

Context Awareness for Group Interaction Support

Alois Ferscha, Clemens Holzmann, Stefan Oppl
Institut für Pervasive Computing, Johannes Kepler Universität Linz
Altenbergerstraße 69, A-4040 Linz
{ferscha,holzmann,oppl}@soft.uni-linz.ac.at

ABSTRACT

In this paper, we present an implemented system for supporting group interaction in mobile distributed computing environments. First, an introduction to context computing and a motivation for using contextual information to facilitate group interaction is given. We then present the architecture of our system, which consists of two parts: a subsystem for location sensing that acquires information about the location of users as well as spatial proximities between them, and one for the actual context-aware application, which provides services for group interaction.

Categories and Subject Descriptors

C.2.4 [Computer-Communication Networks]: Distributed Systems – *distributed applications*.

H.1.2 [Models and Principles]: User/Machine Systems – *human factors*.

H.5.3 [Information Interfaces and Presentation]: Group and Organization Interfaces – *asynchronous interaction, collaborative computing, theory and models, synchronous interaction*.

General Terms

Design, Experimentation

Keywords

Context awareness, group interaction, location sensing, sensor fusion

1. INTRODUCTION

Today's computing environments are characterized by an increasing number of powerful, wirelessly connected mobile devices. Users can move throughout an environment while carrying their computers with them and having remote access to information and services, anytime and anywhere. New situations appear, where the user's context – for example his current location or nearby people – is more dynamic; computation does not occur at a single location and in a single context any longer, but comprises a multitude of situations and locations. This development leads to a new class of applications, which are *aware of the context* in which they run in and thus bringing virtual and real worlds together.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

MobiWac'04, October 1, 2004, Philadelphia, Pennsylvania, USA.
Copyright 2004 ACM 1-58113-920-9/04/0010...\$5.00.

Motivated by this and the fact, that only a few studies have been done for supporting group communication in such computing environments [12], we have developed a system, which we refer to as *Group Interaction Support System (GISS)*. It supports group interaction in mobile distributed computing environments in a way that group members need not to be at the same place any longer in order to interact with each other or just to be aware of the others situation.

In the following subchapters, we will give a short overview on context aware computing and motivate its benefits for supporting group interaction. A software framework for developing context-sensitive applications is presented, which serves as middleware for GISS. Chapter 2 presents the architecture of GISS, and chapter 3 and 4 discuss the location sensing and group interaction concepts of GISS in more detail. Chapter 5 gives a final summary of our work.

1.1 What is Context Computing?

According to Merriam-Webster's Online Dictionary¹, *context* is defined as the “*interrelated conditions in which something exists or occurs*”. Because this definition is very general, many approaches have been made to define the notion of context with respect to computing environments.

Most definitions of context are done by enumerating examples or by choosing synonyms for context. The term context-aware has been introduced first in [10] where context is referred to as location, identities of nearby people and objects, and changes to those objects. In [2], context is also defined by an enumeration of examples, namely location, identities of the people around the user, the time of the day, season, temperature etc. [9] defines context as the user's location, environment, identity and time.

Here we conform to a widely accepted and more formal definition, which defines context as “*any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves*”. [4]

[4] identifies four primary types of context information (sometimes referred to as *context dimensions*), that are – with respect to characterizing the situation of an entity – more important than others. These are *location, identity, time* and *activity*, which can also be used to derive other sources of contextual information (secondary context types). For example, if we know a person's identity, we can easily derive related information about this person from several data sources (e.g. day of birth or e-mail address).

¹ <http://www.m-w.com>

According to this definition, [4] defines a system to be context-aware “if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task”. [4] also gives a classification of features for context-aware applications, which comprises *presentation* of information and services to a user, *automatic execution* of a service and *tagging* of context to information for later retrieval.

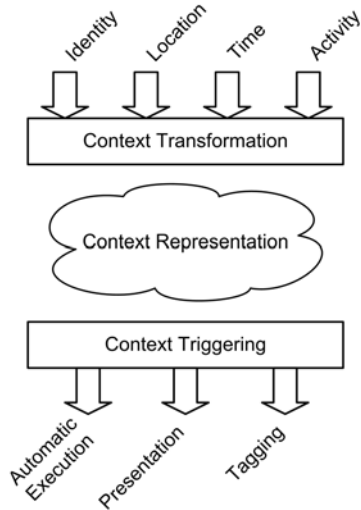


Figure 1. Layers of a context-aware system

Context computing is based on two major issues, namely identifying relevant context (*identity, location, time, activity*) and using obtained context (*automatic execution, presentation, tagging*). In order to do this, there are a few layers between (see Figure 1). First, the obtained low-level context information has to be transformed, aggregated and interpreted (*context transformation*) and represented in an abstract context world model (*context representation*), either centralized or decentralized. Finally, the stored context information is used to trigger certain context events (*context triggering*). [7]

1.2 Group Interaction in Context

After these abstract and formal definitions about what context and context computing is, we will now focus on the main goal of this work, namely how the interaction of *mobile group members* can be supported by using context information.

In [6] we have identified organizational systems to be crucial for supporting mobile groups (see Figure 2). First, there has to be an *Information and Knowledge Management System*, which is capable of supporting a team with its information processing- and knowledge gathering needs. The next part is the *Awareness System*, which is dedicated to the perceptualisation of the effects of team activity. It does this by communicating work context, agenda and workspace information to the users. The *Interaction Systems* provide support for the communication among team members, either synchronous or asynchronous, and for the shared access to artefacts, such as documents. *Mobility Systems* deploy mechanisms to enable any-place access to team memory as well as to the capturing and delivery of awareness information from and to any places. Finally yet importantly, the organisational innovation system integrates aspects of the team itself, like roles, leadership and shared facilities.

With respect to these five aspects of team support, we focus on *interaction* and partly cover *mobility-* and *awareness-*support.

Group interaction includes all means that enable group members to communicate freely with all the other members. At this point, the question how *context information* can be used for supporting group interaction comes up. We believe that information about the current situation of a person provides a surplus value to existing group interaction systems. Context information facilitates group interaction by allowing each member to be aware of the availability status or the current location of each other group member, which again makes it possible to form groups dynamically, to place virtual post-its in the real world or to determine which people are around.

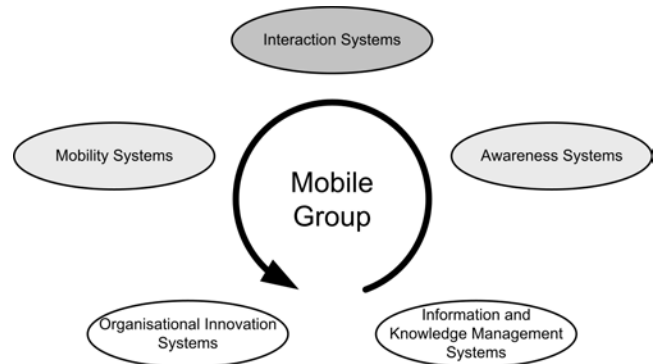


Figure 2. Support for Mobile Groups [6]

Most of today’s context-aware applications use location and time only, and location is referred to as a crucial type of context information [3]. We also see the importance of location information in mobile and ubiquitous environments, wherefore a main focus of our work is on the utilization of location information and information about users in spatial proximity.

Nevertheless, we believe that location, as the only used type of context information, is not sufficient to support group interaction, wherefore we also take advantage of the other three primary types, namely identity, time and activity. This provides a comprehensive description of a user’s current situation and thus enabling numerous means for supporting group interaction, which are described in detail in chapter 4.4.

When we look at the types of context information stated above, we can see that all of them are *single user-centred*, taking into account only the context of the user itself. We believe, that for the support of group interaction, the status of the group itself has also be taken into account. Therefore, we have added a fifth context-dimension *group-context*, which comprises more than the sum of the individual member’s contexts. Group context includes any information about the situation of a whole group, for example how many members a group currently has or if a certain group meets right now.

1.3 Context Middleware

The *Group Interaction Support System (GISS)* uses the software-framework introduced in [1], which serves as a middleware for developing context-sensitive applications. This so-called *Context Framework* is based on a distributed communication architecture and it supports different kinds of transport protocols and message coding mechanisms.

A main feature of the framework is the *abstraction* of context information retrieval via various sensors and its *delivery* to a level where no difference appears, for the application designer, between these different kinds of context retrieval mechanisms; the information retrieval is hidden from the application developer. This is achieved by so-called *entities*, which describe objects – e.g. a human user – that are important for a certain context scenario.

Entities express their functionality by the use of so-called *attributes*, which can be loaded into the entity. These attributes are complex pieces of software, which are implemented as Java classes. Typical attributes are encapsulations of sensors, but they can also be used to implement context services, for example to notify other entities about location changes of users.

Each entity can contain a collection of such attributes, where an entity itself is an attribute. The initial set of attributes an entity contains can *change dynamically at runtime*, if an entity loads or unloads attributes from the local storage or over the network. In order to load and deploy new attributes, an entity has to reference a *class loader* and a *transport and lookup layer*, which manages the lookup mechanism for discovering other entities and the transport. *XML configuration files* specify which initial set of entities should be loaded and which attributes these entities own.

The communication between entities and attributes is based on context *events*. Each attribute is able to trigger events, which are addressed to other attributes and entities respectively, independently on which physical computer they are running. Among other things, an event contains the name of the event and a list of parameters delivering information about the event itself.

Related with this event-based architecture is the use of *ECA (Event-Condition-Action)-rules* for defining the behaviour of the context system. Therefore, every entity has a *rule-interpreter*, which catches triggered *events*, checks *conditions* associated with them and causes certain *actions*. These rules are referenced by the entity’s XML configuration. A rule itself is even able to trigger the insertion of new rules or the unloading of existing rules at runtime in order to *change the behaviour of the context system dynamically*.

To sum up, the context framework provides a flexible, distributed architecture for hiding low-level sensor data from high-level applications and it hides external communication details from the application developer. Furthermore, it is able to adapt its behaviour dynamically by loading attributes, entities or ECA-rules at runtime.

2. ARCHITECTURE OVERVIEW

As GISS uses the Context Framework described in chapter 1.3 as middleware, every user is represented by an entity, as well as the central server, which is responsible for context transformation, context representation and context triggering (cf. Figure 1).

A main part of our work is about the *automated acquisition* of position information and its *sensor-independent provision* at application level. We do not only sense the current *location* of users, but also determine *spatial proximities* between them. Developing the architecture, we focused on keeping the client as simple as possible and reducing the communication between client and server to a minimum.

Each client may have various location and/or proximity sensors attached, which are encapsulated by respective Context Framework-attributes (“*Sensor Encapsulation*”). These attributes are responsible for integrating native sensor-implementations into the Context Framework and sending sensor-dependent position information to the server. We consider it very important to support different types of sensors even at the same time, in order to improve location accuracy on the one hand, while providing a pervasive location-sensing environment with seamless transition between different location sensing techniques on the other hand.

All location- and proximity-sensors supported are represented by server-side context-attributes, which correspond to the client-side sensor encapsulation-attributes and abstract the sensor-dependent position information received from all users via the wireless network (*sensor abstraction*). This requires a context repository, where the mapping of diverse physical positions to standardized locations is stored.

The *standardized location- and proximity-information* of each user is then passed to the so-called “*Sensor Fusion*”-attributes, one for symbolic locations and a second one for spatial proximities. Their job is to merge location- and proximity-information of clients, respectively, which is described in detail in Chapter 3.3. Every time the symbolic location of a user or the spatial proximity between two users changes, the “*Sensor Fusion*”-attributes notify the “*GISS Core*”-attribute, which controls the application.

Because of the abstraction of sensor-dependent position information, the system can easily be *extended by additional sensors*, just by implementing the (typically two) attributes for encapsulating sensors (some sensors may not need a client-side part), abstracting physical positions and observing the interface to “*GISS Core*”.

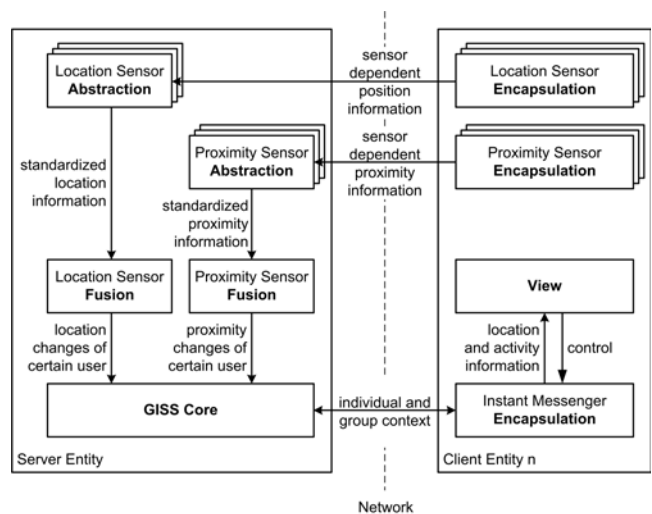


Figure 3. Architecture of the Group Interaction Support System (GISS)

The “*GISS Core*”-attribute is the central coordinator of the application as it shows to the user. It not only serves as an interface to the location-sensing subsystem, but also collects further context information in other dimensions (time, identity or activity).

Every time a change in the context of one or more users is detected, “GISS Core” evaluates the effect of these changes on the user, on the groups he belongs to and on the other members of these groups. Whenever necessary, events are thrown to the affected clients to trigger context-aware activities, like changing the presentation of awareness information or the execution of services.

The client-side part of the application is kept as simple as possible. Furthermore, *modular design* was not only an issue on the sensor side but also when designing the user interface architecture. Thus, the complete user interface can be easily exchanged, if all of the defined events are taken into account and understood by the new interface-attribute.

The currently implemented user interface is split up in two parts, which are also represented by two attributes. The central attribute on client-side is the so-called “*Instant Messenger Encapsulation*”, which on the one hand interacts with the server through events and on the other hand serves as a proxy for the external application the user interface is built on.

As external application, we use an existing open source instant messenger – the ICQ²-compliant *Simple Instant Messenger (SIM)*³. We have chosen an instant messenger as front-end because it provides a well-known interface for most users and facilitates a seamless integration of group interaction support, thus increasing acceptance and ease of use. As the basic functionality of the instant messenger – to serve as a client in an instant messenger network – remains fully functional, our application is able to use the features already provided by the messenger. For example, the contexts *activity* and *identity* are derived from the messenger network as it is described later.

The *Instant Messenger Encapsulation* is also responsible for supporting group communication. Through the interface of the messenger, it provides means of synchronous and asynchronous communication as well as a context-aware reminder system and tools for managing groups and the own availability status.

The second part of the user interface is a visualisation of the user’s locations, which is implemented in the attribute “*Viewer*”. The current implementation provides a two-dimensional map of the campus, but it can easily be replaced by other visualisations, a three-dimensional VRML-model for example. Furthermore, this visualisation is used to show the artefacts for asynchronous communication. Based on a floor plan-view of the geographical area the user currently resides in, it gives a quick overview of which people are nearby, their state and provides means to interact with them.

In the following chapters 3 and 4, we describe the location sensing-backend and the application front-end for supporting group interaction in more detail.

3. LOCATION SENSING

In the following chapter, we will introduce a *location model*, which is used for representing locations; afterwards, we will describe the *integration of location- and proximity-sensors* in

² <http://www.icq.com/>

³ <http://sim-icq.sourceforge.net>

more detail. Finally, we will have a closer look on the *fusion of location- and proximity-information*, acquired by various sensors.

3.1 Location Model

A *location model* (i.e. a *context representation* for the context-information location) is needed to represent the locations of users, in order to be able to facilitate location-related queries like “given a location, return a list of all the objects there” or “given an object, return its current location”. In general, there are two approaches [3,5]: *symbolic models*, which represent location as abstract symbols, and a *geometric model*, which represent location as coordinates.

We have chosen a *symbolic location model*, which refers to locations as abstract symbols like “Room P111” or “Physics Building”, because we do not require geometric location data. Instead, abstract symbols are more convenient for human interaction at application level. Furthermore, we use a *symbolic location containment hierarchy* similar to the one introduced in [11], which consists of top-level *regions*, which contain *buildings*, which contain *floors*, and the floors again contain *rooms*. We also distinguish four types, namely *region* (e.g. a whole campus), *section* (e.g. a building or an outdoor section), *level* (e.g. a certain floor in a building) and *area* (e.g. a certain room). We introduce a fifth type of location, which we refer to as *semantic*. These so-called semantic locations can appear at any level in the hierarchy and they can be nested, but they do not necessarily have a geographic representation. Examples for such semantic locations are *tagged objects* within a room (e.g. a desk and a printer on this desk) or the name of a department, which contains certain rooms.

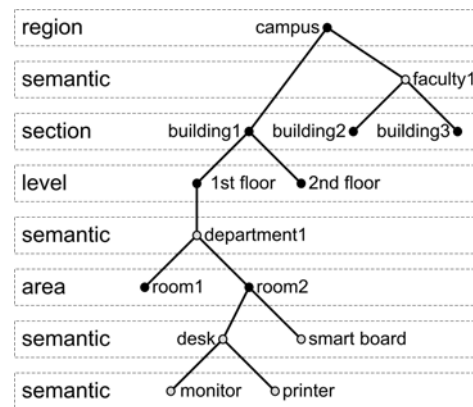


Figure 4. Symbolic Location Containment Hierarchy

The hierarchy of symbolic locations as well as the type of each position is stored in the context repository.

3.2 Sensors

Our architecture supports two different kinds of sensors: *location sensors*, which acquire location information, and *proximity sensors*, which detect spatial proximities between users.

As described above, each sensor has a server- and in most cases a corresponding client-side-implementation, too. While the client-attributes (“*Sensor Abstraction*”) are responsible for acquiring low-level sensor-data and transmitting it to the server, the corresponding “*Sensor Encapsulation*”-attributes transform them into a *uniform and sensor-independent format*, namely symbolic locations and IDs of users in spatial proximity, respectively.

Afterwards, the respective attribute “Sensor Fusion” is being triggered with this sensor-independent information of a certain user, detected by a particular sensor. Such notifications are performed every time the sensor acquired new information. Accordingly, “Sensor Abstraction”-attributes are responsible to detect when a certain sensor is no longer available on the client side (e.g. if it has been unplugged by the user) or when position respectively proximity could not be determined any longer (e.g. RFID reader cannot detect tags) and notify the corresponding sensor fusion about this.

3.2.1 Location Sensors

In order to sense physical positions, the “Sensor Encapsulation”-attributes asynchronously transmit sensor-dependent position information to the server. The corresponding location “Sensor Abstraction”-attributes collect these physical positions delivered by the sensors of all users, and perform a repository-lookup in order to get the associated symbolic location. This requires certain tables for each sensor, which map physical positions to symbolic locations. One physical position may have multiple symbolic locations at different accuracy-levels in the location hierarchy assigned to, for example if a sensor covers several rooms. If such a mapping could be found, an event is thrown in order to notify the attribute “Location Sensor Fusion” about the symbolic locations a certain sensor of a particular user determined.

We have prototypically implemented three kinds of location sensors, which are based on WLAN (IEEE 802.11), Bluetooth and RFID (Radio Frequency Identification). We have chosen these three completely different sensors because of their differences concerning accuracy, coverage and administrative effort, in order to evaluate the flexibility of our system (see Table 1).

The most accurate one is an *RFID sensor*, which is based on an active RFID-reader. As soon as the reader is plugged into the client, it scans for active RFID tags in range and transmits their serial numbers to the server, where they are mapped to symbolic locations. We also take into account RSSI (Radio Signal Strength Information), which provides position accuracy of few centimetres and thus enables us to determine which RFID-tag is nearest. Due to this high accuracy, RFID is used for locating users within rooms. The administration is quite simple; once a new RFID tag is placed, its serial number simply has to be assigned to a single symbolic location. A drawback is the poor availability, which can be traced back to the fact that RFID readers are still very expensive.

The second one is an 802.11 *WLAN sensor*. Therefore, we integrated a purely software-based, commercial WLAN positioning system for tracking clients on the university campus-wide WLAN infrastructure. The reached position accuracy is in the range of few meters and thus is suitable for location sensing at the granularity of rooms. A big disadvantage is that a map of the whole area has to be calibrated with measuring points at a distance of 5 meters each. Because most mobile computers are equipped with WLAN technology and the positioning-system is a software-only solution, nearly everyone is able to use this kind of sensor.

Finally, we have implemented a *Bluetooth sensor*, which detects Bluetooth tags (i.e. Bluetooth-modules with known position) in range and transmits them to the server that maps to symbolic locations. Because of the fact that we do not use signal strength-

information in the current implementation, the accuracy is above 10 meters and therefore a single Bluetooth MAC address is associated with several symbolic locations, according to the physical locations such a Bluetooth module covers. This leads to the disadvantage that the range of each Bluetooth-tag has to be determined and mapped to symbolic locations within this range.

Table 1. Comparison of implemented sensors

Sensor	Accuracy	Coverage	Administration
RFID	< 10 cm	poor	easy
WLAN	1-4 m	very well	very time-consuming
Bluetooth	~ 10 m	well	time-consuming

3.2.2 Proximity Sensors

Any sensor that is able to detect whether two users are in spatial proximity is referred to as *proximity sensor*. Similar to the location sensors, the “Proximity Sensor Abstraction”-attributes collect physical proximity information of all users and transform them to mappings of user-IDs.

We have implemented two types of proximity-sensors, which are based on *Bluetooth* on the one hand and on *fused symbolic locations* (see chapter 3.3.1) on the other hand.

The *Bluetooth*-implementation goes along with the implementation of the Bluetooth-based location sensor. The already determined Bluetooth MAC addresses in range of a certain client are being compared with those of all other clients, and each time the attribute “Bluetooth Sensor Abstraction” detects congruence, it notifies the proximity sensor fusion about this.

The second sensor is based on *symbolic locations* processed by “Location Sensor Fusion”, wherefore it does not need a client-side implementation. Each time the fused symbolic location of a certain user changes, it checks whether he is at the same symbolic location like another user and again notifies the proximity sensor fusion about the proximity between these two users. The range can be restricted to any level of the location containment hierarchy, for example to room granularity.

A currently unresolved issue is the incomparable granularity of different proximity sensors. For example, the symbolic locations at same level in the location hierarchy mostly do not cover the same geographic area.

3.3 Sensor Fusion

Core of the location sensing subsystem is the *sensor fusion*. It merges data of various sensors, while coping with differences concerning accuracy, coverage and sample-rate. According to the two kinds of sensors described in chapter 3.2, we distinguish between *fusion of location sensors* on the one hand, and *fusion of proximity sensors* on the other hand.

The fusion of symbolic locations as well as the fusion of spatial proximities operates on standardized information (cf. Figure 3). This has the advantage, that additional position- and proximity-sensors can be added easily or the fusion algorithms can be replaced by ones that are more sophisticated.

Fusion is performed for each user separately and takes into account the measurements at a single point in time only (i.e. no history information is used for determining the current location of a certain user). The algorithm collects all events thrown by the “Sensor Abstraction”-attributes, performs fusion and triggers the “GISS Core”-attribute if the symbolic location of a certain user or the spatial proximity between users changed.

An important feature is the persistent storage of *location- and proximity-history* in a database in order to allow future retrieval. This enables applications to visualize the movement of users for example.

3.3.1 Location Sensor Fusion

Goal of the fusion of location information is to improve precision and accuracy by merging the set of symbolic locations supplied by various location sensors, in order to reduce the number of these locations to a minimum, ideally to a single symbolic location per user. This is quite difficult, because different sensors may differ in accuracy and sample rate as well.

The “Location Sensor Fusion”-attribute is triggered by events, which are thrown by the “Location Sensor Abstraction”-attributes. These events contain information about the identity of the user concerned, his current location and the sensor by which the location has been determined.

If the attribute “Location Sensor Fusion” receives such an event, it checks if the amount of symbolic locations of the user concerned has changed (compared with the last event). If this is the case, it notifies the “GISS Core”-attribute about *all* symbolic locations this user is currently associated with.

However, this information is not very useful on its own if a certain user is associated with several locations. As described in chapter 3.2.1, a single location sensor may deliver multiple symbolic locations. Moreover, a certain user may have several location sensors, which supply symbolic locations differing in accuracy (i.e. different levels in the location containment hierarchy). To cope with this challenge, we implemented a fusion algorithm in order to reduce the number of symbolic locations to a minimum (ideally to a single location).

In a first step, each symbolic location is associated with its number of occurrences. A symbolic location may occur several times if it is referred to by more than one sensor or if a single sensor detects multiple tags, which again refer to several locations. Furthermore, this number is added to the previously calculated number of occurrences of each symbolic location, which is a child-location of the considered one in the location containment hierarchy. For example, if – in Figure 4 – “room2” occurs two times and “desk” occurs a single time, the value 2 of “room2” is added to the value 1 of “desk”, whereby “desk” finally gets the value 3. In a final step, only those symbolic locations are left which are assigned with the *highest number of occurrences*.

A further reduction can be achieved by assigning *priorities* to sensors (based on accuracy and confidence) and cumulating these priorities for each symbolic location instead of just counting the number of occurrences.

If the remaining *fused locations* have changed (i.e. if they differ from the fused locations the considered user is currently associated with), they are provided with the current timestamp,

written to the database and the “GISS”-attribute is notified about where the user is *probably* located.

Finally, the most accurate, *common* location in the location hierarchy is calculated (i.e. the least upper bound of these symbolic locations) in order to get a single symbolic location. If it changes, the “GISS Core”-attribute is triggered again.

3.3.2 Proximity Sensor Fusion

Proximity sensor fusion is much simpler than the fusion of symbolic locations. The corresponding proximity sensor fusion-attribute is triggered by events, which are thrown by the “Proximity Sensor Abstraction”-attributes. These special events contain information about the identity of the two users concerned, if they are currently in spatial proximity or if proximity no longer persists, and by which proximity-sensor this has been detected.

If the sensor fusion-attribute is notified by a certain “Proximity Sensor Abstraction”-attribute about an existing spatial proximity, it first checks if these two users are already known to be in proximity (detected either by another user or by another proximity-sensor of the user, which caused the event). If not, this change in proximity is written to the context repository with current timestamp. Similarly, if the attribute “Proximity Fusion” is notified about an ended proximity, it checks if the users are still known to be in proximity, and writes this change to the repository if not.

Finally, if spatial proximity between the two users actually changed, an event is thrown to notify the “GISS Core”-attribute about this.

4. CONTEXTSENSITIVE INTERACTION

4.1 Overview

In most of today’s systems supporting interaction in groups, the provided means lack any awareness of the user’s current context, thus being unable to adapt to his needs.

In our approach, we use context information to enhance interaction and provide further services, which offer new possibilities to the user. Furthermore, we believe that interaction in groups also has to take into account the current *context of the group* itself and not only the context of individual group members. For this reason, we also retrieve information about the group’s current context, derived from the contexts of the group members together with some sort of meta-information (see chapter 4.3).

The sources of context used for our application correspond with the four primary context types given in chapter 1.1 – *identity (I)*, *location (L)*, *time (T)* and *activity (A)*. As stated before, we also take into account the context of the group the user is interaction with, so that we could add a fifth type of context information – *group awareness (G)* – to the classification. Using this context information, we can trigger context-aware activities in all of the three categories described in chapter 1.1 – *presentation of information (P)*, *automatic execution of services (A)* and *tagging of context to information for later retrieval (T)*.

Table 2 gives an overview of activities we have already implemented; they are described comprehensively in chapter 4.4. The table also shows which types of context information are used for each activity and the category the activity could be classified in.

Table 2. Classification of implemented context-aware activities

Service	L	T	I	A	G	P	A	T
Location Visualisation	X		X			X		
Group Building Support	X		X	X			X	
Support for Synchronous Communication			X	X	X		X	
Support for Asynchronous Communication	X	X	X	X	X	X		X
Availability Management	X	X					X	
Task Management Support	X	X	X			X		
Meeting Support	X	X	X		X	X	X	

Reasons for implementing these very features are to take advantage of all four types of context information in order to support group interaction by utilizing a comprehensive knowledge about the situation a single user or a whole group is in.

A critical issue for the user acceptance of such a system is the *usability of its interface*. We have evaluated several ways of presenting context-aware means of interaction to the user, until we came to the solution we use right now. Although we think that the user interface that has been implemented now offers the best trade-off between seamless integration of features and ease of use, it would be no problem to extend the architecture with other user interfaces, even on different platforms.

The chosen solution is based on an existing instant messenger, which offers several possibilities to integrate our system (see chapter 4.2). The biggest advantage of this approach is that the user is confronted with a graphical user interface he is already used to in most cases. Furthermore, our system uses an instant messenger account as an identifier, so that the user does not have to register a further account anywhere else (for example, the user can use his already existing ICQ²-account).

4.2 Instant Messenger Integration

Our system is based upon an existing instant messenger, the so-called *Simple Instant Messenger (SIM)*³. The implementation of this messenger is carried out as a project at Sourceforge⁴.

SIM supports multiple messenger protocols such as AIM⁵, ICQ² and MSN⁶. It also supports connections to multiple accounts at the same time. Furthermore, full support for SMS-notification (where provided from the used protocol) is given.

SIM is based on a *plug-in concept*. All protocols as well as parts of the user-interface are implemented as plug-ins. Its architecture is also used to extend the application's abilities to communicate with external applications. For this purpose, a *remote control plug-in* is provided, by which SIM can be controlled from external applications via socket connection. This remote control interface is extensively used by GISS for retrieving the contact list, setting the user's availability-state or sending messages. The

functionality of the plug-in was extended in several ways, for example to accept messages for an account (as if they would have been sent via the messenger network).

The messenger, more exactly the contact list (i.e. a list of profiles of all people registered with the instant messenger, which is visualized by listing their names as it can be seen in Figure 5), is also used to display locations of other members of the groups a user belongs to. This provides location awareness without taking too much space or requesting the user's full attention. A more comprehensive description of these features is given in chapter 4.4.

4.3 Sources of Context Information

While the location-context of a user is obtained from our location sensing subsystem described in chapter 3, we consider *further types of context than location* relevant for the support of group interaction, too.

Local time as a very important context dimension can be easily retrieved from the real time clock of the user's system. Besides location and time, we also use context information of user's activity and identity, where we exploit the functionality provided by the underlying instant messenger system. *Identity* (or more exactly, the mapping of IDs to names as well as additional information from the user's profile) can be distilled out of the contents of the user's contact list.

Information about the *activity* of a certain user is only available in a very restricted area, namely the activity at the computer itself. Other activities like making a phone call or something similar, cannot be recognized with the current implementation of the activity sensor. The only context-information used is the instant messenger's availability state, thus only providing a very coarse classification of the user's activity (online, offline, away, busy etc.). Although this may not seem to be very much information, it is surely relevant and can be used to improve or even enable several services.

Having collected the context information from all available users, it is now possible to distil some information about the *context of a certain group*. Information about the context of a group includes how many members the group currently has, if the group meets right now, which members are participating at a meeting, how many members have read which of the available posts from other team members and so on.

Therefore, some additional information like a list of members for each group is needed. These lists can be assembled manually (by users joining and leaving groups) or retrieved automatically. The context of a group is secondary context and is aggregated from the available contexts of the group members. Every time the context of a single group member changes, the context of the whole group is changing and has to be recalculated.

With knowledge about a user's context and the context of the groups he belongs to, we can provide several *context-aware services* to the user, which enhance his interaction abilities. A brief description of these services is given in chapter 4.4.

⁴ <http://sourceforge.net/>

⁵ <http://www.aim.com/>

⁶ <http://messenger.msn.com/>

4.4 Group Interaction Support

4.4.1 Visualisation of Location Information

An important feature is the visualisation of location information, thus allowing users to be *aware of the location* of other users and members of groups he joined, respectively.

As already described in chapter 2, we use *two different forms of visualisation*. The maybe more important one is to display location information in the *contact list* of the instant messenger, right beside the name, thus being always visible while not drawing the user's attention on it (compared with a two-dimensional view for example, which requires a own window for displaying a map of the environment).

Due to the restricted space in the contact list, it has been necessary to implement some sort of *level-of-detail concept*. As we use a hierarchical location model, we are able to determine the *most accurate common location* of two users. In the contact list, the current symbolic location one level below the previously calculated common location is then displayed. If, for example, user A currently resides in room "P121" at the first floor of a building and user B, which has to be displayed in the contact list of user A, is in room "P304" at the third floor, the most accurate common location of these two users is the building they are in. For that reason, the floor (i.e. one level beyond the common location, namely the building) of user B is displayed in the contact list of user A. If both people reside on the same floor or even in the same room, the room would be taken.

Figure 5 shows a screenshot of the Simple Instant Messenger³, where the current location of those people, whose location is known by GISS, is displayed in brackets right beside their name. On top of the image, the heightened, integrated GISS-toolbar is shown, which currently contains the following, implemented functionality (from left to right): Asynchronous communication for groups (see chapter 4.4.4), context-aware reminders (see chapter 4.4.6), two-dimensional visualisation of location-information, forming and managing groups (see chapter 4.4.2), context-aware availability-management (see chapter 4.4.5) and finally a button for terminating GISS.

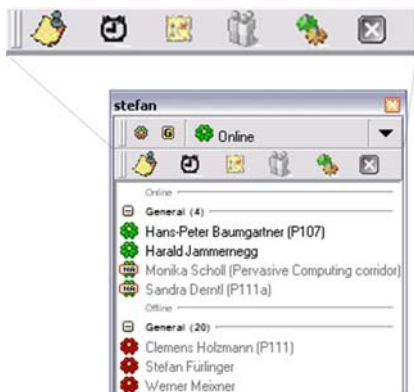


Figure 5. GISS integration in Simple Instant Messenger³

As displaying just this short form of location may not be enough for the user, because he may want to see the most accurate position available, a "fully qualified" position is shown if a name in the contact-list is clicked (e.g. in the form of "desk@room2@department1@1stfloor@building 1@campus").

The second possible form of visualisation is a *graphical* one. We have evaluated a three-dimensional view, which was based on a *VRML model* of the respective area (cf. Figure 6). Due to lacks in navigational and usability issues, we decided to use a *two-dimensional view* of the floor (it is referred to as *level* in the location hierarchy, cf. Figure 4). Other levels of granularity like section (e.g. building) and region (e.g. campus) are also provided.

In this floor-plan-based view, the current locations are shown in the manner of ICQ² contacts, which are placed at the currently sensed location of the respective person. The availability-status of a user, for example "away" if he is not on the computer right now, or "busy" if he does not want to be disturbed, is visualized by colour-coding the ICQ²-flower left beside the name. Furthermore, the floor-plan-view shows so-called the virtual post-its, which are virtual counterparts of real-life post-its and serve as our means of asynchronous communication (more about virtual post-its can be found in chapter 4.4.4).

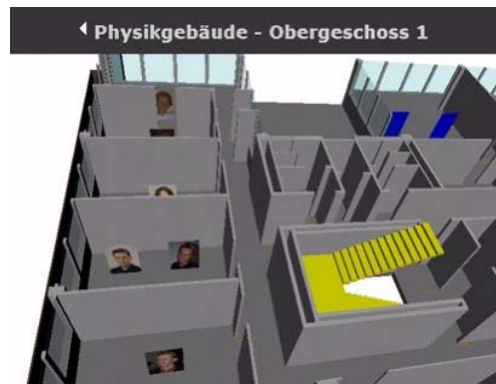


Figure 6. 3D-view of the floor (VRML)

Figure 7 shows the two-dimensional map of a certain floor, where several users are currently located (visualized by their name and the flower left beside). The location of the client, on which the map is displayed, is visualized by a green circle. Down to the right, two virtual post-its can be seen.

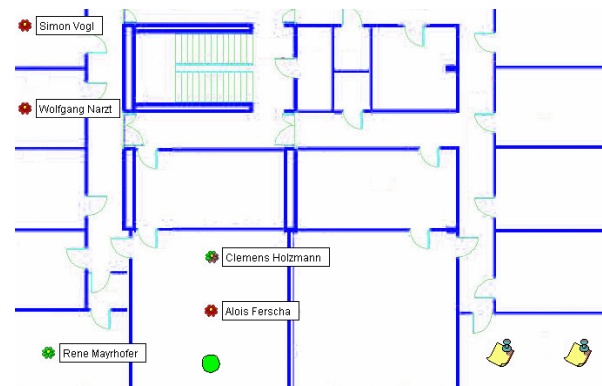


Figure 7. 2D view of the floor

Another feature of the 2D-view is the *visualisation of location-history* of users. As we store the complete history of a user's locations together with a timestamp, we are able to provide information about the locations he has been back in time. When the mouse is moved over the name of a certain user in the 2D-view, "footprints" of a user, placed at the locations he has been, are faded out the stronger, the older the location information is.

4.4.2 Forming and Managing Groups

To support interaction in groups, it is first necessary to form groups. As groups can have different purposes, we distinguish *two types of groups*.

So-called *static groups* are groups, which are built up manually by people joining and leaving them. Static groups can be further divided into two subtypes. In *open static groups*, everybody can join and leave anytime, useful for example to form a group of lecture attendees of some sort of interest group. *Closed static groups* have an owner, who decides, which persons are allowed to join, although everybody could leave again at any time. Closed groups enable users for example to create a group of their friends, thus being able to communicate with them easily.

In contrast to that, we also support the creation of *dynamic groups*. They are formed among persons, who are at the same location at the same time. The creation of dynamic groups is only performed at locations, where it makes sense to form groups, for example in lecture halls or meeting rooms, but not on corridors or outdoor. It would also be not very meaningful to form a group only of the people residing in the left front sector of a hall; instead, the complete hall should be considered. For these reasons, all the defined locations in the hierarchy are tagged, whether they allow the formation of groups or not. Dynamic groups are also not only formed granularity of rooms, but also on higher levels in the hierarchy, for example with the people currently residing in the area of a department.

As the members of dynamic groups constantly change, it is possible to *create an open static group* out of them.

4.4.3 Synchronous Communication for Groups

The most important form of *synchronous communication* on computers today is *instant messaging*; some people even see instant messaging to be the real killer application on the Internet. This has also motivated the decision to build GISS upon an instant messaging system.

In today's messenger systems, peer-to-peer-communication is extensively supported. However, when it comes to communication in groups, the support is rather poor most of the time. Often, only sending a message to multiple recipients is supported, lacking means to take into account the current state of the recipients. Furthermore, groups can only be formed of members in one's contact list, thus being not able to send messages to a group, where not all of its members are known (which may be the case in settings, where the participants of a lecture form a group).

Our approach does not have the mentioned restrictions. We introduce *group-entries in the user's contact list*; enable him or his to send messages to this group easily, without knowing who exactly is currently a member of this group. Furthermore, group messages are only delivered to persons, who are currently not busy, thus preventing a disturbance by a message, which is possibly unimportant for the user.

These features cannot be carried out in the messenger network itself, so whenever a message to a group account is sent, we intercept it and route it through our system to all the recipients, which are available at a certain time. Communication via a group account is also stored centrally, enabling people to query missed messages or simply viewing the message history.

4.4.4 Asynchronous Communication for Groups

Asynchronous communication in groups is not a new idea. The goal of this approach is not to reinvent the wheel, as email is maybe the most widely used form of asynchronous communication on computers and is broadly accepted and standardized. In our work, we aim at the *combination of asynchronous communication with location awareness*.

For this reason, we introduce the concept of so-called *virtual post-its* (cp. [13]), which are messages that are bound to physical locations. These virtual post-its could be either visible for all users that are passing by or they can be restricted to be visible for certain groups of people only. Moreover, a virtual post-it can also have an expiry date after which it is dropped and not displayed anymore. Virtual post-its can also be commented by others, thus providing some form of *forum-like interaction*, where each post-it forms a thread.

Virtual post-its are displayed automatically, whenever a user (available) passes by the first time. Afterwards, post-its can be accessed via the 2D-viewer, where all visible post-its are shown. All readers of a post-it are logged and displayed when viewing it, providing some sort of awareness about the group members' activities in the past.

4.4.5 Context-aware Availability Management

Instant messengers in general provide some kind of *availability information* about a user. Although this information can be only defined in a very coarse granularity, we have decided to use these means of gathering activity context, because the introduction of an additional one would strongly decrease the usability of the system.

To support the user managing his availability, we provide an interface that lets the user define rules to *adapt his availability to the current context*. These rules follow the form "on *event (E)* if *condition (C)* then *action (A)*", which is directly supported by the *ECA-rules of the Context Framework* described in chapter 1.3.

The testing of conditions is periodically triggered by throwing events (whenever the context of a user changes). The condition itself is defined by the user, who can demand the change of his availability status as the action in the rule. As a condition, the user can define his location, a certain time (also triggering daily, every week or every month) or any logical combination of these criteria.

4.4.6 Context-Aware Reminders

Reminders [14] are used to give the user the opportunity of defining tasks and being reminded of those, when certain criteria are fulfilled. Thus, a reminder can be seen as a *post-it to oneself*, which is only visible in certain cases. Reminders can be bound to a certain place or time, but also to spatial proximity of users or groups. These criteria can be combined with Boolean operators, thus providing a powerful means to remind the user of tasks that he wants to carry out when a certain context occurs.

A reminder will only pop up the first time the actual context meets the defined criterion. On showing up the reminder, the user has the chance to resubmit it to be reminded again, for example five minutes later or the next time a certain user is in spatial proximity.

4.4.7 Context-Aware Recognition and Notification of Group Meetings

With the available context information, we try to *recognize meetings of a group*. The determination of the criteria, when the system recognizes a group having a meeting, is part of the ongoing work. In a first approach, we use the location- and activity-context of the group members to determine a meeting. Whenever more than 50 % of the members of a group are available at a location, where a meeting is considered to make sense (e.g. not on a corridor), a *meeting minutes post-it* is created at this location and all absent group members are notified of the meeting and the location it takes place.

During the meeting, the *comment-feature of virtual post-its* provides a means to take notes for all of the participants. When members are joining or leaving the meeting, this is automatically added as a note to the list of comments.

Like the recognition of the beginning of a meeting, the recognition of its end is still part of ongoing work. If the end of the meeting is recognized, all group members get the complete list of comments as a meeting protocol at the end of the meeting.

5. CONCLUSIONS

This paper discussed the potentials of support for group interaction by using context information. First, we introduced the notions of context and context computing and motivated their value for supporting group interaction.

An architecture is presented to support context-aware group interaction in mobile, distributed environments. It is built upon a flexible and extensible framework, thus enabling an easy adoption to available context sources (e.g. by adding additional sensors) as well as the required form of representation.

We have prototypically developed a set of services, which enhance group interaction by taking into account the current context of the users as well as the context of groups itself. Important features are dynamic formation of groups, visualization of location on a two-dimensional map as well as unobtrusively integrated in an instant-messenger, asynchronous communication by virtual post-its, which are bound to certain locations, and a context-aware availability-management, which adapts the availability-status of a user to his current situation.

To provide location information, we have implemented a subsystem for automated acquisition of location- and proximity-information provided by various sensors, which provides a technology-independent presentation of locations and spatial proximities between users and merges this information using sensor-independent fusion algorithms. A history of locations as well as of spatial proximities is stored in a database, thus enabling context history-based services.

6. REFERENCES

- [1] Beer, W., Christian, V., Ferscha, A., Mehrmann, L. *Modeling Context-aware Behavior by Interpreted ECA Rules*. In Proceedings of the International Conference on Parallel and Distributed Computing (EUROPAR'03). (Klagenfurt, Austria, August 26-29, 2003). Springer Verlag, LNCS 2790, 1064-1073.
- [2] Brown, P.J., Bovey, J.D., Chen X. *Context-Aware Applications: From the Laboratory to the Marketplace*. IEEE Personal Communications, 4(5) (1997), 58-64.
- [3] Chen, H., Kotz, D. *A Survey of Context-Aware Mobile Computing Research*. Technical Report TR2000-381, Computer Science Department, Dartmouth College, Hanover, New Hampshire, November 2000.
- [4] Dey, A. *Providing Architectural Support for Building Context-Aware Applications*. Ph.D. Thesis, Department of Computer Science, Georgia Institute of Technology, Atlanta, November 2000.
- [5] Svetlana Domnitcheva. *Location Modeling: State of the Art and Challenges*. In Proceedings of the Workshop on Location Modeling for Ubiquitous Computing. (Atlanta, Georgia, United States, September 30, 2001). 13-19.
- [6] Ferscha, A. *Workspace Awareness in Mobile Virtual Teams*. In Proceedings of the IEEE 9th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'00). (Gaithersburg, Maryland, March 14-16, 2000). IEEE Computer Society Press, 272-277.
- [7] Ferscha, A. *Coordination in Pervasive Computing Environments*. In Proceedings of the Twelfth International IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE-2003). (June 9-11, 2003). IEEE Computer Society Press, 3-9.
- [8] Leonhard, U. *Supporting Location Awareness in Open Distributed Systems*. Ph.D. Thesis, Department of Computing, Imperial College, London, May 1998.
- [9] Ryan, N., Pascoe, J., Morse, D. *Enhanced Reality Fieldwork: the Context-Aware Archaeological Assistant*. Gaffney, V., Van Leusen, M., Exxon, S. (eds.) Computer Applications in Archaeology (1997)
- [10] Schilit, B.N., Theimer, M. *Disseminating Active Map Information to Mobile Hosts*. IEEE Network, 8(5) (1994), 22-32.
- [11] Schilit, B.N. *A System Architecture for Context-Aware Mobile Computing*. Ph.D. Thesis, Columbia University, Department of Computer Science, May 1995.
- [12] Wang, B., Bodily, J., Gupta, S.K.S. *Supporting Persistent Social Groups in Ubiquitous Computing Environments Using Context-Aware Ephemeral Group Service*. In Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04). (March 14-17, 2004). IEEE Computer Society Press, 287-296.
- [13] Pascoe, J. *The Stick-e Note Architecture: Extending the Interface Beyond the User*. Proceedings of the 2nd International Conference of Intelligent User Interfaces (IUI'97). (Orlando, USA, 1997), 261-264.
- [14] Dey, A., Abowd, G. *CybreMinder: A Context-Aware System for Supporting Re-minders*. Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing (HUC'00). (Bristol, UK, 2000), 172-186.