android intents*) represent the notion of intentions, i.e. they are abstract request specifications. Roles registering *applicable intents* need to provide mechanisms to satisfy those *intents*. μ-agent *intents* can be raised by any agent on the platform and are automatically bound to a satisfying agent (if any agent can fulfil those).

Apart from the interaction mechanisms, organisational modelling capabilities are realized using the special '*Group Leader Role*'. Group leaders control an arbitrary number of sub-agents and serve the purpose to propagate control messages from the agent platform, as well as to structure the agent society. This allows flexible partitioning by functionality aspects. Agents playing the group leader role typically use sub-agents to compose their own functionality. Sub-agents playing this role can themselves have sub-agents which allows an agent organisation of arbitrary depth. The advantages of this organisational modelling mechanism not only include the strong degree of decomposition down to a very

fine-grained level, but the organisational modelling also allows the definition of abstraction levels to hide details from the application developer.

Apart from the *group leader role*, other specializations to be mentioned are: *social roles* which support asynchronous communication between agents; and *passive roles* which only rely on synchronous communication. The purpose of the latter role is to provide the most fine-grained functionality without the performance penalty introduced by asynchronous message passing. The notion of events is used to provide event subscription capabilities, which are of particular concern when coordinating state in decentralised systems.

At this point we want to draw the reader's attention to one aspect, which is the similar role and denomination of intents. In both, Android and μ^2, *intent* instances represent request specifications for particular tasks. However, the intent structures of Android and μ-agents are not compatible. The dynamic resolution mechanism is similar, as in both cases they allow a loose coupling between application components (in the shape of *IntentFilters* in Android and, respectively, intent-based dynamic binding (via *applicable intents*) with μ-agents).

Some of the core principles of μ-agents such as loose coupling and asynchronous communication are essential part of mobile application platforms such as Android. μ-agents offer a more unified yet consistent architecture and extend those modelling mechanisms with an explicit task-oriented organisational perspective that is not part of Android's concept.

2.4 Integrating μ-agents with Android

To make our vision of μ-agents a reality, we have ported our μ-agent framework μ^2 to the Android platform. A key to this operation is the open communication principle of Android that enables the integration of μ-agents in Android by translating the differing intent data structures at runtime.

Android intents have a static class structure but can hold dynamically-typed content. The structure of μ-agent intents is not predefined and leaves the developer with a wide range of options; the μ-agents intent structure also includes the capability to integrate methods. The core of μ-agents on Android (MOA) – which is schematically visualized on Figure 2 – is thus the conversion of the different intent types at runtime in order to allow direct interaction between Android components and μ-agents.

From an architectural perspective this integration is achieved by encapsulating the μ-agent organisation into an Android service which is addressable by other legacy Android application components. The μ-agents wrapped in the Android service can directly address arbitrary Android application components, pass data, and receive responses from addressable application components. Application components themselves can address μ-agents by using a predefined intent type that allows the specification of an agent name. Intents sent from Android not using this particular type are raised as events that μ-agents may or may not have subscribed to (and eventually react to).

To allow the direct access to Android functionality that is not addressable via intents (such as the GPS functionality of the LocationManager, or the Sms-

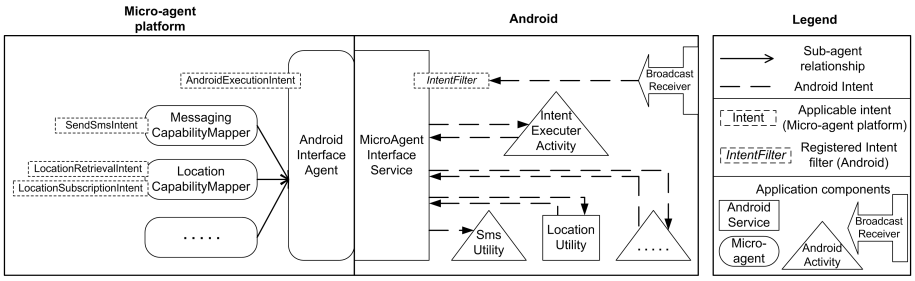

Fig. 2. μ-agents on Android Architecture

Manager), this functionality is remodelled with dedicated μ-agents and offered to other agents in the μ-agent organisation. Through flexible dynamic binding, this model allows agents to use Android functionality in a consistent manner without concern about where the actual functionality is executed. This gives MOA – compared to the desktop version of the μ-agent platform (μ^2) – an expanded functionality set, as μ-agents can thereby make use of the wide range of sensors, sensory information and context (e.g. retrieve location information) and communication channels (e.g. write SMS text messages).

3 Mobile application for Ad hoc meetings at Unconferences

In order to show the potential of μ-agents to complement mobile application development, we describe a practical application that exploits some of the characteristics offered by μ-agents.

3.1 Application context

Unconferences represent a new phenomenon that has emerged as a counterpart to conventional conferences that require intensive and expensive quality assurance mechanisms, organisation, committees, peer review, and publication of presented papers. The idea of an unconference is based on the perception that the collective knowledge of the audience is likely to be more extensive than that of the scheduled speakers; apart from typically time-constrained question sessions this knowledge is hardly used at conventional conferences. In short, many productive and inspiring talks actually seem to happen in hallways, by spontaneous interaction between a few conference participants instead of well-prepared and time-constrained presenters.

Apart from the contrary philosophy of this grass-roots approach, Unconferences also heavily rely on the widely adopted social media of the Web 2.0, such as blogs and social networking sites. Depending on the particular nature of the unconference, topics of concern can be suggested and rated via Wiki-based websites before the event, or are negotiated on the spot, at the venue. During the

actual unconference, constant multi-channel interactions take place; apart from the presentations – which can eventually transform to discussions – participants constantly share ideas and inspiration via social media[1]. Input can also come from people without physical presence – for example Twitter, Facebook or the like. As a general rule, every person going to an unconference should expect to at least give his opinion on a topic of concern, or communicate his or her own idea.

Talks can spin off into smaller groups of people discussing either niche topics or specialist aspects. Those groups typically vary in size, as well as in composition with regards to personal attributes (such as rhetorical abilities and confidence) and skills (e.g. skills to use social media). Persons providing bright and innovative ideas may not necessarily have the soft skills of comparable quality. This makes the presence of a person with moderation skills beneficial, in order to convene a session and encourage participation of less dominant attendees, rather than leaving this entirely to the uncertain and unguided dynamics of the group.

Although general purpose social media can sometimes be effective to support the organisation of those dynamic sub-events, we think that dedicated software can outperform and improve the basis for productive and balanced interactions while capitalising on experiences from previous meetings. Another seemingly simpler but important logistic aspect is the coordination of concurrent meetings with differing numbers of participants and a limited number of rooms at a venue. Depending on the number of participants and the available time slots, the system can effectively coordinate spontaneous scheduling of rooms.

We use this scenario to show the potential of the agent-based modelling approach for mobile applications.

3.2 Mobile application for ad hoc organisation of spontaneous talks

The use of μ-agents as a modelling paradigm on mobile devices supports a fairly broad spectrum of modelling needs. Agents represent the interacting entities, intents are used to express particular requests, and events are used to inform an unspecified number of recipients. Apart from the support in shaping group structure, group size is also of concern. To facilitate this, the application demands more static information about the location of the venue in order to function productively.

The overall application is thus divided into an agent platform holding an agent that deals with room assignments and the management of topics, while an arbitrary number of further platforms – running the Android-based client part of the application – can actively use the system.

The following Figure 3 visualizes the static structure of this application. The notation used shows the organisational decomposition of applications into agents and emphasizes their hierarchical relationships. Agents are annotated with intents they are able to process (i.e. fulfill) and events they want to subscribe to.

[1] An exemplified overview on collaborative tools involved in Unconferences is provided by Crossett et al. [2]

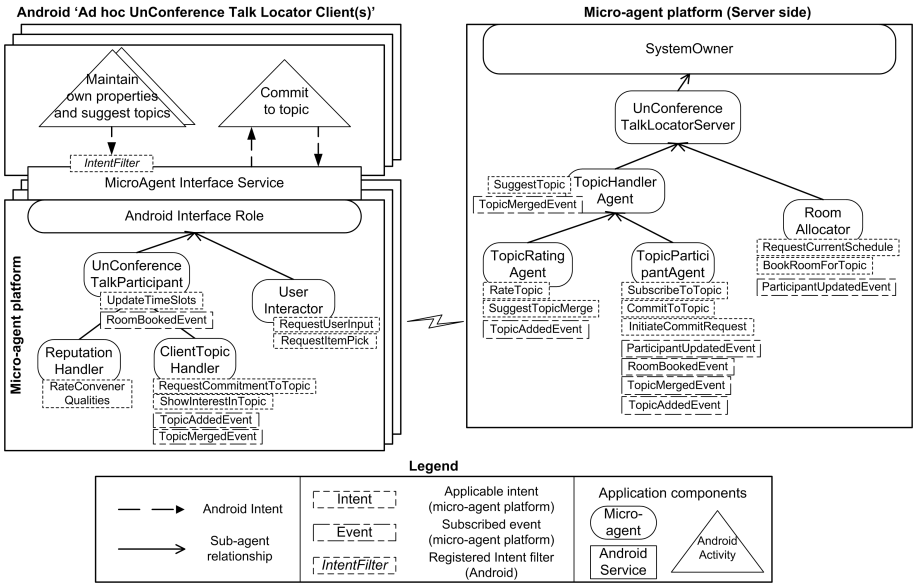

Fig. 3. Static Structure of application

For the description of the agent internals, we use message-centric Coloured Petri Nets (CPN) [7]. CPNs are a good way to capture the dynamic aspects within the different agents – and thus implicitly the dynamic structure of the entire application. With 'message-centric' we refer to the fact that μ-agents generally react on incoming messages (be it intents or events delivered in messages – or even custom messages without intents or events), which modify their state (visualized by places representing state repositories), and eventually produce outgoing messages. On this level, internals of μ-agents are thus fully represented in terms of message flow.

Instead of describing all application details in the form of diagrams, we provide below an overview of the overall application in a narrative manner. For a selected μ-agent we describe the internal message flow to show the intent-based loose coupling as well as to emphasize the decentralized character of application composition using μ-agents.

The application allows clients connected to the server part of the application to see, rate, and suggest new topics (by invoking the *SuggestTopic* intent resolving to the *TopicHandlerAgent* on the server side). New topics are immediately available to all clients and potential participants can suggest their merging (if topics seem related) – using the *SuggestTopicMerge* intent – and, most importantly, subscribe to topics of interest (using the *SubscribeToTopic* intent). Subscribing to topics does not automatically imply participation, but it ensures that the subscriber is notified about all modifications (e.g. via the *TopicMergedEvent* registered to the *ClientTopicHandler*) and contacted once another subscriber

requests commitment to the topic. All subscribers then need to accept or decline this request, and in consequence a suitable room (according to the number of committing participants) and time is scheduled, with preference for topics of high rating.

This base set of functionality gives rise to strong dynamics that can unfold, given a high number of users and a wide range of different interests.

Participants can not only rate topics but also rate other attendees with regards to their mediation qualities, which is of help to suggest potential moderators for particular talks. The merging of topics can be suggested by any client connected the server, but demands for confirmation by the ones who originally suggested the affected topics.

To give some insight on the internals of μ-agents, we take a representative diagram for a server side agent entity.[2]

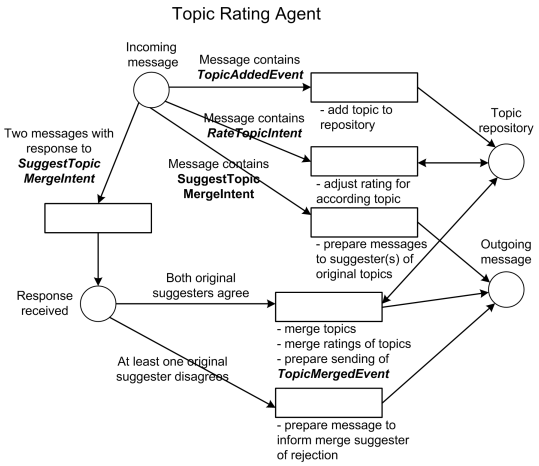

Fig. 4. TopicRatingAgent on the server side

The *TopicRatingAgent*, whose operation is depicted by the CPN in Figure 4, resides on the server side of the application, and handles the aspects of adding a topic, rating it, and merging topics. Upon receiving a message, the agent checks it for contained intents or events. It reacts to messages containing a *TopicAddedEvent*; the topic is saved to the local topic repository (indicated as a Petri-net place in the diagram). Messages containing a *RateTopicIntent* carry information about a rating done by a client, and results in an update of the topic rating on the server side. Upon receipt of a *SuggestTopicMergeIntent*, the

[2] In contrast to the static structure shown in Figure 3, where intents and events can be disambiguated by the graphical notation, intents and event names shown in the CPN diagram in Figure 4 are written in bold italics and additionally carry the type name as suffix to facilitate the interpretation.

request is forwarded to the original suggesters of the topics concerned. Upon their response – and the case that both original suggesters approve the merging – the topics are merged and an according event is sent out in order to notify all subscribers (which includes the clients but also two other μ-agents on the server side as seen from the static application diagram in Figure 3). If they disagree, the client who suggested the merge is notified about the rejection of his request.

Note that Figure 4 shows the message-centrism of μ-agents which enables a streamlined modelling of core functionality with a low threshold between design and implementation. The only aspects not captured by the diagrams are the intent internals. In fact, the internals of intents are only known to the μ-agents that either raise or process those. This principle hides unnecessary information from non-affected agents and thus avoids the unintented or accidental interpretation by unrelated agents. As briefly mentioned in section 2.3, the dynamic binding (matching of raised intents against registered *applicable intents*) between different agents is realized by the underlying agent platform in an asynchronous manner and is transparent to the application level. Still, if of interest, agents can formulate custom messages addressed by agent name or using various addressing patterns (e.g. broadcast or rolecast).

Although only briefly shown through the above example, the MOA-based interaction mechanism allows a strong embedding of μ-agents with contextual information from sensor information (e.g. GPS, accelerometer, gyroscope) and use of communication channels (e.g. WiFi, Bluetooth, SMS). This opens the MOA approach for a wide range of smart context-embedded applications and eases extension of those. An example would be the suggestion of topics via SMS or the suggestion of informal meeting points for a smaller number of participants based on the proximity of all attendees. On the server side the obvious potential is to allow remote users to suggest topics (e.g. via the Web) and keep those informed about the current schedule of events.

4 Related work

Looking at related work in the field of Unconferences is rather difficult, as both the process of organisation (i.e. often decentralized organisation, at most a central wiki) as well as the output (i.e. no formal proceedings) is informal. To the authors' knowledge, the only comprehensive description of a collaborative toolset for the operation of Unconferences is provided by Crossett et al. [2][3]. From the available literature, tools used to organise such venues are only weakly integrated (e.g. wikis, blogs, twitter) and rather heterogeneous in their nature. Given this context, a more unified but still decentralized and extensible μ-agent-based infrastructure seems helpful to foster a more efficient ad hoc organisation of events and management of limited resources (e.g. rooms and time) during Unconferences.

Taking a look at Android-based agent platforms, a number of those are derived from desktop platforms. An Android derivate of the prominent multi-agent

[3] A further resource of information is the Unconference blog of Kaliya Hamlin [6].

platform JADE, JADE-Android [5], is one of the most prominent examples, and essentially is a subset of the desktop version of JADE. It allows the execution of one agent. For the distributed operation it relies on at least one connected desktop version of the platform.

Another, quite elaborated, approach is JaCa-Android [9] which identifies agents as first order entities and interprets all handled objects (e.g. GPS coordinate) as so-called artifacts. The internal architecture of the platform relies on an implementation of the belief-desire-intention (BDI) model.

Agüero et al. [1] are among the earliest adopters of agent-based software development for Android. They implemented an abstract agent platform model (Agent Platform Independent Model (APIM)) on Android, which largely focuses on agent internals; organisational modelling aspects are not covered. The implementation extends the Android application components directly, thus building an agent platform 'on top of Android', in contrast to the interfacing approach taken by MOA (see section 2.4).

None of the above mentioned approaches encourages the use on many lightweight μ-agents, but instead, requires rather a limited number of more intelligent agents. This is a useful approach to offer smart applications, but hinders strong degrees of decomposition and flexible reconfiguration in cases involving newly introduced agents and changing agent capabilities at runtime.

5 Conclusion

The use of μ-agents, as a general application modelling tool, provides a strong degree of decentralization, effective (hierarchical) decomposition and transparent interoperability between application components (i.e. external dependencies of agents are captured using message-centric diagrams as shown in section 3.2).

The possibility of employing more primitive (bare-boned) μ-agents facilitates the decomposition to a fine-grained level. μ-agents can be used where the instantiation of yet another Android application component would be inefficient. MOA-based applications are open to extension and allow developers to capitalise on implementation efforts by reusing existing intents exploiting the dynamic binding mechanism. In principle, this mechanism is available across various applications and thus allows a better contextualization not only of isolated applications (by integrated sensors and communication channels) but over the entire application landscape. Applications can not only rely on other applications (such as those offered with Android's own development approach), but also on elements of other applications (i.e. μ-agents of MOA applications, or individual application components of traditional Android applications).

Using the example of Unconferences – an inherently dynamic and decentralized phenomenon in itself – μ-agents seem suitable to cope with the fluid and heterogeneous nature of those events, and offer useful mechanisms of coordination while maintaining the necessary flexibility. Relating it to the given application scenario, μ-agents allow the handling of unexpected events, such as the unexpected occupation of a scheduled location, or the 'no show' of participants.

Also, bearing in mind the wide set of different tools used to run Unconferences, the agent principles are adequate to handle the coordination of many heterogeneous information sources. As such the application described in this context is only a starting point but provides enough understanding about the mechanics of μ-agents to inspire further, more context-dependent functionality.

We hope that the μ-agent modelling principles find adoption as an extension to the wide-spread use of object-oriented modelling, and we think that the lightweight framework presented in this article provides a low threshold to achieve this. The development around Android has shown how open software and open communication principles foster a diverse application ecosystem. μ-agents can extend those principles, not only to ease communication between devices and applications, but promote the ad hoc organisation of more elementary application elements, treating the mobile device as their natural open environment.

References

1. J. Agüero, M. Rebollo, C. Carrascosa, and V. Julián. Does Android Dream with Intelligent Agents? In J. Corchado, S. Rodríguez, J. Llinas, and J. Molina, editors, *International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008)*, volume 50 of *Advances in Soft Computing*, pages 194–204. Springer Berlin / Heidelberg, 2009.
2. L. Crossett, J. Kraus, and S. Lawson. Collaborative tools used to organize a library camp unconference. http://eprints.rclis.org/bitstream/10760/12831/1/Preprint-CollaborativeToolsUsedtoOrganizeaLibraryCampUnconference.pdf, March 2009. Accessed on: 24th August 2011.
3. J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. *Third International Conference on Multi-Agent Systems (ICMAS '98), IEEE Computer Society*, pages 128–135, 1998.
4. Google. What is Android? http://developer.android.com/guide/basics/what-is-android.html. Accessed on: 24th August 2011.
5. D. Gotta, T. Trucco, M. Ughetti, S. Semeria, C. Cucè, and A. M. Porcino. JADE Android Add-on Guide. http://jade.tilab.com/doc/tutorials/JADE_ANDROID_Guide.pdf. Accessed on: 24th August 2011.
6. K. Hamlin. Unconference blog. http://www.unconference.net. Accessed on: 24th August 2011.
7. K. Jensen. *Coloured Petri-Nets – Basic concepts, Analysis Models and Practical Use*, volume 1. Springer-Verlag, 1992.
8. M. Nowostawski, M. Purvis, and S. Cranefield. KEA - Multi-Level Agent Architecture. In *Proceedings of the Second International Workshop of Central and Eastern Europe on Multi-Agent Systems (CEEMAS 2001)*, pages 355–362. Department of Computer Science, University of Mining and Metallurgy, Krakow, Poland, 2001.
9. A. Santi, G. Marco, and A. Ricci. JaCa-Android: An Agent-based Platform for Building Smart Mobile Applications. In *In Proceedings of LAnguages, methodologies and Development tools for multi-agent systemS (LADS-2010)*, 2010.
10. Wikipedia. Unconference. http://en.wikipedia.org/wiki/Unconference. Accessed on: 24th August 2011.
11. M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.