Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 11, Number 1, pp. 63–71. http://www.scpe.org

GROUP-BASED INTERACTIONS FOR MULTIUSER APPLICATIONS

CARMEN MORGADO*, JOSÉ C. CUNHA*, NUNO CORREIA*, AND JORGE CUSTÓDIO*

Abstract. We propose a group-based model for the development of interactive applications where multiple users communicate and collaborate with each other and share information. The model defines the organization of a system in terms of entities and groups, each one with a well-defined interface, that mediates all the interactions. Entities can dynamically enter and leave groups, and distinct forms of communication among the group members are allowed, namely based on messages, events and a shared space. This model provides support for explicit groups and a novel concept of implicit groups, which allows the system to automatically adapt the group membership according to the dynamic changes in user profiles. We discuss applications of the model and illustrate its use to support example scenarios in local communities.

Key words: group-based model, interactive applications, shared spaces

1. Introduction. The recent increase in the development of applications exploring user mobility, interaction and information sharing, motivates the need for more expressive abstractions, models and mechanisms to enable a transparent and flexible specification of group-based interactions between users. The relevance of group models [3] has been recognized in multiple domains like collaborative work systems, multi-agents and more recently interactive Web applications for social networking. We describe a model that supports group abstractions to dynamically manage and organize small and medium location based communities of users. The MAGO model—Modeling Applications with a Group Oriented approach [11, 12], and its computing platform facilitates the organization, access and sharing of contents by multiple users, and enables their dynamic interactions. By considering groups as confined spaces for user interactions, the model allows to exploit geographic proximity of users, their associated locality, privacy and access control issues. We identify three main contributions of this work:

1. the identification of the main requirements of applications that led to the development of our model;
2. the possibility to dynamically adjust and control group membership, as well as to define policies to guide the implicit and dynamic group formation according to user profiles;
3. the support of a combination of forms of interactions that are required by real life applications, including peer-to-peer, multicast, asynchronous event notification, and access to a shared space, internal to each group.

In section 2 we identify requirements found in current emerging interactive applications. In section 3 we present the MAGO model, main concepts and functionalities. In section 4 the system architecture is described. In section 5 we describe application examples for small place-based communities. Finally, conclusions and current research directions are presented.

2. Motivation and requirements. We identified a diversity of scenarios where groups can be used to ease the modeling and development of applications where users form groups, to establish interactions and to share information. Examples arise in working places, travels, tourism spots, shopping centers or any other places where "social interactions" may exist. Places like schools campus, museums, shopping centers, restaurants and coffee shops, or airports, are examples where the users may tend to use groups as a way to interact, share information or to achieve a common goal. In most of these places people stay there for a limited period of time and are open to interact with each other. Another requirement is the need to structure the users space, so the services offered can be personalized based on the users preferences.

Similar requirements are also found in applications that manage user collaboration, connections and context awareness [10]. Examples of such applications are on the social software domain Facebook, MySpace or Hi5, and applications promoting the direct communication among users like ICQ, Skype, MSN Messager or Google Talk. On the information sharing domain there are general purpose applications like Flicker, Ringo or YouTube, or domain specific applications like canal*ACCESSIBLE [15] to help the navigation of people with locomotion disabilities using photos captured with mobile devices, or CONFECTIONARY [14], to create narratives based on user content, such as photos and videos. Also many applications exploit user mobility, allowing users to upload and access contents using handheld devices. Context awareness is a key issue in such applications [10]. In

*CITI, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal, (cpm@fct.unl.pt, {jcc, nmc, jorge.custodio}@di.fct.unl.pt).

the tourism area we can find many examples of interactive applications, like Tech4Tourism [2] where the tourists can navigate through the maps gathering contextualized information (audio, video, photos or explanation texts) based on their actual position and personal interests; InStory [4] allows the users to have more interaction with the system and with each other, sharing information and grouping themselves based on different activities; or Marble [13] where the museum visitors can have access to information concerning the art objects nearby. Although all the above can benefit in some way from the use of group based concepts, none of them however, has the possibility of dynamic creation of groups, or offers an integrated platform for communications and data sharing.

Concerning the development of models that meet the needs for application structuring, information sharing and the establishment of interaction among the application entities, there is a diversity of proposals. Group models have been proposed for application structuring since the 1980's, in the context of distributed systems [16], and have been further explored at distinct levels of abstraction and with different goals, namely concerning CSCW/groupware system [17] and multi-agent systems.

Currently the group concept remains a significant approach to support the organization of an application in terms of collections of dynamic entities, each group defining a confined space for interactions among group members. From the above mentioned applications requirements, different forms of interactions should be supported, with distinct semantics concerning synchronous versus asynchronous behavior, including event-based, direct versus indirect interactions, and supported by shared or distributed memory.

However, in most of the proposals, groups are associated with forms of message-based communication among the group members. Such existing group systems already provide significant guarantees concerning, for example message delivering orderings, the management of group view consistency and the handling of failure situations. Such guarantees greatly contribute to ease distributed application development.

In order to provide other forms of interactions, other proposals have considered forms of asynchronous and indirect communication between entities based on shared spaces, easing the dynamic and loosely-coupled interactions among asynchronous entities. One significant proposal is the Linda model [9] that has been used in a large diversity of projects, to support coordination of distributed processes based on a shared tuple space.

In order to support some level of structuring, at the application level, some proposals have been made [18] allowing an application to be organized in terms of multiple tuple spaces. The shared tuple space concept remains a relevant approach to model information sharing functionalities, in two distinct dimensions: (i) to allow access to persistent information repositories, that maintain information on the users identification and profiles, and also access to contents; (ii) to allow anonymous asynchronous and indirect interactions among the application entities through the shared tuples, during execution.

In our proposal, we explore a distinct approach that relies on the group concept, which is interpreted as a confined interaction space, integrating in an unified way multiple forms of interactions, in order to provide improved flexibility and expressiveness to the applications. The model ensures the consistency of the group views, with respect to the group membership modifications, and the communication events (message multicast and access to the shared space).

A distinctive goal of our proposed model is to offer, in the same platform, the following functionalities:

- Dynamic management of users and their profiles;
- Flexible and dynamic creation of explicit and implicit groups;
- Dynamic group membership;
- Transparent access to personal and group information;
- Access to external information systems for manipulation of different kinds of media;
- User interactions and task coordination.

3. The MAGO model and characteristics. As a solution to the above identified problems we propose the MAGO model [11, 5] which has the following main characteristics:

- Users are represented as elementary entities and are modeled as Java objects;
- Groups are units for application composition with a well-defined interface. An Universal Group includes all entities and groups defined in the system. Besides, each entity can belong to multiple (explicit ou implicit) groups;
- Groups are dynamic, allowing multiple entities entering and leaving during a group life cycle;
- A group allows distinct forms of communication, namely, direct point-to-point, multicast (messages and asynchronous events), and shared memory;

This model relies on the transparent management of the consistency of views that are observed by the group members, concerning the events related to messages, events, and access to shared state, and event causality. The MAGO model offers, to the developers, a set of primitives and services specially designed to support group interactions within a dynamic environment. The primitives fall into three main classes, that we succinctly describe in the next subsections.

3.1. Entities and groups. This includes primitives for managing entities and groups and dynamic group membership. When creating a new group its defining characteristics are specified including: name; admission policy; maximum number of members admitted; type of group; activation and deactivation deadlines. In order to model situations of disconnected operations of entities that are group members, an entity can be put in an online/offline state. When online, the entity can interact with the members of the group; when offline, the entity although still belonging to the group, cannot be accessed.

3.2. Communications. To reflect the different kinds of possible interactions the model combines three forms of communication within a group space:

- Direct: point-to-point communication, based on a direct call to an interface method of an entity or group.
- Events: based on a publish-subscribe mechanism [7], and is used to multicast messages inside a group space. Group members can subscribe to any event type that has been advertised on the group, and receive a notification on event publication. An event can be associated to messages or to changes on internal structures.
- Shared space: a form of communication that supports a shared space associated to each group. This mechanism is based on the concept of a Linda [8] tuple space with corresponding primitives to insert (*update*), remove (*get*) and consult (*consult*). Additionally we have defined a primitive (*find*) that allows the search of multiple tuples that match the search parameters specified. Each tuple has a name, owner information, the type and application specific information. The model uses the event-based mechanism to allow the notification of group members, about changes on the shared space.

3.3. Implicit groups. MAGO also provides a higher level group concept that enables the definition and automatic management of groups based on user properties and attributes (*implicit groups*). The implicit group membership can vary over time and is based on the users currently on the system, their characteristics, and the rules previously defined by the application at group creation time. Such rules are simple conjunctions of user attributes and their satisfaction is evaluated by a search engine, always looking for candidate members that match the specified rules.

The action of joining an implicit group occurs in three distinct situations:

- when a new implicit group is created and its characteristics are defined;
- when a new entity enters the system, and specifies its attributes;
- when an entity changes its attributes (this last action can also lead to an implicit action of leaving an implicit group).

To support the above model functionalities we developed a system architecture, that is described next.

4. System architecture. A layered architecture supports the proposed model as shown in figure 4.1. The MAGO architecture relies on a low-level Java API—JGroupSpace [6], which is an extension to JGroups [1], and offers primitives for group management, message and event multicast, and an implementation of the Linda tuple space. The MAGO model offers a higher level semantics for interactive applications concerning messages, event publication and subscription, access to the shared tuple space, and also provides the concept of implicit groups and an interface to an information system.

An external information system can store the contents produced and shared by the users, and also information related to user and group attributes, which is used to manage implicit groups. The information system offers the possibility of data persistence related to the system organization and contents information. Our system offers an interface that supports access to the basic functionalities of the information system (see figure 4.2). The current prototype supports access to an ORACLE Database system.

The model was implemented on an object oriented platform, where the applications are responsible for the instantiation of elementary entities (objs) and then have access to the MAGO functionalities/primitives (figure 4.3).

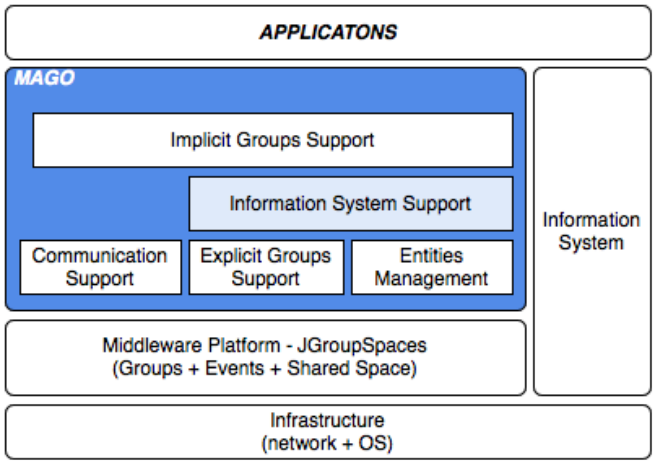

FIG. 4.1. *System layers and modules—MAGO components are included in the gray area in the figure. The bottom layer offers basic communication and system operation support, responsible for connections and management between devices. Middleware layer is responsible for group communications, events and shared space. Information system gives access and manages persistent data manipulated by the system.*

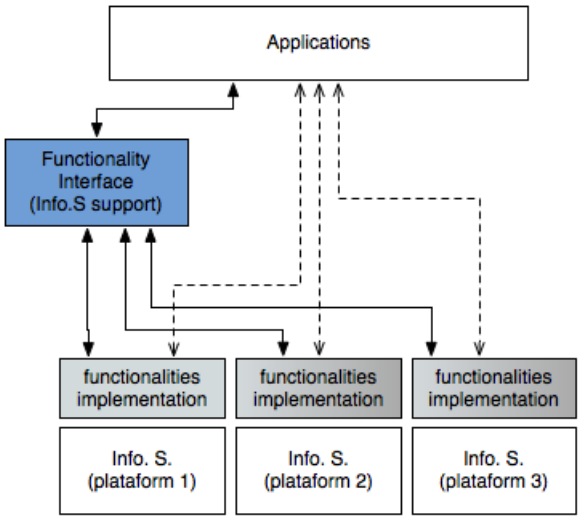

FIG. 4.2. *Interaction between applications our interface and the information system.*

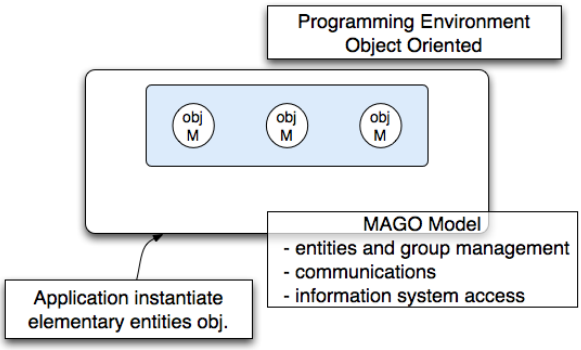

FIG. 4.3. *Interaction between applications our interface and the information system.*

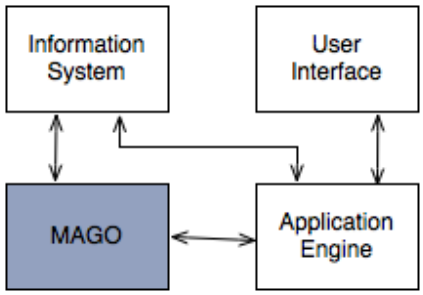

FIG. 5.1. *Applications basic modules.*

A working prototype of the MAGO architecture and an user interface giving access to the model primitives were implemented in Java, and used to support the experimentation with real applications (a detailed description can be found in [11]).

5. Application examples. The MAGO model promotes on organization where an application is based on four building blocks (figure 5.1): user interface; application engine; information system; and the MAGO module.

This organization is independent of the functionalities offered. Basically applications need to manage:

- the registration of users within the system;
- the users and groups profiles;
- the creation, remotion and group membership;
- different kinds of communication within a group space;
- the production and access to shared data;
- the coordination of tasks within a group space.

In order to assess the adequacy of the model functionalities and its usability, we experimented with already existing real world applications. With this experimentation we illustrate how our system can be used to model and extend functionalities in existing applications. It is also illustrated how the system is easily integrated into already developed architectures.

There is a diversity of scenarios where grouping of entities can be used to ease the modeling and development of applications. As mentioned before, examples of such application scenarios arise in work places, travels, tourism spots, shopping centers or any places where we have "social interactions".

Within these environments users usually want to form groups (in an explicit or implicit mode), to establish interactions and to share information. Places like a tourism spot or a school campus are examples of places where the users may tend to use groups as a way to interact, share information to achieve a common goal, to play a game or simply as a way to meet people that share similar preferences.

5.1. Tourism scenario. We describe how MAGO supports a tourism scenario in a local community where groups are used to facilitate user activities and interactions in a cultural heritage setting. This work extends a real world application previously developed in the InStory project [4]. Tourism applications usually deal with interaction between multiple users, information dissemination and content creation and sharing. In this scenario many users have mobile devices, for example PDAs, cell phones or small laptops and are likely to interact with others to establish groups, and share information and content like photos or videos. Based on the existing InStory architecture we integrated our model as a way to simplify the support for existing functionalities and to develop new ones, where the users (visitors and managers) were modeled as elementary entities (figure 5.2).

In the following several examples of groups are given for this application.

Visitor groups. These groups are used to distribute information and share photos (or other content) taken during a tour. Assuming that all the visitors have already registered into the system, by invoking the *register* primitive, a typical sequence of actions is:

- a visitor guide creates a new explicit group with name "tour14" using the primitive create (`g1=create(..., "tour14", EXPLICIT, ...)`);
- a visitor (`u1`) can join the previously created group (g1) using the primitive `join(u1,g1, ...)`);

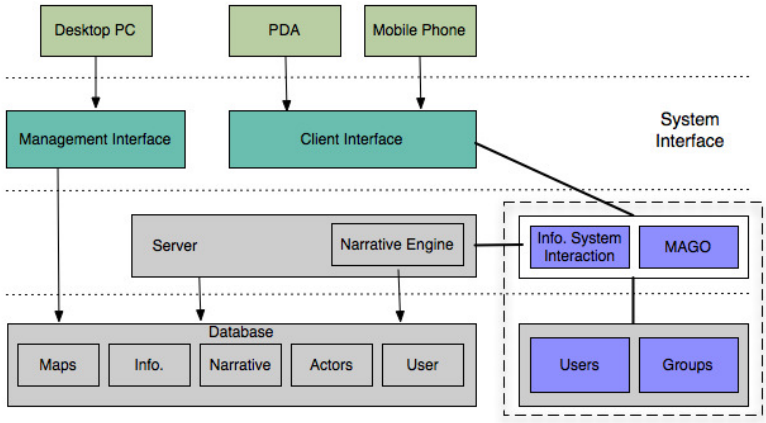

FIG. 5.2. *InStory architecture (an already existing system architecture) adapted to incorporate the MAGO model.*

- the visitor can access group functionalities:
  - sharing a photo and notify all group members: first update a file to the information system, then create a data object (`data_obj`) with information specified by the application (for example with the reference to the data file on the information system and a location reference), and finally update data on the "tour14" group shared space notifying all the group members (`update(u1,g1,, NOTIFY,data_obj)`);
  - other members can now access the photo, based on notification information: first create a data object with the notification information, then consult (without blocking) the data on "tour14" group shared space (`consult(u1,g1,,, NO_BLOCK, data_obj)`) and extract the information system reference to the photo, from the data object, finally get the file from the information system and visualize it on the local device;
  - disseminate information to all group members: spread a message to the group, by invoking the send primitive (`send(u1, g1, SPREAD, "tour start in 5m")`).

  In this example, the data files such as the ones containing photos and videos are stored in the information system maintained by the application.

Implicit groups to personalize the information. While the above functionalities could rely on the explicit specification and management of the mentioned groups, that could become a quite cumbersome task due to the highly dynamic nature of the applications. Our defined concept of implicit groups incorporated in the MAGO model is very useful to manage the dynamic changes in user profiles and system configuration, as illustrated in the following scenarios.

First a system manager (acting as the tour guide) creates an implicit group defining the value attributes that all the group members must match.

- define the attributes list (`lst=[nationality,"italian"]`);
- create the new group (`create(id_usr,"itVisit",IMPLICIT,-,-,lst)`);
- update the group information on the information system.

After having created the group, all the users that match the specified attributes are automatically joined to the group. Now the new group can be used, for example, to send personalized messages. If a new visitor has attributes, that match the implicit group attributes, he/she automatically becomes a member of the group.

Implicit groups and location dependent contents. As in the previous example a system manager creates an implicit group, this time with attributes `lst=[local, "Templar Well"]`. Having created this group, each time a visitor arrives at the specified location, the location attribute changes to "Templar Well", and the visitor becomes a member of the group, allowing access to the group resources. Having arrived at that location, the visitors can interact with other group members, using the group communication mechanisms offered by the model, and can also update and access the contents of the group shared space.

When a visitor leaves that location (by changing the location attribute) he/she will be automatically removed from the group "Templar Well", by the system.

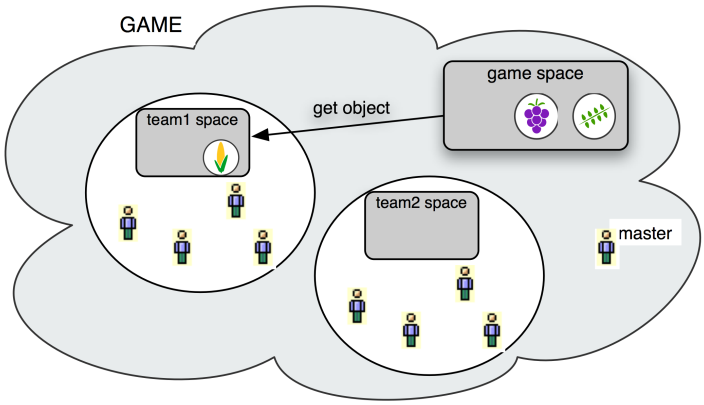

FIG. 5.3. *Mobile game scenario*

Use in a mobile game scenario. We defined a mobile game scenario where groups of users compete with each other and try to be the fastest on collecting virtual artifacts that were previously defined and distributed on the gardens and palace by the game manager (figure 5.3). The team that collects more artifacts wins the game.

For modeling this application we defined several groups: one for each team and one for the game space. The artifacts are characterized by a name, a position and other information related to the game functionality. At the beginning of the game they are on the shared space of the game group.

When arriving at a specified position a player asks the game group if an artifact is available, by invoking a get primitive (`get(player1, game, ...)`). If this operation succeeds (artifact was on the game group shared space), this means that the artifact gets removed from the game space. Then the player must put the object on the player's team space and notify the other team members that a new object is available (`update(player1,team1,..,NOTIFY,ObjArt)`). When executing this get operation, if no artifact is collected this means that another player has already collected that object.

In a similar way the players can collect clues instead of artifacts that help progressing in the game. During the game the team players can also communicate directly with each other or with the teams (*send()*), or leave clues for their team members (`update(player1,team1,PUBLIC,NO_NOTIFY,ObjClue)`).

The game manager can also communicate at any time with all the players and teams, using the same primitives. For instance, for sending a message to all the game players announcing the end of the game, a send primitive will be used directed to the group game (`send(master,game,SPREAD,"Game Over")`).

5.2. School campus. With this case study, like in the previous one we start from an already existing system and incorporate MAGO as a way to extend some of the application functionalities (figure 5.4). In this case study, the users (professors and students) were modeled as elementary entities.

The establishment and generation of groups is usually based on the users professional activities and/or personal characteristics, interests and hobbies, such as (5.5):

- group of teachers or computer science teacher (CS teachers),
- group of computer science student (Joe Group),
- players of the football team (FTeam),
- groups associated with classes or courses (Programming (P1) Class, Math Class),
- group of users of room 123.

Or groups associated with the users preferences and/or profiles, like for instance:

- fans of action movies,
- students looking for a room/apartment to rent/share,
- students looking for a part-time job,
- people that have cars and are going to town,
- people that need a ride to town.

These groups are used for spreading information (for example: about a change on a due date for an assignment), for scheduling meetings (ex: agreement on a meeting date) and share media data (ex: put a video

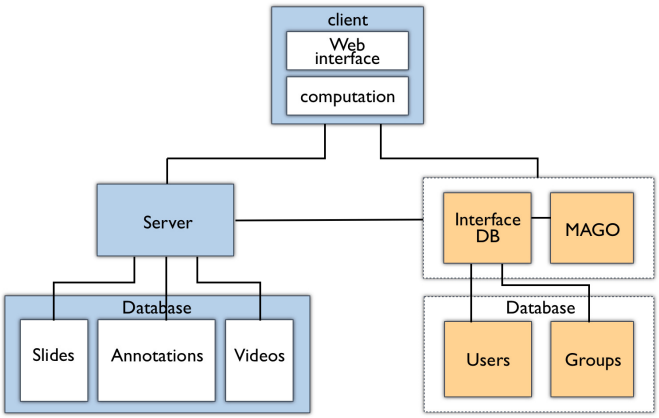

FIG. 5.4. *VideoStore architecture (an already existing system architecture) incorporated with the MAGO model.*

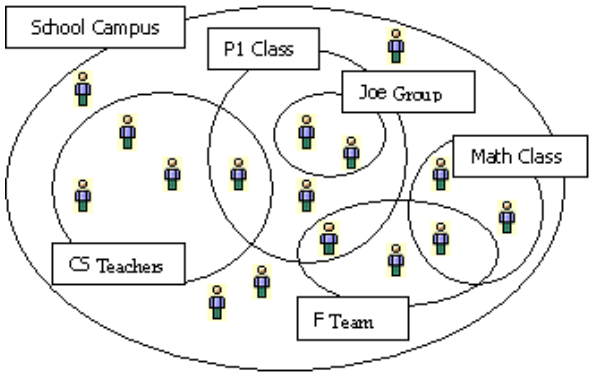

FIG. 5.5. *Possible groups in a school campus scenario*

and/or slides of a class that will be available to the group of students of that specific class). Next we illustrate some of the possibilities of the use of explicit and implicit groups in this campus scenario.

Explicit groups. For example to form and manage the explicit group for the user Joe (Joe Group), a typical sequence of actions that will be performed is as follows:

- Create the group "JoeG": this action is launched by user Joe and is made by invoking the create primitive: `joegID=create(joeID,"JoeG",EXPLICIT,by_creator,5)`. This will create a group limited to 5 members and where new members admission is granted only by the group creator Joe;
- Join the group "JoeG": this action is performed by group candidates by invoking a join primitive (`(join(maryID,joegID)`. After this action Mary will become a member of the group "JoeG", if authorization to join was granted;
- After a group creation, the members can use it to share and access group data, to disseminate information to all group members, or simply to communicate directly with each other:
  - share data, using the update primitive: `update(maryID,joegID, ,NOTIFY,data_object`. This instruction will announce to all group members that new data was updated. The data can be accessed by all group members with a consult primitive (`consult(joeID, joegID, , ,NO_BLOCK, data_object)`. The `data_object` can be of any type, for instance a text, a video or a photo.
  - send message to the group, using the send primitive with the `spread` option (`send(joeID, JoeG, SPREAD, "Hello all")`: this will disseminate the message to all group members.

Implicit groups. If for instance a teacher wants to create a group of all the school members that live in Lisbon, he/she should create an implicit group, where the matching rule that must be verified by all the members will be: `"home=Lisbon"`. In this case the create primitive will have to specify the attributes list `list=[home,"Lisbon"]`. So the create primitive will have the form: `homeID=create(t1,"LisbonG",IMPLICIT, , ,list)`

As already mentioned in the tourism case study, the implicit groups are also useful to create groups associated to location, like for instance the group of people that currently are near "Building1" (`list=[local, "Building1"]`). With this group created, each time a system user arrives at the specified location, his/hers location attributes change and the user will automatically became a member of that group. This group can be used in order to give information about an event that is taking place at that location. The group can also be used for the users to communicate and share data with each other.

6. Conclusions and ongoing work. Emerging interactive applications increasingly exhibit requirements for collaboration and information sharing in multiuser dynamic environments. In order to model those applications, we are exploiting group based abstractions, according to the MAGO approach. The model supports a framework for group management, allowing the structuring of an application in terms of a dynamic collection of entities. Furthermore, groups provide confined interactions spaces, offering different forms of communication (messages, events and shared space), which were found suitable to model the typical application interaction patterns. In order to ease the modeling process, MAGO supports a new concept of implicit groups, allowing an automated management of groups, based on dynamic identification of user profiles.

As part of ongoing work MAGO is being used to design and implement other applications with more complex services, namely in the tourism and cultural heritage domains, and in learning environments.

Acknowledgments. The authors thank the FCT/MCTES for the support given to the development of this work.

REFERENCES

[1] Bela Ban, *JavaGroups—Group Communication Patterns in Java*, Technical Report,Cornell University, 1998.
[2] Luca Chittaro, Demis Corvaglia, Luca De Marco, Silvia Gabrielli, and Augusto Senerchia, *Tech4Tourism*, `http://hcilab.uniud.it/t4t/index_en.html`, 2006.
[3] Gregory V. Chockler, Idid Keidar, and Roman Vitenberg, *Group communication specifications: a comprehensive study*, ACM Comput. Surv., 33 (2001), pp 427–469, ACM Press.
[4] Nuno Correia, Luis Alves, Helder Correia, Luis Romero, Carmen Morgado, Luís Soares, José C. Cunha, Teresa Romão, A. Eduardo Dias,and Joaquim A. Jorge, *InStory: a system for mobile information access, storytelling and gaming activities in physical spaces*, ACE '05: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology, pp 102–109, ACM Press.
[5] José C. Cunha, Carmen P. Morgado, and Jorge F. Custódio, *Group Abstractions for Organizing Dynamic Distributed Systems*, Euro-Par 2008 Workshops—Parallel Processing, pp 450–459, Springer-Verlag
[6] Jorge Custódio, *JGroupSpace- Support for group oriented distributed programming (in Portuguese)*, Master Thesis, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2008.
[7] Patrick Th. Eugster and Pascal A. Felber and Rachid Guerraoui and Anne-Marie Kermarrec, *The many faces of publish/subscribe*, ACM Comput. Surv., 35 (2003), pp.114–131.
[8] David Gelernter, *Generative communication in Linda*, ACM Trans. Program. Lang. Syst., 7 (1985), pp. 80–112, New York, NY, USA.
[9] Nicholas Carriero and David Gelernter, *Linda in context*, Commun. ACM, 32 (1989), pp. 444–458, New York, NY, USA.
[10] R. Kernchen, D. Bonnefoy, A. Battestini, B. Mrohs, M. Wagner, and M. Klemettinen, *Context-Awareness in MobiLife*, Proceedings of the 15th IST Mobile and Wireless Communication Summit, 2006.
[11] Carmen Morgado, *A group-based model for interactive distributed applications (in Portuguese)*, Ph.D. thesis, Department of Computer Science, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Lisboa, 2009.
[12] Carmen Morgado, Nuno Correia, and J. C. Cunha, *A group-based approach for modeling interactive mobile applications (poster)*, International ACM Conference on Supporting Group Work—Group07, Sanibel, USA, 2007.
[13] Carmen Santoro, Fabio Paterno, Giulia Ricci, and Barbara Leporini, *A Multimodal Mobile Museum Guide for All*, Mobile Interaction with the Real World (MIRW 2007)—Workshop @ MobileHCI, 2007.
[14] MediaLab, *CONFECTIONARY*, `http://mf.media.mit.edu/confectionary`, Accessed July 2009.
[15] canal*accessible, *zexe.net*, `http://www.megafone.net/BARCELONA`, Accessed August 2009.
[16] Kenneth P. Birman, *The process group approach to reliable distributed computing*, Communication ACM, 36 (1993), pp 37–53.
[17] David C. Yen, H. Joseph Wen, Binshan Lin, and David C. Chou, *Groupware: a strategic analysis and implementation*, Industrial Management and Data Systems, 99 (1999), pp 64–70.
[18] P. Wyckoff and S. W. McLaughry and T. J. Lehman and D. A. Ford, *TSpaces*, IBM Systems Journal, 37, April, 1998.

Edited by: Costin Bădică, Viorel Negru
Received: March 6, 2010
Accepted: March 31, 2010