**IST BASIC RESEARCH PROJECT**
**SHARED COST RTD PROJECT**
**THEME: FET DISAPPEARING COMPUTER**
**COMMISSION OF THE EUROPEAN COMMUNITIES**
**DIRECTORATE GENERAL INFSO**
**PROJECT OFFICER: THOMAS SKODAS**

**Global Smart Spaces**

# Proximity Based Group Communications for Mobile Ad Hoc Networks

D.14

03/10/2003/TCD/WP6/VFINAL

ANDRONIKOS NEDOS, KULPREET SINGH, SIOBHÁN CLARKE

| | | | |
|---|---|---|---|
| **IST Project Number** | IST-2000-26070 | **Acronym** | GLOSS |
| **Full title** | Global Smart Spaces | | |
| **EU Project officer** | Thomas Skodas | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Deliverable** | **Number** | D 14 | **Name** | Binding model with partial failure model | | | |
| **Task** | **Number** | T | **Name** | | | | |
| **Work Package** | **Number** | WP 6 | **Name** | Theories of Mobility | | | |
| **Date of delivery** | **Contractual** | | PM 30 | **Actual** | 03/10/2003 | | |
| **Code name** | <codename> | | | **Version** 1.0 draft ☐ final ☑ | | | |
| **Nature** | Prototype ☐ Report ☑ Specification ☐ Tool ☐ Other: | | | | | | |
| **Distribution Type** | Public ☑ Restricted ☐ to: <partners> | | | | | | |
| **Authors (Partner)** | Trinity College Dublin | | | | | | |
| **Contact Person** | Dr. Siobhán Clarke | | | | | | |
| | **Email** | Siobhan.Clarke@cs.tcd.ie | **Phone** | +353-1-6082224 | **Fax** | +353-1-6772204 | |
| **Abstract (for dissemination)** | This document describes a formal specification for a service that provides reliable and efficient connectivity to support communication amongst groups of devices in a mobile setting. Further, we describe an implementation of Hearsay (one of Gloss's conceptual tools) that supports hearsay based on both location and time. | | | | | | |
| **Keywords** | Group Communication, Reliable Pervasive Applications, Context-based messaging | | | | | | |

# CONTENTS

**CONTENTS** ........ **3**
**1 INTRODUCTION** ........ **4**
**2 PBGC WITHIN CONTEXT OF GLOSS SCENARIO** ........ **5**
*2.1* SHARED DEVICES REQUIRING MUTUAL EXCLUSION........ 5
*2.2* RELIABLE MESSAGE DISSEMINATION........ 5
*2.3* COLLABORATIVE WORK........ 6
**3 PBGC SPECIFICATION** ........ **8**
*3.1* INTRODUCTION ........ 8
*3.2* PBGC REQUIREMENTS ........ 9
*3.3* PBGC SYSTEM MODEL AND ASSUMPTIONS ........ 9
*3.3.1 Mathematical Model ........ 10*
*3.3.2 Stable Components and Failure Detectors ........ 12*
*3.3.3 Stability Conditions ........ 15*
*3.3.4 Partition Anticipator ........ 15*
*3.4* SPECIFYING A PROXIMITY GROUP MEMBERSHIP SERVICE ........ 16
*3.5* SPECIFYING A PROXIMITY GROUP DELIVERY SERVICE ........ 19
*3.5.1 Virtual Synchrony ........ 19*
*3.5.2 Ordering Properties ........ 22*
**4 IMPLEMENTING HEARSAY WITH LATTE** ........ **23**
*4.1* EMAIL-BASED HEARSAY SERVICE ........ 23
*4.1.1 Location ........ 24*
*4.1.2 Time ........ 25*
*4.1.3 Identity ........ 25*
*4.2* IMPLEMENTATION ........ 26
*4.2.1 Overall Architecture ........ 26*
*4.2.2 Sender Model ........ 26*
*4.2.3 Context Engine ........ 27*
*4.2.4 Extending LATTE with PBGC ........ 27*
**5 CONCLUSION** ........ **29**
**6 REFERENCES** ........ **30**

# 1 INTRODUCTION

One of the central objectives for Gloss is to develop essential middleware infrastructures for hybrid mobile ad hoc, peer-to-peer and hierarchical systems. The work on proximity-based group communications (PBGC) described in this document is positioned in the mobile ad hoc part of such an infrastructure. PBGC facilitates reliable dissemination of information to a person on the move, and reliable and consistent sharing of information between groups of people who come together in an ad hoc fashion. For example, as illustrated in the Gloss Scenario [2], Bob receives a personal message as he wanders around Paris; Bob and Jane start to work together in a café in Paris; Bob receives messages from other mobile users around him indicating where there are crowds so he can mingle. Important properties relating to these parts of the scenario are mobility, information dissemination and ad hoc communication.

The Group Communication paradigm [3, 4, 5, 6] has already proven to be a useful tool for building reliable distributed systems. In our work, we are augmenting traditional group communication systems with the notion of a group's "proximity". Proximity captures the intrinsic property of geographic location of mobile nodes, which can be exploited to build several location based services and applications for mobile ad-hoc networks. PBGC can be employed in mobile ad-hoc networks to support consistent sharing of information between the nodes that are in proximity of each other. The asynchronous nature of the communication in such a scenario requires state replication amongst the relevant nodes, reliable message dissemination and mutual exclusion for shared state. PBGC is suitable for providing the appropriate building blocks to support such communication as it takes into consideration the dynamic nature of membership due to mobility, and provides message delivery guarantees within a geographic location.

In Gloss, the network services that connect people and artefacts may be abstracted into the *core* network, which constitutes the backbone of the communication infrastructure handling traffic between remote locations (e.g. Paris-Brussels in the scenario) and the *edge* network, which provides the infrastructure for smaller-scale local interactions. This separation provides the best balance between scalability, cost and reliability. Communication in the edge network takes place primarily between mobile nodes connecting in an ad hoc fashion, enabling local interactions to be achieved with reduced cost and reduced power consumption, by using no fixed infrastructure that requires powerful transceivers or investment in infrastructure. At the same time performance characteristics such as latency are improved, since no intermediate entities are being used. However the multi-hop nature of the mobile ad hoc mode of operation decreases reliability due to frequent disconnections caused by mobility. PBGC provides guarantees to counteract this inherent unreliability within a geographic location.

The main body of work described in this document is a formal specification for reliable and efficient connectivity for groups of devices in a mobile setting. Further, we describe an implementation of Hearsay (one of Gloss's conceptual tools) that supports hearsay based on both location and time. The prototype system, called LATTE (Location and Time Triggered Email), has an internal events-based architecture that illustrates the need for a supporting middleware that provides reliable messaging within a location. This prototype provides an ideal platform on which to evaluate the development of PBGC. First, though, the next section discusses PBGC in the context of the Gloss Scenario [2].

# 2 PBGC WITHIN CONTEXT OF GLOSS SCENARIO

In this section we discuss the use of mobile group communications for Gloss by analysing occurrences in the Gloss scenario where we have identified a strong requirement for building blocks such as those provided by PBGC. These use-cases contribute to the requirements on which the specification for PBGC is based.

## *2.1* SHARED DEVICES REQUIRING MUTUAL EXCLUSION

As illustrated in Figure 1, there are situations when Bob is travelling from Brussels to Paris where potentially many users may wish to use the same resource. For example, in Figure 1(a), Bob is on his way to Paris, and he uses active walls situated inside the station. The same usage pattern re-appears in Figure 1(b) when Bob is sitting inside the café in Paris attempting to use the active wall situated inside.

The red route is not busy this morning. As Bob is walking close to an active wall, he is presented a message relevant to his trip in Paris

***(a) Bob at the Train Station***

When Bob sits down in the café, the active surface on the table informs him of the local history of the café. An active wall displays a selection of postcards for his perusal.

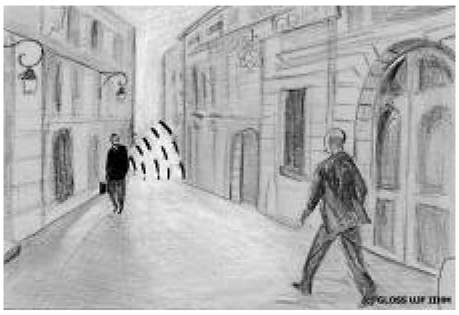
***(b) Bob in a café in Paris***

**Figure 1: Need for Mutual Exclusion**

Although the active wall is a shared device that can be used by any of the people in proximity, its actual functioning requires exclusive use by each client. This is then a problem of mutual exclusion to a shared resource, which has received particular attention in classic distributed systems literature [7], but which reappears here in the context of mobile ad hoc computing. Though any algorithms from the existing literature can be used for solving it, solutions for this problem require the underlying network to have the ability for reliable broadcast. *The current specification for Proximity Groups offers exactly that in the form of broadcasting to the members of a proximity group that is defined in the vicinity of the active wall.*

## *2.2* RELIABLE MESSAGE DISSEMINATION

Gloss explored the notion of location-specific notifications and captured it in a conceptual tool called Hearsay as a form of ambient communication. In Figure 2, we illustrate one Hearsay example from the Gloss scenario. While Bob is in Paris, he receives a notification when he is in the proximity of a café that his friend in Denmark recommended.



Now knowing the right direction, Bob starts to move towards the Pompidou Centre. After Bob has walked a block, his personal communications device vibrates.

**Figure 2: Need for Reliable Message Dissemination**

Implementing this concept in a peer-to-peer way over mobile ad hoc networks decreases latency and communication costs by keeping all communication in the locality of the relevant devices. A proximity groups framework can help establish an awareness zone in the vicinity of an area (i.e. the café in this case). Using this service, a trivial service discovery protocol can provide notification to interested parties, as well as help expand the awareness zone, by exploiting the multi-hop aspects of the network. As a result, a proximity group composed of devices inside a designated area containing that region in their list of preferences will receive a notification, while other devices remain unaffected.

## *2.3* COLLABORATIVE WORK

An occurrence of collaborative work between users that gather in an ad hoc fashion appears in the Gloss scenario, as illustrated in Figure 3. While in the restaurant in Paris, Bob and Jane interact with the table in the restaurant, and use shared objects to illustrate ideas.

When Bob and Jane have eaten, they bring out the work that they had started earlier and continue to work on it. They use shared objects to illustrate their ideas for the layout of the site they are planning together.

**Figure 3: Need for Collaborative Work**

This is an example of collaborative work enabled when mobile wireless devices come into communication range. The communication pattern in this instance is in a one-to-many or many-to-many fashion, which is the pattern used in group communications. A proximity group service in this case can offer a substrate for reliable transmission and ordering of messages sent between group participants, while transparently adapting the membership when other interested devices enter the proximity of the active table. In general, collaborative work between mobile participants can greatly benefit from such a service as the necessary functions of broadcast and ordering are abstracted in the middleware, so applications can reuse them accordingly.

# 3 PBGC SPECIFICATION

## *3.1* INTRODUCTION

This section provides a formal specification for a proximity based group communication service. High-level concepts and requirements are initially described, followed by detail of the system model properties on which the service will reside. We then formally specify the service against these properties.

Members in a group (each of which we refer to as a "node") share information, and need a consistent view of that information. To achieve this, nodes need a consistent view of all the other nodes involved in the communication. All group communication services have a membership service part, and the service for the mobile, ad hoc scenario we are targeting is no different. In this case, geographical location of nodes determines if they can become members of a group. We define later how the proximity of a group and the location of nodes help in establishing the group's membership. The membership service provides a dynamic view of the current membership to all nodes in the group.

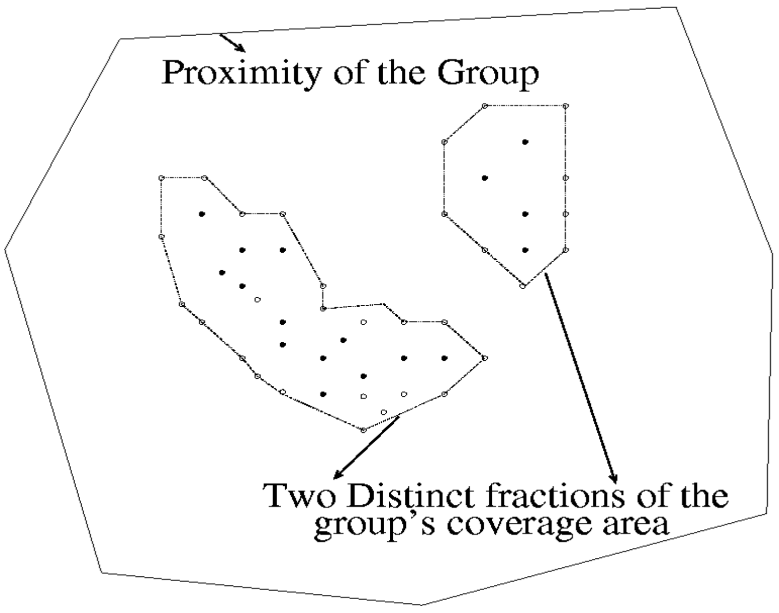


**Figure 4 Disjoint Coverage Areas**

We define a group to have a predefined proximity associated with it. This proximity describes the geographical region in which the group can exist. Only the nodes that are present within the defined proximity can participate in the group's communication. It is not necessary that all nodes in a region be group members. Nodes have to explicitly issue a request to join a proximity group's communication. When nodes leave the proximity, they are removed from the group's membership.

The radio ranges of nodes in a group cover a certain geographical area, as shown in Figure 4. This geographical area is called a "coverage area". A group's predefined proximity can be partitioned into a number of coverage areas, each of which would then correspond to a "partition" of a group. Partitions occur when a subset of members move out of radio range of other members. It is possible due to the mobile nature of nodes, that not all of the group's proximity area will be covered by the sum of the coverage areas.

Information about coverage areas can help build a service that "anticipates" partitions and provides such information to applications. Prior knowledge of when a partition is likely to happen allows an application behave in an "orderly" manner – in other words, in a manner that is appropriate for the application's semantics.

In this document we rigorously specify a proximity based group communication service for mobile ad-hoc networks. We also formally specify the behaviour of the partition anticipator and elaborate how it helps in providing a mobile group communication with the guarantees specified.

## *3.2* PBGC REQUIREMENTS

The responsibilities of the PBGC service for mobile ad-hoc networks are listed below. Later we try and map these high level requirements to rigorous specifications.

1. If a group exists, all member nodes have access to the view of the partition they are in.
2. When a new node enters a 'coverage area' of a group, it is provided with information about the group and an opportunity to join. A new node can choose to join the group either immediately or later. The group's information should be available as long as there is any member of a group present in the proximity.
3. If nodes from a certain coverage area leave each other's vicinity, that coverage area is then split into partitions. The group continues with disparate partitions.
4. Partitions from the same group merge when their coverage areas overlap/intersect.
5. When partitions merge, the application is given an opportunity to initiate state transfer.
6. If, at start-up, a group has disjoint partitions within the same proximity, the partitions are *unaware* of each other.
7. A node can be a member of more than one group.
8. Messages are delivered, in an agreed order, to all members of a group that are in the same partition. If a group has partitioned, the members in different partitions continue to maintain the local total order for that partition.

The nodes involved in the group communication service are mobile devices, requiring correct handling of frequent topology changes and of a dynamic (not predetermined) set of interacting processors.

## *3.3* PBGC SYSTEM MODEL AND ASSUMPTIONS

Previous work on Group Communication Services is primarily based on fixed infrastructure networks. However, the dynamic nature of mobile ad-hoc networks has a significant impact on a group communication service's specification, affecting the system model and the assumptions for the system. This section describes the system model and assumptions for the PBGC service, which are an extension to those used by the Chockler survey [6]. The Chockler survey establishes a framework for specifying the properties of group communication systems and presents the properties of various group communication systems using the framework. The survey is the latest such effort and provides a solid ground to develop, compare and contrast properties of different group communication systems.

The assumptions that are different to the Chockler paper are marked with a (•), and the assumptions that have no counter-part in the Chockler system model are marked with (†).

1. • The system consists of a dynamic set of processors[1]. This implies that processors can dynamically join and leave the system.
2. Processors communicate using message passing.
3. †• The underlying network provides unreliable *node-to-neighbours* datagram communication facility.
4. † A multi-hop unreliable datagram based communication facility can be built on top of the node-to-neighbours communication facility.
5. † A node's neighbours can dynamically change. This can happen either because of node mobility or failures of nodes.
6. † We assume all nodes will participate in the ad-hoc network, even if they are not part of a group.
7. Nodes can crash and recover.
8. Nodes have stable storage on them.
9. Failures can partition the communication network into disjoint components, as described in Section 3.1.
10. Disjoint components can merge.
11. We assume the absence of Byzantine failures.
12. †• We assume the lack of *eventually perfect failure detector*. This is because the dynamics of a mobile ad-hoc network make it unsuitable to implement an eventually perfect failure detector. This is further elaborated in section 3.3.2

### *3.3.1 MATHEMATICAL MODEL*

We follow the guidelines presented in the framework developed in [6] and model the system as an untimed I/O automaton. We present the service specifications by defining its external signature in this section and its trace properties in section 3.4.

#### *3.3.1.1 ESSENTIAL TYPES*

The essential types used in the specification are listed below:

$P$ — The *dynamic* set of processors that the system can have at a given instance. This set is never truly fully determined, and thus should not be used in implementation details. The set is used here only for elucidating various properties.

$M$ — The set of messages sent by the application.

$VID$ — The set of view identifiers.

[1] A "processor" is equivalent to a "node"

$V$ — The set of views delivered to the application. The view is a pair containing of a view identifier and the set of members.

### *3.3.1.2 EXTERNAL ACTIONS*

Our system utilises a reduced set of *external actions* (**Events**) as defined by the Chockler paper[2]. The reduced set that we use are listed below —

$\mathbf{send}(p,m)\ |\ p\in P,\ m\in M$ — Processor $p$ sends a message $m$.

$\mathbf{recv}(p,m)\ |\ p\in P,\ m\in M$ — Processor $p$ receives a message $m$.

$\mathbf{crash}(p)\ |\ p\in P$ — Processor $p$ is reported to have crashed. **crash** encapsulates all types of *crash* and *communication* failures.

$\mathbf{recover}(p,m)\ |\ p\in P$ — Processor $p$ is reported to have recovered from a crash. If a processor that has not been participating in a group enters the proximity of a group and wants to start participating a group, the same event occurs.

$\mathbf{view\_chng}(p,\langle id,members\rangle)\ |\ p\in P, id\in VID, members\in 2^P$ — Processor $p$ installs a new view with identity $id$ and members as in the set of processors *members*.

The properties of the PBGC system use a collection of trace properties, where a **trace** is defined as a sequence of external actions the automaton accepts. For our purposes, we fix a trace $t_1,t_2,\ldots$, and all properties are stated with respect to the trace.

If an event $t_i$ occurs in the context of a view, we say the $viewof(t_i)$ is last installed at that processor.

### *3.3.1.3 PROXIMITY AND COVERAGE AREA*

To be able to define the proximity and coverage area of a group, we need to define the properties of location of a processor and proximity:

1. **Location of a node** — the geographical point occupied by the centre of gravity of the node. This location is captured in terms of abstract coordinates.

Now we can define the proximity of a group as —

2. **Proximity** — the geographical area, $PXM(G)$ associated with a group $G$, from which mobile nodes can participate in the group's communication. Proximity of a group can take arbitrary shape. A node $p$ is defined to be *inProximity* of a group $G$ as follows —

[2] We do not include the **safe_prefix** event defined in the survey as we choose the approach of delivering only those messages which have been determined to be stable. Such an approach allows a faster progress when the network is a dynamic one, as in the case for ad-hoc networks. We also do not include the *Transition Set* as part of the **view_chng** event as we do not use the notion of *Extended Virtual Synchrony* – see Section 3.5.1. for further discussion on Virtual Synchrony.

**inProximity** $(p, G)$ if $location(p) \in PXM(G)$

The geographical area covered by the radio-ranges of the members of a partition is defined as the Coverage Area for that partition. If the group has only one partition at a certain instance, it will have one coverage area. The coverage area of a partition of a group is bounded by the proximity of the group. Any geographical point within the radio range of a node that lies outside the group's proximity is not considered a part of the coverage area. Each partition of a group has its own coverage area, and thus the definition —

3. **Coverage Area** — the geographical area $CA$ covered by the radio ranges of the processors in a partition and bounded by the proximity of the group. All the processors $p \in CA$ are reachable from each other. The following two events are used to signal when two processors have become reachable or unreachable from each other.

**reachable** $(p,q)$ if $p, q \in CA_i \wedge$

$inProximity(p, G) \wedge inProximity(q, G)$

**unreachable** $(p,q)$ if $p \in CA_i \;\wedge\; q \in CA_j \;\wedge\; CA_i \cap CA_j = \varnothing \vee$ if

$inProximity(p, G) = false \vee inProximity(q, G) = false$

The above definition allows us to use the events "reachability" and "unreachability" in our system specifications. We are targeting a system where an explicit failure detector does not generate these events. Each processor "reaches a consistent conclusion" based on the messages sent and received by it.

### 3.3.2 STABLE COMPONENTS AND FAILURE DETECTORS

In this section we first present stability conditions and failure detectors used in specifications developed by Chockler et al in [6] and Babaouglu et al in [10]. Both are useful, if not essential, to provide guarantees for message delivery within a group. We further describe how the models in [6, 10] relate to our system model. And finally, we present the stability conditions for our specification.

#### 3.3.2.1 STABLE COMPONENT

In [10], stability conditions are provided using abstract properties for a failure detector. These properties are adapted for a partitionable system using the fundamental failure detector properties developed in [11].

The framework developed in [6] provides for stability conditions using a "stable component" and the abstract failure detector properties developed by [10]. A stable component is achieved when there is a stable network. A network is considered stable from the point in time when no processes crash or recover, communication becomes symmetric and transitive and no changes occur in network connectivity. For systems that do not support the notion of a stable component, the Chockler et al framework provides for alternative properties, even though these alternative properties are quoted to be "not particularly useful" for applications.

Our system needs to satisfy the requirements of a stable component. That is, if we look at a particular partition (or a coverage area) where no processes crash or recover, we can say that the partition satisfies the properties of the coverage area definition. But if we take a closer look, when processors move from the neighbourhood of one processor $p$ to that of another processor $q$, the immediate reachability changes contradicting the "no changes in network connectivity" requirement. Similarly, $p$ and $q$ would have differing observations about processors that have failed and recovered; conflicting the "no processors crash and recover" requirement.

#### *3.3.2.2 FAILURE DETECTORS*

The mobile and multi-hop network environment introduces dependencies from communication failure on to mobility of the processors and crash failures. This section discusses how such dependencies affect our ability to provide a failure detector. We limit our discussion to abstract failure detectors and their *completeness* and *accuracy* properties as introduced in [11].

The completeness property of a failure detector pertains to detecting a failure and informing the processors in the system about this failure. The accuracy property of a failure detector pertains to detecting a correct process that the processors in the system are not yet aware of. In [11] these properties are defined as —

4. **Completeness** — there is a time after which every process that crashes is permanently suspected by some correct process.

5. **Accuracy** — there is a time after which some correct process is never suspected by any correct process.

[11] presents a classification of failure detectors along the axis of varying degree of completeness and accuracy properties supported. There are 2 completeness and 4 accuracy properties described and thus 8 classifications of failure detectors provided. An algorithm $T_{D \to D'}$ is presented which reduces the study of failure detectors to only consider 4 of the 8 classes of failure detectors. This is achieved by reducing one of the two completeness properties to the other, resulting in 1 completeness and 4 accuracy properties.

We note here that unlike the system model in [11], the processors in our system are not "fully connected". We also assume the set $\Pi$ ($P$ in our case) of processors is not known. This limits the application of the algorithm $T_{D \to D'}$, as proposed in the aforementioned paper, to reduce Weak Completeness to Strong Completeness of failure detectors. The limitation is caused by the use of "$\forall q \in \Pi$: send $m$ to $q$". It is implicit in the algorithm that these sends require a connection to all the processors in $\Pi$, which we can only emulate using a "broadcast domain"[12]. We argue that if we use a broadcast domain to provide "strong completeness" guarantees, we can directly build a group communication service without requiring an explicit distributed failure detector running on each node.

Traditional group communication services implement a failure detector using a *pinging* service. In Relacs [5], each processor *pings* each other processor it is aware of and determines failures based on time outs for responses to the pings. A direct port of such a service to mobile ad-hoc networks would require being able to *ping* the mobile processors in the network, requiring routing paths to exist between these nodes.

Maintaining routes between processors in a mobile ad-hoc network itself requires the routing service to somehow broadcast route requests and responses, over the multi-hop network. Ensuring all your processors are able to *ping* all other processors in the network would require all these routes to be maintained at all times, limiting the benefits of "on-demand" routing protocols and adding redundancy.

We argue that we can replace such a *ping* service by a periodic beacon broadcasted to immediate neighbours only. For the system model under examination, an alternative to implementing a failure detector, is to have all processors periodically geocast an "I am Alive" beacon to the proximity of the groups it is participating in. This is different from the algorithms suggested by [11] and [10], as those algorithms require an explicit connection from each processor in $P$ to every other processor in $P$; which requires an implicit knowledge of all processors in $P$ and routes to all of them, which cannot be assumed for our system model.

The geocasting of "I am Alive" messages as suggested above can be used to provide completeness and accuracy conditions for detecting failed and new processes as discussed in section 3.3.3. We propose that such a geocast can help us implement a membership service without requiring an "explicit" failure detector module running on each of the processors. But it is important to remember that our service does provide the semantics and guarantees as required by a failure detector defined in section 3.4.

If no regular communication messages are being exchanged, "null" messages can be exchanged to keep the protocol from making progress and adhere to the liveness properties; this is motivated by solutions proposed by [9, 13, 4].

Another alternative to failure detectors for this system model is to provide a stateful local failure detector module at each node that keeps track of its neighbours. This can be done by simple neighbour discovery services that have been proposed in the literature [25]. Using a neighbour discovery service would help scale the system and would work without the need of the knowledge of the set $\Pi$ of all processors in the system. The problem faced with such a solution is that when a processor $r$ moves from $p$'s neighbourhood to $q$'s neighbourhood, $p$ would detect this as a failure and try to let the other processors in the system know of this failure. At the same time $q$ would detect a "new" processor and would try to make such information available to the rest of the processors.

*FAILURE DETECTORS AND PARTITIONS*

Another problem in using the neighbourhood service to implement a failure detector occurs in the case of partitions. Let us assume two disjoint subsets $L$ and $M$ of processors connected via a hop through processor $p$. Let us further look at the situation where processors $r \in L$ and $q \in M$ are in the neighbourhood of processor $p$. If $p$ fails, $r$ and $q$ would detect $p$ to have failed and inform the processors in their corresponding partitions, $L$ and $M$. The problem is that none of the local failure detector components running in $L$ and $M$ would be able to detect the failure of the rest of the processors in the other partition.

Such a system is possible to implement as noted in [8]. After discussing and proving the impossibility of a group membership service for a primary component service, [8] argue that using failure detectors or probabilistic algorithms can help implement a primary component membership service. They also state that a partitionable group membership services bypasses the impossibility presented in the paper. We will focus our future work on not using a failure detector for implementing the membership service, but instead depend on the fact that our service is partitionable.

### *3.3.3 STABILITY CONDITIONS*

PBGC will provide a group communication service where a traditional view of a "stable component" is not present. The system model also limits the possibility of implementing a separate failure detector. We enforce our **reachable** and **unreachable** events to satisfy the accuracy and completeness properties of the failure detectors as developed in [10].

In [10], properties are provided that allow applications meaningful means of reliable group communication even without explicit requirement of a stable component. Since our system cannot confirm with the requirements of a stable component as described above, we adapt some of the properties developed in [10] for our purposes.

The adaptations made in [10] for *Strong Completeness* and *Eventual strong Accuracy* lead to a very different failure detector from the $\Diamond P$ failure detector presented in [11]. Babaoglu et. al. [10] use the notion of their $\Diamond P$ to argue about the correctness of their algorithms.

We evaluate our failure detector to be $\Diamond W$ as follows —

- Accuracy — when a processor enters a coverage area of a partition, its immediate neighbours immediately detect the presence of a new processor. These immediate neighbours then facilitate sharing the information about this new correct process with all other processors. Since our system is not guaranteed to maintain a certain topology for a long enough period, we assume information about a new processor may not reach every processor in the coverage area before the constituents of the coverage area change. This was illustrated by the example in Section 3.3.3.2 where processors $p$ and $q$ make conflicting observations about processor $r$. Thus we assume, that "some correct process is never suspected by any correct process" from [11] implying *Eventual Weak Accuracy*.
- Completeness — again following the situation involving processors $p, q$ and $r$ we can see that only the immediate neighbours of a failing processor are guaranteed to detect the failures. How this failure report is propagated and whether there are conflicting reports about the same processor allow us to only guarantee that "every process crash is permanently suspected by *some* correct process" from [11], implying *Weak Completeness* .

With the known guarantees for **reachable** and **unreachable** events, we can provide arguments for correctness of our specifications, along the lines of arguments presented in [11]. It is important to observe we use the notion of a $\Diamond W$ failure detector for the purposes of arguing about correctness, and we do not incorporate an explicit failure detector in our system.

### *3.3.4 PARTITION ANTICIPATOR*

A partition anticipator (PA) is a "distributed oracle" similar to a distributed failure detector. This means that each processor runs an instance of the PA, and each instance contains no global knowledge. Each processor receives events from the instance of the PA running on the same processor. This section formalises the type of input that the membership service expects from a partition anticipator and how the service reacts to incoming events from the partition anticipator service.

The PA generates a $\textbf{anticipated\_chng}(V)$ event, which implies that the processors current view is anticipated to change to $V$. Formally —

$$\exists i\,(t_i) = \textbf{anticipated\_chng}(V) \wedge viewof\,(t_i) = V' \wedge V \neq V'$$

We should mention here that the PA is being treated as a black box with its function and characteristics being orthogonal to those of the membership service. Here we assume that whenever the PA generates events, it guarantees $V \neq V'$.

The group communication service's response to the events generated by the PA is linked tightly with the property *Optimistic Virtual Synchrony*, described in Section 3.5.1.

## *3.4* SPECIFYING A PROXIMITY GROUP MEMBERSHIP SERVICE

In the section we specify the group membership service for mobile ad-hoc networks. We do not explicitly differentiate between safety and liveness properties, but instead provide specifications to capture the higher level requirements listed in the section 3.2. The property names here are similar with those devised by Chockler et. al. in [6].

Before going into individual properties we repeat some shortcut predicates from [6] which will be used in presenting the properties that follow (see Table 1).

Table 1: Shortcut predicates used by Chockler

| | |
|---|---|
| $delivers(q,m)$ | $\exists i\; t_i = \textbf{recv}(q,m)$ |
| $delivers\_in(q,m,V)$ | $\exists i\; t_i = \textbf{recv}(q,m) \wedge viewof\,(t_i) = V$ |
| $installs(p,V)$ | $\exists i\; t_i = \textbf{view\_chng}(p,V)$ |
| $installs\_in(p,V,V')$ | $\exists i\; t_i = \textbf{view\_chng}(p,V) \wedge viewof\,(t_i) = V'$ |
| $recv\_before(p,m,m')$ | $\exists i \exists j\; (t_i = \textbf{recv}(p,m) \wedge t_j = \textbf{recv}(p,m') \wedge i < j$ |
| $recv\_before\_in(p,m,m',V)$ | $\exists i \exists j\; (t_i = \textbf{recv}(p,m) \wedge t_j = \textbf{recv}(p,m') \wedge$ $viewof\,(t_i) = viewof\,(t_j) = V \wedge i < j$ |

**Self Inclusion –** If a process $p$ installs view $V$, then $p$ is a member of $V$. Formally —

$$\exists i \;\; t_i = \textbf{view\_chng}(p,V)$$

**Local Monotonicity –** If a process $p$ installs view $V$ after installing view $V'$ then the identifier of $V$ is greater than that of $V'$. Formally —

$$t_i = \textbf{view\_chng}(p,V) \wedge$$

$$t_j = \textbf{view\_chng}(p,V') \wedge i > j$$

$$\Rightarrow V.id > V'.id$$

This property makes sure that a process does not install the same view twice and if two processes install the same views they install them in the same (partial) order.

**Initial View Event –** Every send or receive event occurs within some view.

$$t_i = \textbf{send}(p,m) \vee$$

$$t_i = \textbf{recv}(p,m) \vee$$

$$\Rightarrow viewof(t_i) \neq \perp$$

**Membership Accuracy –** If there is a time after which processes $p$ and $q$ are alive and are within the same coverage area, then $p$ eventually installs a view that includes $q$, and every view that $p$ installs afterwards also includes $q$. Formally —

$$\exists i \, \forall \, l > i \; t_l \neq \textbf{crash}(p) \quad \wedge$$

$$\exists i \; k < i \; t_k = \textbf{reachable}(p,q) \quad \wedge$$

$$\exists i \, \forall \, m > i \; t_m \neq \textbf{unreachable}(p,q) \quad \wedge$$

$$\Rightarrow \exists j \, \exists V s.t.$$

$$t_j = \textbf{view\_chng}(p,V) \quad \wedge \quad q \in V.members$$

$$\wedge \quad \forall k > j \quad \forall V' \; t_k = \textbf{view\_chng}(p,V') \Rightarrow q \in V'.members$$

This property eliminates solutions to the group membership where processors $p$ and $q$ are reachable from each other (directly or indirectly) and neither of them install a view with both as members of the view.

On the other hand, this property does not require processors that are connected to each other to install the same view, it only requires them to include each other in their view. This requires us to provide a property not considered in the framework [6]. The relevant property is later presented as *View Coherency*.

**Termination of Delivery –** [3] If a process $p$ sends a message $m$ in a view $V$, then for each member $q$ of $V$, either $q$ delivers $m$, or $p$ installs a next view $V'$ in $V$. Formally —

$$\exists l \; t_l = \textbf{send}(p,m) \wedge viewof(t_l) = V \quad \wedge$$

$$q \in members(V)$$

[3]This property also implies *Self Delivery* property. Which states that a correct process delivers messages sent by itself.

$\Rightarrow delivers(q,m) \vee$

$\exists V' \, installs\_in(p,V',V)$

This property captures the liveness of the membership service by requiring that if a message is not delivered at some member of a view (implying that the other member is no longer reachable), then the sender of the message installs a new view with the failed process eliminated from the view. Thus our service is forced to make progress one way or the other and hence eliminates use of algorithms that do not make progress.

**View Coherency –** [4] The property can be presented as follows —

1. If a correct process $p$ installs a view $V$, then either all members of $V$ install $V$ or $p$ eventually installs an immediate successor to $V$. Formally —

   $\exists i \; t_i = installs(p,V) \Rightarrow \forall \, q \in V.members \; installs(q,V) \vee$

   $t_{i+1} = installs(p,V')$

2. If two processors $p$ and $q$ initially install the same view $V$ and $p$ later installs an immediate successor to $V$, then eventually either $q$ also installs an immediate successor to $V$, or $q$ crashes. Formally —

   $\exists i \; t_i = installs(p,V') \quad \wedge$

   $\exists j < i \; t_j = installs(p,V) \quad \wedge$

   $\exists k < i \; t_k = installs(q,V)$

   $\Rightarrow \exists l > i \; t_l = installs(q,V') \quad \vee$

   $t_l = \mathbf{crash}(q)$

   It can be argued that the situation handled by this property is already handled by *Membership Accuracy*, but *Membership Accuracy* leaves the installation of the new view totally dependent on a failure detector[5]. On the other hand, this property forces all members of any new view to install the same view.

   In the face of frequent topology changes and disconnections, providing such a property would lead to a highly unstable membership service, but supporting this property will help in reflecting a true view of the system. If there is a need to hide the frequent changes, a higher layer can hide this instability for the application.

   A third and equally important aspect of view installation is presented as the property *Agreement on Successors*. That property enforces all processors going through the same succession of views to install all of those views.

[4]First presented by [10]. We do not include all three items presented there.
[5]Which we do not include in our system model as elaborated earlier

To handle the dynamic nature of the processors that can be a part of our system, we propose a property *Group Awareness* as:

**Group Awareness –** If a processor that is not a member of a group is present in any of the coverage area[6] of a group, it can eventually obtain information about the group.

## *3.5* SPECIFYING A PROXIMITY GROUP DELIVERY SERVICE

In this section we provide a specification for a delivery service that accompanies the membership service in PBGC. The interaction between membership and delivery service is two-way. The delivery service provides the communication primitives for maintaining consistent membership views across every participant, while the membership service provides the list of participants to whom messages are propagated.

**Delivery Integrity** – For every **recv** event, there is a corresponding **send** event. Formally —

$$t_i = \mathbf{recv}(p, m) \Rightarrow \exists q \, \exists j \, (j < i \wedge t_j = \mathbf{send}(q, m))$$

**No Duplication –** Two different **recv** events with the same content cannot occur at the same process.

$$t_i = \mathbf{recv}(p, m) \wedge t_j = \mathbf{recv}(p, m) \Rightarrow i = j$$

The above two properties together eliminate spurious **recv** events. Limiting **recv** events to only those that have been originated by some processor and have not already been delivered.

**Self Delivery –** Process $p$ delivers every message it sent in any view, unless it crashes after sending it. Formally —

$$t_i = \mathbf{send}(p, m) \wedge j > i \, t_j = \mathbf{crash}(p) \Rightarrow receives(p, m)$$

### *3.5.1 VIRTUAL SYNCHRONY*

In a view-oriented group communication service, **send** and **recv** events occur in the context of some view. Some applications may require that a message be delivered in the same view as the view in which it was sent. Other applications may require a message be delivered in the same view for all the members. In this section we present our approach to providing guarantees that address such delivery issues. Our approach utilises the partition anticipator to help an application make meaningful progress without running into some of the problems posed by other traditional services, as we describe below.

Some group communication service specifications require that all members deliver a message in the same view as the one in which it was sent. Such a requirement is called *Sending View Delivery*. A problem with such a requirement is that applications need to block while sending messages while a new view is being installed. In [14] it is proved that without requiring the applications to block the group communication service would violate properties of *Virtual Synchrony* and *Self Delivery*.

[6]That is to say this node is *reachable* from any of the present group members

A weaker alternative to *Sending View Delivery* is suggested as *Same View Delivery*. In *Same View Delivery* all members deliver a message in the same view, which may or may not be the view the message was sent in. The property suffices for applications that do not care which view the message was sent in, but are satisfied by the knowledge that all members receive the message in the same view.

Using S*ame View Delivery*, applications do not need to block while the view changes; the service guarantees messages sent during a change of view are delivered in either of the two views (the original or the changed view). All group communication services satisfy *Same View Delivery* for messages sent/received in the context of some view. Thus, this property is weaker than *Sending View Delivery*.

As an alternative to the above two properties, [14] suggests *Weak Virtual Synchrony*, as a model stronger than *Same View Delivery* yet weaker than *Sending View Delivery*. It is weak enough for applications to not block while a view is changing, yet strong enough to provide some notion of support for "view-aware" applications.

*Weak Virtual Synchrony* (WVS) requires each view change to be preceded by a "suggested view", a view the system believes has the highest probability of being the next view. A further requirement placed on every message sent in the suggested view (which is a superset of the current view) is that they should be delivered in the next view installed by the members. Thus "view-aware" applications know each message they receive was either sent in the current view, or the preceding suggested view.

Even though the WVS model does not require applications to block sending messages, it does not allow new processors to join the next regular group once a suggested view has been installed. If a new processor wanted to join the group after a suggested view has been installed, the service would initiate a new view install right after the next regular view has been installed. This causes the system to never stabilise in the face of frequent changes, as predicted in our system model. Another limitation of the model is that it implicitly works for applications that are satisfied to know of a superset of the existing view.

Yet another model was recently proposed by [15] in 2000. The model, called *Optimistic Virtual Synchrony* requires each view installation to be preceded by an optimistic view event, which provides the application with a "guess" about what the next view would be.

After this event, applications can optimistically send messages assuming they will be delivered in a view identical to the optimistic view. If the next view that is installed is not identical to the optimistic view, the application could either choose to use the messages received or roll back the optimistic message.

**Optimistic Virtual Synchrony** — we propose using an enhanced version of the *Optimistic Virtual Synchrony* model in our system. The enhanced model takes an "educated guess" on the next view, based on input from the Partition Anticipator. This model retains the benefits of providing a non-blocking view-based group communication service, but uses a mechanism to provide a meaningful prediction for the next view.

When the partition anticipator generates an **anticipated_chng**$(V)$ event, our service delivers an **optimistic view**$(V)$ event to the application. The application may choose to use this information and send messages in an optimistic view. If the application chooses to do so, it may or may not choose to "roll-back" if the next view installation is different from the optimistic view. Such support for delivering messages in the context of a view allows applications to have the flexibility on how to deal with changing views. We believe using such a model should also allow smooth handling of the frequent topology changes in a mobile ad-hoc network.

The above properties do not yet allow applications to reason about when groups become partitioned. To elaborate, we repeat a classic example from [6, 10]. Consider the case where three nodes $p$, $q$ and $r$, are in a common view called $v_1$, and two events occur. In the first event $r$ crashes while subsequently $p$ and $q$ become temporarily disconnected.

Let us consider the scenario where $p$ reacts to both the events and installs a view $v_2$ with only itself as a member. Process $q$ on the other hand reacts only to $r$'s failure and installs a view $v_3$ with $p$ and $q$ as members.

Now assume $p$ and $q$ share the same state at $v_1$ but $p$ undergoes changes while disconnected from $q$. Following this $q$ and $p$ reconnect and then install a view $v_3$ which has both $p$ and $q$ as members.

In such a situation, due to $p$'s state changes while it was disconnected from $q$, this would result in inconsistent behaviour on the part of $q$ (given only the above properties are being satisfied). Such situations require an additional property to handle merges of partitions, and thus ensure reasonable behaviour from our system. This scenario can be generalised to any trace where overlapping partitions views merge. The following properties eliminate traces that allow such inconsistencies.

**Agreement on Successors –** [7] If a process $p$ installs view $V$ in view $V'$, and if some process $q$ also installs $V$ and $q$ is a member of $V'$ then $q$ also installs $V$ in $V'$.

$$installs\_in(p,V,V') \wedge installs(q,V) \wedge q \in V'.members$$
$$\Rightarrow installs\_in(q,V,V')$$

*Agreement on Successors* guarantees that all processors from $p$'s current view and $p$'s previous view are coming from the same view and thus the Virtual Synchrony hypothesis holds. The property also invalidates traces where two overlapping views are allowed to merge, as brought out above.

[7] Extended Virtual Synchrony and Transitional Sets provide an alternative property that can help determine if the proposition of Virtual Synchrony holds or a state transfer is required.

### *3.5.2 ORDERING PROPERTIES*

The next important issue to consider for completing the specifications for the group communication service is message ordering. In the first case we support the *Weak Total Order* property for messages exchanged within a single group.

Before we define *Weak Total Order*, we repeat the notion of a Timestamp (TS) function as defined in [6]. A timestamp (TS) function is a one-to-one function from $M$ to the set of natural numbers:

$$TS\_function(f) \Rightarrow f : M \to N \wedge f(m) = f(m') \Rightarrow m = m'$$

**Weak Total Order –** Can be stated in two parts

1. For every pair of views $V$ and $V'$ there is a timestamp function $f$ so that every process that installs $V$ in $V'$ receives messages in $V'$ in an order consistent with $f$.

   $$\forall V\, \forall V' \quad \exists f\, (TS\_function(f) \wedge$$
   $$\forall p\, \forall m\, \forall m'\, (installs\_in(p, V, V') \wedge recv\_before\_in(p, m, m', V') \Rightarrow$$
   $$f(m) < f(m')).$$

2. For every view $V$ there is a timestamp function $f$ so that every process that has $V$ as its last view receives messages in $V$ in an order consistent with $f$. Formally —

   $$\forall V\, \exists f\, (TS\_function(f) \wedge \forall p \forall m\, \forall m' (last\_view(p, V) \wedge$$
   $$recv\_before\_in(p, m, m', V) \Rightarrow f(m) < f(m')))$$

By supporting the *Weak Total Order* property we imply that all messages will be *Causally Ordered* and the *Reliable Causal* property will apply.

**Reliable Causal –** If message $m$ causally precedes a message $m'$, and both are sent in the same view, then any process $q$ that receives $m'$ receives $m$ before $m'$. Formally —

$$t_i = \mathbf{send}(p, m) \wedge t_j = \mathbf{send}(p', m') \wedge$$
$$t_i \to t_j \wedge viewof(t_i)) = viewof(t_j) \wedge$$
$$\mathbf{recv}(q, m') \Rightarrow recv\_before(q, m, m')$$

*Weak Total Order* allows processors to disagree on the order of messages in case they disconnect from each other. But at the same *Weak Total Order* requires processors that remain connected with each other to receive messages in the same order.

When a number of processes belong to two or more of the same groups, we say that the groups overlap. In this case, there is a issue relating to the ordering of messages to be delivered at all processes that are members of more than one overlapping groups. *Atomic Multicast* specifies that all members of multiple groups deliver messages to all the groups in the same order. Implementing *Atomic Multicast* is expensive and is not directly crucial for the situations group communications for mobile ad-hoc networks will be used in. Thus we do *not* require our service to support *Atomic Multicast*; but at the same time depending on the implementation algorithms we might be able to support the same.

# 4 IMPLEMENTING HEARSAY WITH LATTE

One of the core conceptual tools defined in Gloss is "Hearsay" [24]. Hearsay is a form of communication where a message is discovered in the environment. A hearsay tool matches the profile and context of the receiver with the profile and context of the message to be delivered. This section describes the design and implementation of a hearsay system that uses email as the base infrastructure. The system, called LATTE (Location And Time Triggered Email), adds location and time to the standard email model as the hearsay contextual elements for sending and receiving email. LATTE's architecture provides a natural fit for demonstrating the work on proximity-based group communication.

LATTE is based on the idea that for the current email user, especially the mobile one, a significant proportion of email received is not relevant for a variety of reasons. Conversely, mobile users often do not receive messages that may be of interest to them because these messages relate to the location the users are currently visiting. Adding a level of context awareness to messaging services is likely to ameliorate these problems, and significantly enhance the relevancy of a user's email. Requirements for such a service include offering relevant information to recipients under specific spatio-temporal constraints. It should be possible to augment messages with a variety of contexts (for example, any combination of location, time and identity) and deliver them when their contextual conditions are satisfied. LATTE is therefore based on the email paradigm, and extends it to include dynamic consideration of location and time to determine the appropriate recipients for a message.

## *4.1* EMAIL-BASED HEARSAY SERVICE

As an initial step, we generalised the hearsay usage scenarios from the Gloss's "Bob goes to Paris". These can generally be categorised as:

1. **Alerts:** In this category, messages are attached to a location to indicate some warning. For example, well-wishing individuals or companies who undertake traffic monitoring as a commercial service may provide a traffic alert. Another example is a *periodic* form of alert message that is delivered to users in particular locations – for example, "this museum closes in fifteen minutes" is a message that should be delivered to people in the museum at the appropriate time every day the museum is open.
2. **Digital signposts:** This category provides similar functionality as the "digital graffiti" model [17; 18; 19] where a message is sent to a specific location with any user in the proximity receiving it. There is likely to be no information of any secrecy in this kind of message.
3. **Intended Delivery:** In this category, recipients are identified by some combination of identity, location and some timing constraints.

In general, context aware messaging can help reduce the amount of irrelevant information delivered to email users by using different kinds of context as a filter. In addition, it can help ensure that email users receive information relevant to them because of a change to their context. From scenarios within the categories listed above, we identified *three* context attributes that should be used as such a filter: *location*, *time* and *identity*. The goal of the context-aware email system is to deliver contextually valid messages to the right entities at the appropriate time. Where any combination of the three context attributes have been specified, validity is a function of testing the actual context of recipients against the one defined by the sender of the message. It is not necessary to specify values for all of the context attributes – as long as there is a recipient specified (either a named email user/group or a location), the other contextual attributes take sensible default values that try to emulate the familiar email behaviour as much as possible. We discuss each of the context attributes in the following sections.

### *4.1.1 LOCATION*

Location is defined as the spatial region a recipient must occupy in order to receive a message. Location is modelled as a series of discrete hierarchical regions with defined boundaries. Each region may be subdivided into smaller ones, which are arranged in a hierarchical manner. This approach was adopted from the Intentional Naming System (INS) [26] for its expressiveness and simplicity. Each level of the hierarchical tree has a logical name. In essence, the whole tree represents a mapping between logical names and geographical coordinates. There are many approaches to determining geographical coordinates, both indoor and outdoor – for example GPS or RADAR [20]. Our context-aware email service is designed in an open manner to support input relating to coordinates from any source. In the current version, we have implemented GPS and an indoor version using infrared transceivers is also being designed.

Another important feature in LATTE is the ability to support overlapping regions by allowing child nodes in the location hierarchy to have multiple parents. Essentially, location in this model is represented more as graph rather than a tree as in Figure 5. This feature allows multiple administrative domains to share physical locations and makes possible a more *decentralised delivery model* where LATTE servers are responsible for different (possible overlapping) geographical areas. Sending a LATTE message then, requires the knowledge of the individual server that is responsible for the wider geographical area the message is destined for. Directory services can be built to aid individual users composing LATTE messages but this is outside the scope of the current work. The final LATTE component that enables this type of distribution is a basic service discovery protocol that was built to enable automatic registration when LATTE clients are found in the vicinity of LATTE servers.

From the perspective of the email sender, location may be either explicitly specified, or left "anonymous". Where the location is specified, emails (without any further filtering based on identity/time) are valid to all users in that location. Messages can therefore be left at a location without requiring the sender to physically be there to tag it with a message. Where the location is anonymous, location is not considered in the filtering process for the email message.

As derived from the Intentional Naming System model, a location is named in an abstract, hierarchical plaintext format with the following syntax:

```
[location=value [division=value [subdivision=value]]..]
```

This gives a partial tree resembling Figure 5. For example, the front square in Trinity College might be described by

```
[Country=ie [City=Dublin [Area=TrinityCollege [Place=FrontSquare]]]]
```

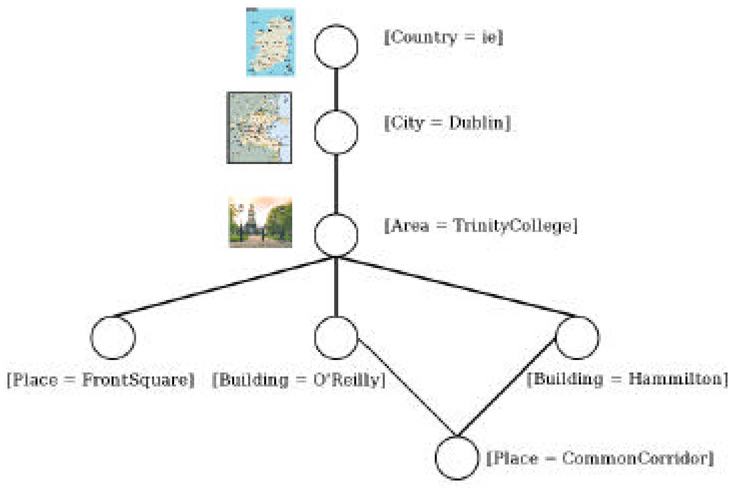


**Figure 5: Location Representation in LATTE**

### *4.1.2 TIME*

Time is also an important property to consider when evaluating whether to deliver particular messages. In LATTE, time can be *bounded* or *unbounded* and *periodic or non-periodic*. If a message is time bounded, then delivery of that message is valid for a certain duration. For example, a sender might want to send a meeting notice reminder to relevant participants any time from one hour before to five minutes before the meeting starts. If a message is not time bounded, then time is not a factor that is considered when filtering email for delivery. Time-bounded messages may also be tagged as periodic or non-periodic. Periodic messages are delivered to appropriate recipients at the stated period – for example, every day at 16:45, send a "museum closes in 15 minutes" message. Time-bounded messages that are non-periodic are valid only within the explicitly specified time bounds.

### *4.1.3 IDENTITY*

The main difference between the notion of identity from classical email systems and identity in the context-aware extensions relates to the issue of when it is decided that a message should be delivered to a particular recipient. In classical email, the identity of the recipient is known at composition. This can be considered *early* binding. Adding contextual properties to a message, however, means that the message may not have its destination identity explicitly specified before transmission. Instead, there is a set of contextual attributes that can be matched against potentially multiple recipients at delivery time. This can therefore be termed *late* binding. Late binding improves flexibility and makes it possible to realise the categories of scenarios based on alerts, digital signposts and intended delivery as described in Section 4.1.

In addition, we further classify the contextual values of identity as individual, group and anonymous. An *individual* identity is defined when the recipient's identity is known at composition and is therefore an early-binding mechanism similar to email. *Groups* consist of one or more identities that may not be known at email composition time. This group model is similar to the classical mailing-list model with one significant difference. In addition to membership being defined similarly to mailing lists, further filtering occurs based on the location and time of the intended message. Therefore, group membership is dynamic based on context. Lastly, when the identity is *anonymous*, any potential recipients who conform to the remaining contextual attributes of location and time will receive the message.

## *4.2* IMPLEMENTATION

LATTE is implemented in Java on the client side for the GUI, and for the dissemination of context information to the context server. Java gives good portability especially when considering platforms such as PDAs and laptop computers. Still, the client side has no dependency on Java in particular and can support any language that provides an interface to the SMTP. On the server side a MySQL database was used to store messages and user profiles and interface with SMTP component.

In this section we describe the overall architecture of LATTE, and also how LATTE messages flow from the sender, through the LATTE server to the delivery to recipient(s).

### *4.2.1 OVERALL ARCHITECTURE*

The architecture to support the flow of a LATTE message is depicted in Figure 6. In the current implementation, every LATTE email is sent via SMTP to a special account (e.g. latte@domain.org) at some mail server (see section “4.2.2 Sender Model”). There, it is processed and stored in the message database for later retrieval (see section “4.2.3 Context Engine”). After their initial registration with the Context Engine, receivers periodically transmit their dynamic context, (i.e., location and group membership) by using simple beacons. Recipients receive emails in the standard way by running a POP or an IMAP client.

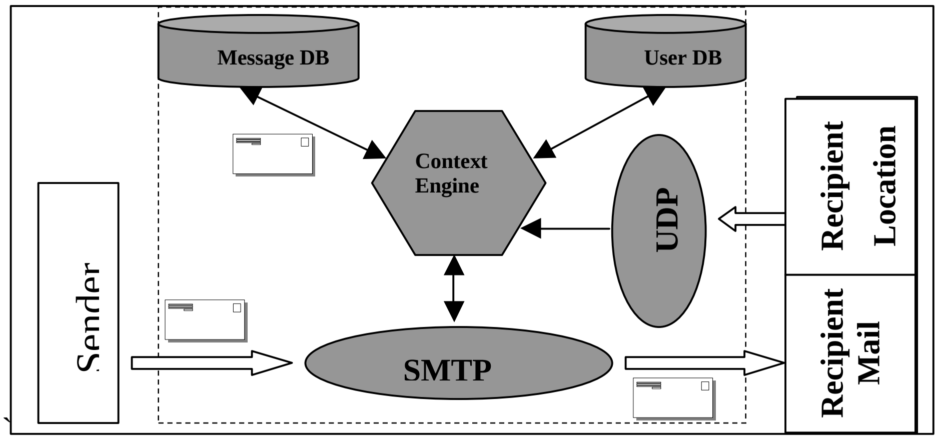


**Figure 6: LATTE Architecture**

### *4.2.2 SENDER MODEL*

Message transmission is achieved through the use of the SMTP protocol [16]. SMTP was chosen for a number of reasons. It provides a familiar interface to users and is interoperable with existing clients. Furthermore, SMTP allows the transmission of messages (emails) that can contain an extended set of headers [21]. These headers are marked as “X-headers”, and are designed for clients to transmit user-defined information. LATTE uses these X-headers to capture the required contextual extensions to email, calling them X-LATTE-* headers. The following headers have been defined to encapsulate the required message context from the sender:

- `X-LATTE-Identity` - Specifies the LATTE extensions to identity.
- `X-LATTE-Location` – Specifies the logical location in the INS format explained in section 4.1.2.1
- `X-LATTE-Time` – Specifies the time the system should deliver the message.
- `X-LATTE-Duration` – Specifies the time for which the message is valid – new users moving into a valid contextual state in this duration may be considered as a recipient.
- `X-LATTE-Offset` – Allows for a message to be repeated at regular intervals.

LATTE clients may, of course, send LATTE emails using their usual desktop/laptop. However, given the significance of mobility for the LATTE system, we have also implemented a version for the client that uses a PDA. As illustrated in Figure 6, the interface is designed to look and feel like a standard email client, with the additional contextual information added in a similar manner to named recipients, etc. in standard email.

### *4.2.3 CONTEXT ENGINE*

Context evaluation refers to the processing of recipient context against a set of evaluation rules. Resolving location is the most challenging task if it is to be done efficiently and not overload the server. The location model described in section 4.1.2.1 is encoded in the server as a binary tree. Each node in the tree represents a rectangular area that has a set of four coordinate points. As a message arrives containing a logical description of a place, this textual description gets converted into GPS coordinates, which are matched against the nodes in the binary tree. The matching node will then hold a reference to the message's sequence id.

Part of the initial handshake procedure when clients register with a LATTE server, is the automatic transfer to the client of a location file that contains the area the server is responsible for. This file is essentially a mapping between the logical INS format and GPS coordinates. Thus, the processing of the location coordinates is partly done on the client device greatly enhancing flexibility and performance of the overall system by reducing the load on the server side.

When recipients transmit their location, the tree is traversed and any messages found in the parent and child nodes get tagged as "location ready". Then time and identity of the messages are evaluated and if they match those of the client, they are made available for displaying.

### *4.2.4 EXTENDING LATTE WITH PBGC*

As was described above, LATTE requires a mechanism for announcing the presence of LATTE servers to clients inside their communication range and their geographical area of responsibility. Essentially, this can be translated to a requirement for an event service that subscribes LATTE clients to a LATTE server as soon as they appear inside the server's area of responsibility. Currently this is implemented as a simple beaconing service that has no knowledge of location and only extends to immediate neighbours. A proper event based service would notify only clients inside a designated area and would extend to multiple hops. Such an underlying framework would have the extra benefit of providing LATTE servers with a proper notification mechanism so LATTE could be extended to easily handle group notification as another mode of interaction alternative to email.

The requirement for such an event service is a perfect candidate for having proximity groups as the underlying communication medium. In [22] it is being recognised that group communication provide a natural mean to support event-based communication models. Communication groups provide a one to many communication pattern that can be used by event producers to propagate events to consumers. Although other approaches exist, such as remote method invocation (CORBA, RMI), group communication has been identified as the most suitable approach [22, 23].

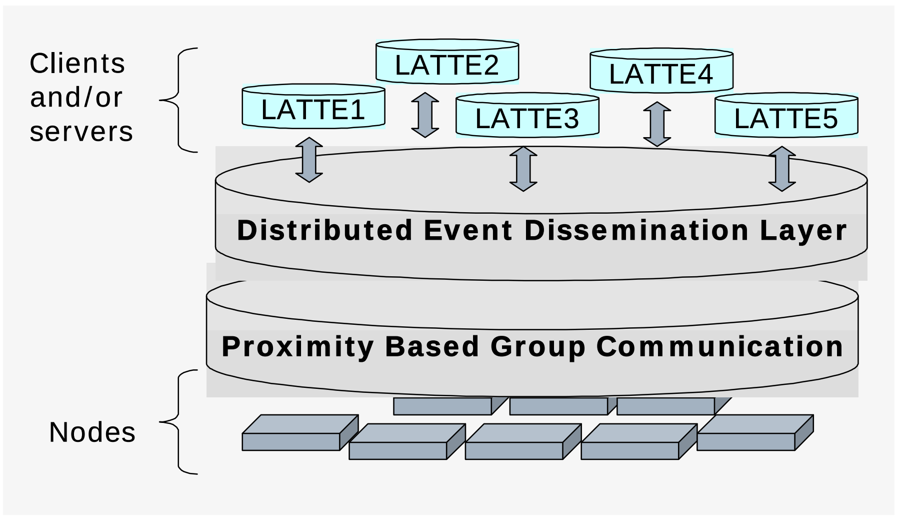

Clients and/or servers
LATTE2
LATTE4
LATTE1
LATTE3
LATTE5
Distributed Event Dissemination Layer
Proximity Based Group Communication
Nodes

**Figure 7: Extended LATTE Architecture**

As illustrated in Figure 7, the next incarnation of LATTE will provide an underlying communication medium based on proximity groups with an event-based notification framework on top of that. As such, the range of LATTE servers is expanded due to the multi-hop nature of proximity groups while the notification framework opens up possibilities for other modes of interaction. The envisioned transport mechanism will then consist of the layered architecture illustrated.

# 5 CONCLUSION

Proximity-based group communication (PBGC) supports reliable, consistent communication of information between nodes that are in proximity to each other, in a mobile, ad hoc environment. Within the Gloss middleware infrastructure, this service covers communications within what we consider to be the “edge” network – the network the supports local, mobile, interactions.

This deliverable describes a formal specification for PBGC that takes advantage of existing formal specifications for group communications systems, but extends them to include dynamic membership due to mobility, membership based on geographic proximity, and partition anticipation. These additional properties have had a significant impact on both the group membership and the delivery specifications of a communication service.

In addition, this deliverable describes an implementation of a Hearsay service called LATTE, that uses the email model to provide contextual message delivery based on location and time. As an initial proof of concept, this prototype was implemented on existing technologies. As an evaluator for the PBGC service, however, the architecture of LATTE will exploit reliable message delivery within the required proximity.

# 6 REFERENCES

[1] K.Singh, S.Clarke and A.Nedos. Proximity Groups for Mobile Ad Hoc Networks, GloSS Deliverable 13, 2002.

[2] Starlab and UJF. Gloss scenario: Bob goes to Paris, April 2002.

[3] R. van Renesse and S. Maffeis and K. Birman. Horus: A Flexible Group Communications System, *Communications of the ACM*, 39(4):76-83, April 1996.

[4] D. Dolev and D. Malki. The Design of the Transis System, Dagstuhl Workshop on Unifying Theory and Practice in Distributed Computing, September 1995.

[5] Ö. Babaouglu and R. Davoli and A. Montresor and R. Segala. System Support for Partition-Aware Network Applications. In *Proceedings of the 18th International Conference on Distributed Computing Systems*, volume 18, pages 184-191. IEEE, May 1998.

[6] G. V. Chockler and I. Keidar and R. Vitenberg. Group Communication Specifications: A Comprehensive Study. *ACM Computing Surveys(CSUR)*, 33(4):427-469,2001.

[7] G.Ricart and A.K.Agrawala. An optimal algorithm for mutual exclusion in computer networks. Comm. ACM, 24(1):9-17, 1981.

[8] T. D. Chandra and V. Hadzilacos and S. Toueg. On The Impossibility of Group Membership, In *PODC,* pages 322-330, 1996.

[9] L. E. Moser and P.M. Melliar-Smith and V. Agarwal. Processor Membership in Asynchronous Distributed Systems, *IEEE Transaction on Parallel and Distributed Systems*, 5(5), May 1994.

[10] Ö. Babaouglu and R. Davoli and A. Montresor. Group Membership in Partitionable Systems: Specification and Algorithms, *IEEE Transactions on Software Engineering*, 27(4), April 2001.

[11] T. D. Chandra and S. Toueg. Unreliable Failure Detectors for Reliable Distributed Systems, *Journal of the ACM*, 43(2):225-267, March 1996.

[12] D. Dolev and S. Kramer and D. Malki. Total Ordering of Messages in Broadcast Domains, Technical Report CS92-9, Computer Science Department, The Hebrew University of Jerusalem, 1992.

[13] L. L. Peterson, N.C. Buchholz and D. Schlichting. Preserving and Using Context Information in Interprocess Communications, *Transactions on Computer Systems*, 7(3):217-246, August 1989.

[14] R. Friedman and R. van Renesse. Strong and Weak Virtual Synchrony in Horus, Technical Report TR 95-1537, Department of Computer Science, Cornell University, 1995.

[15] J. Sussman, I. Keidar and K. Marzullo. Optimistic Virtual Synchrony, 19th IEEE Symposium on Reliable Distributed Systems, pages 42-51, IEEE, 2000.

[16] J. Postel. RFC 821: Simple mail transfer protocol. Technical report, DDN Network Information Center, August 1982.

[17] Sandin, A, et al. (2001) GeoNotes: Social and Navigational Aspects of Location-Based Information Systems, *Ubicomp 2001*.

[18] Tarumi, H., Morishita, K., Nakao, M., Kambayashi, Y. (1999) SpaceTag: An Overlaid Virtual System and its Applications, *IEEE International Conference on Multimedia Computing and Systems Volume I-Volume*.

[19] W. G. Griswold, R. Boyer, S. W. Brown, T. M. Truong, E. Bhasker, G. R. Jay, R. B. Shapiro ActiveCampus - Sustaining Educational Communities through Mobile Technology.

[20] Bahl, P., and Padmanabhan, V. (200) RADAR: An in-building RF-based user location and tracking system. In *Proceedings of IEEE INFOCOM*, volume 2, pages 775–784, March 2000.

[21] D. H. Crocker. Standard for the Format of ARPA Internet Text Messages. Request for Comments 822, DDN Network Information Center, SRI International, August 1982.

[22] G. Banavar, T. Chandra, R. Strom and D. Sturman. A Case for Message Oriented Middleware, 13th International Symposium on DIStributed Computing (DISC'99), Bratislava, Slovak Republic, 1999.

[23] R. Meier and V. Cahill. STEAM: Event-Based Middleware for Wireless Ad Hoc Networks, 22nd International Conference on Distributed Computing Systems Workshops (ICDCSW'02), July 2002.

[24] P. Welen, A. Wilson, P. Nixon. Design Guidelines for Integrated spaces. GlosSS Deliverable D.5, 2002.

[25] L. Briesemeister and G. Hommel. Localized group membership service for ad hoc networks. In *Proceedings of International Workshop on Ad Hoc Networking (IWAHN)*, pages 94-100, August 2002.

[26] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. *Proceedings of the seventeenth ACM symposium on Operating systems principles*, pages 186–201. ACM Press, 1999.