SELMA: A middleware platform for self-organizing distributed applications in mobile multihop ad-hoc networks[*]

Daniel Görgen Hannes Frey Johannes K. Lehnert Peter Sturm

Abstract

This work presents a middleware architecture for distributed applications communicating solely over a mobile multihop ad-hoc network. The design of the platform is based on the marketplace pattern where communication among negotiating components of an application concentrates on bounded geographical areas, called marketplaces. The marketplace pattern allows to develop resource saving and self-organizing applications needing no additional administrative overhead and coping well with permanent network partitions.

1 Introduction

Supporting mobile devices is central to any next generation networking technology. Future paradigms such as pervasive computing and ubiquitous computing [1] are characterized by countless numbers of small and thus mobile devices that communicate with neighboring peers using wireless transmission techniques. Several of these mobile devices may form highly dynamic ad-hoc networks at any given time. These networks can be classified whether they are used alone or used to reach a conventional wireful network, and, furthermore, if they are using single hops to communicate directly or require multiple hops within the network to communicate with another device. Singlehop networks are state-of-the-art, e.g. in the WLAN infrastructure mode by means of access points (possibly by crossing the transmission ranges of access points using some handover mechanism), in the basic WLAN ad-hoc mode, and within elementary bluetooth cells. By contrast, from a research as well as an engineering perspective, modelling and running multihop ad-hoc networks successfully is still a challenging task.

[*]This work is funded in part by DFG, Schwerpunktprogramm SPP1140 "Basissoftware für selbstorganisierende Infrastrukturen für vernetzte mobile Systeme" and Microsoft Research Embedded Systems IFP (Contract 2003-210)

These multihop ad-hoc networks can be further differentiated depending on the average mobility of all participating devices. Mobile devices with a given inertia allow the assembly and the ongoing maintenance of a routing infrastructure. Within such a "sluggish" network, new but still familiar protocols such as mobile IP can be deployed successfully to support a wide range of traditional communication needs.

Spontaneous multihop networks with a high device mobility and frequent fluctuation cannot be governed by routing strategies that enable transparent end-to-end communication as it is provided by conventional TCP/IP-based infrastructures for instance. Self-organization is required as the primary design principle to cope with the anticipated frequency of change. This leads to the requirement that any decision within a mobile device can only be based on local information, e.g. the state of the mobile device itself and its current neighborhood. In this environment, most of the goals of a distributed mobile application are achieved by synergy through altruistic behavior of all involved devices. For instance, position-based routing is able to fulfill this requirement and even to cope with sporadic network partitions.

This paper presents the prototype of SELMA (Self-organized Marketplace-based Middleware for Mobile Ad-hoc Networks), a middleware platform for this type of spontaneous multihop ad-hoc networks. Most of the services of this middleware are centered around a specific communication pattern resembling traditional marketplaces. By using this pattern, mobile applications forward data or agents to well-defined geographical locations (marketplaces), where the probability to find required information increases drastically. Sending data and agents back and forth between interested devices and marketplaces is achieved using variations of geographic routing and epidemic message distribution algorithms. There are various of applications like electronic blackboards, university rideboards, public information services and online auctions for instance which can use ad-hoc communication based on the marketplace-pattern solely. Security and privacy are not investigated at the moment but will be concerned after the concept of marketplace-based communication pattern and services provided by the platform are evaluated sufficiently.

In the next section, the marketplace pattern, which is central to the middleware platform, is presented by discussing an online auction system for multihop ad-hoc networks. Section 3 presents the main components of the middleware architecture. Fianlly, related work is discussed in section 4 and section 5 concludes the paper.

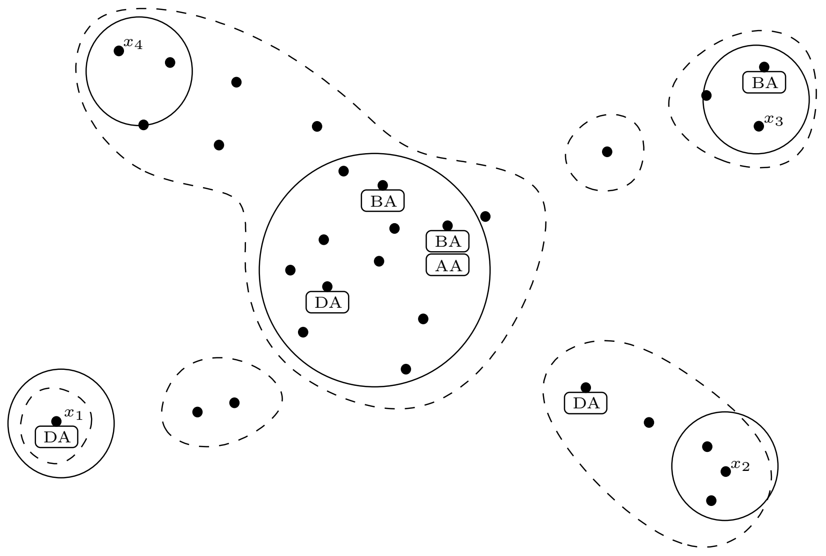

Figure 1: Example scenario for UbiBay

2 The marketplace pattern

The marketplace-based communication pattern is essential to the middleware design presented in section 3. This section describes the basic idea of marketplaces by means of an example application called UbiBay. The goal of UbiBay is to realize an online auction system based on mobile devices with wireless communication capabilities. Communication is achieved solely over the ad-hoc network formed by these devices.

Running an online auction in a large scale ad-hoc network is a challenging task, since the network inherently tends to be permanently partitioned. Additionally, communication overhead has to be reduced, due to the fact that energy of low powered small devices forming the ad-hoc network has to be conserved. In order to cope with these problems and to increase the probability that auctioneers and bidders can find each other, UbiBay uses the marketplace communication pattern. Auctions are restricted to certain defined geographical areas, the UbiBay marketplaces. Figure 1 gives an example scenario for UbiBay. The circle in the middle is an UbiBay marketplace. The areas with dashed borders are the current network partitions. In order to assure a more reliable communication among auctioneers and bidders, the marketplace is located in a geographical region with a high device population.

UbiBay users are not required to be at the marketplace physically, but are allowed to utilize other ones mobile devices to let a software agent participate in an auction. An agent acting on behalf of its originator is allowed to change its hosting device, in order to reach its corresponding marketplace. After an agent has finished negotiating with other agents at the marketplace, it has to inform its originator about the negotiation result (e.g. the agent has won the auction). In order to allow agents to find their originator, each agent knows the originator's homezone. A homezone is a defined geographical area, with a high sojourn probability for the originator (e.g. an area around his apartment). Thus, an agent willing to inform its originator has to move back to its homezone and wait there until the originator's device enters the homezone. The four small circles in figure 1 depict the homezones for user $x_1, \ldots, x_4$ respectively.

UbiBay distinguishes three agent classes, auction agents (AA), bid agents (BA) and discovery agents (DA). Auction agents are used as auctioneers for one designated UbiBay auction. Once an auction agent has reached its marketplace, the auction is running for a given time period. A user willing to participate in an UbiBay auction sends a discovery agent to the designated marketplace. The discovery agent asks all auction agents for information about their designated auction. Subsequently, the agent moves back to its originators homezone to report all collected information about currently running auctions. In order to participate in a certain auction, a bid agent has to be sent to the corresponding marketplace. On arrival at the marketplace the agent will successively send its bids to the auction agent until its maximum bid is exceeded or no more agents are sending a higher bid. An agent loosing an auction will return immediately to its homezone to inform the user. After the time period for an auction is exceeded, the auctioneer and the winner of the auction are also moving back to their originators.

3 The middleware architecture

The proposed middleware architecture (see figure 2) is divided in three parts, communication abstraction, agent platform and the set of application and service agents. The following subsections present the components of each layer in more detail.

3.1 Communication abstraction

The communication abstraction provides generic methods for positioning, wireless communication, and device discovery. Thus, applications based on this middleware platform are portable across different communication hardware and positioning systems.

Positioning

Many functions in the higher layers of the middleware need information about the actual position of the device and its neighbors. The positioning layer models the world

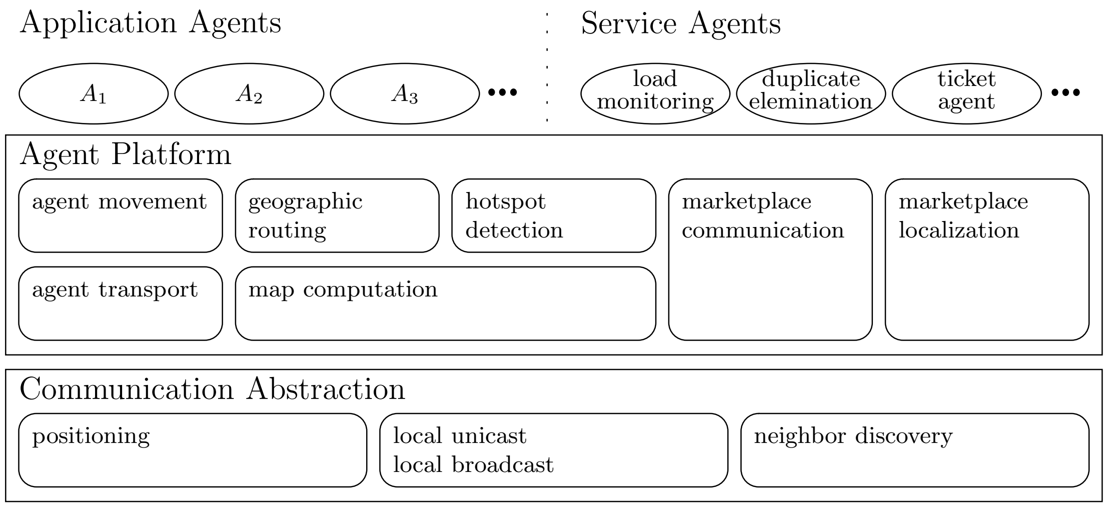

Figure 2: The middleware architecture

with cartesian coordinates. Since there are positioning systems [2] like GPS representing positions with longitude, latitude and altitude, a transformation component has to map native positions from the positioning hardware to cartesian coordinates. Possible inaccuracies of existing positioning systems are handled in this transformation component as well, e.g. if GPS cannot provide a position the previous position or a position calculated from the positions of the neighbors (e.g. by triangulation) may be used. In the actual prototype implementation, positioning is achieved by GPS and by using a Gauss-Krüger projection to map GPS coordinates to cartesian ones.

Wireless communication

The communication model of the middleware is based on two basic wireless communication primitives: local unicast and local broadcast. Local unicast enables a device to send a single message to one device within its sending range, while by local broadcast a message is delivered to all one hop neighbors. In order to provide the upper layers with these two communication primitives, the middleware requires some kind of wireless communication technology. If necessary, the communication layer has to implement local unicast over a broadcast technology or vice versa. The current implementation of SELMA is based on the ad-hoc mode of IEEE 802.11 and local unicast and broadcast are based on UDP packets.

Neighbor discovery

In order to provide devices with knowledge about surrounding devices, the middleware includes a neighbor discovery service, which collects information about all one hop neighbors. The neighbor discovery service exchanges and collects any kind of information the middleware needs, e.g. the actual device position and trajectory, the number of agents on the device or the remaining device capacity to host additional agents. If the underlying communication protocol has a suitable device discovery protocol (e.g. Bluetooth), this protocol will be used by the neighbor discovery service. Otherwise, devices in the neighborhood will be discovered by using the neighbor discovery protocol provided by the middleware. This discovery protocol is based on periodic broadcasts. Each device keeps a neighbor information cache which is updated whenever a new neighbor discovery protocol message is received. When a device receives a discovery protocol message it uses a unicast discovery protocol message to respond. This allows the other device to recognize bidirectional links between the two devices. For the current implementation bidirectional links are necessary to provide reliable agent transport between any two devices as described below.

3.2 Agent Transport Protocol

Agents move to geographic targets by “jumping” from host to host. To ensure that no agent is lost during its transmission to other hosts agent duplication is tolerated. This is achieved as follows: the sender keeps a copy of the agent until the receiver commits the receipt of the agent. The sender deletes the agent and communicates this deletion to the receiver. This ensures that no duplication has occured. If one of the protocol messages is unresolveably lost, at least one agent exists at sender or at receiver side (or both) and continues to move to its target. These agents are always marked as possibly duplicated. Thus, the protocol achieves that an agent is always marked when a duplication occurs, while the opposite does not necessarily hold.

Due to agent duplication, different agreements at a marketplace regarding one agent are possible. This can be avoided, since all instances are tagged as possibly duplicated and all have the same destination marketplace. On arrival at the marketplace a duplicate elimination procedure can be started if the agent is tagged. For instance, a service agent responsible for agent duplicates (see section 3.10) is used.

Since devices provide only limited resources for agents, it is possible that an agent is refused. Such a refuse message causes the sender to keep the agent and try to send it to another device. To avoid agent refusing, neighbor discovery provides additional information about the remaining space for agents per device.

3.3 Agent movement

The agent movement service is used by agents to reach geographic regions such as marketplaces and homezones. The agent sets its current target and the movement service ensures that the agent is sent to it. When the target, e.g. marketplace, is reached the agent is informed.

The agent movement service moves the agent towards the current target by changing the host if possible. A suitable host to “jump” to is selected by the geographic routing service (see section 3.6). The agent is sent to the host by using the agent transport protocol. When the agent enters its predefined target region, this event is passed to the agent. The movement service ensures that the agent stays at its current target until a new target is passed to it. If an agent temporary leaves the target, it is informed about leaving and reentering.

In order to move back to its owner’s device the agent passes its homezone coordinates and the owner’s device id to the movement service. The movement service uses the homezone as the current target. Upon arrival at the homezone, the owner of the agent is searched by geographically limited broadcasts as described in section 3.7. When the owner’s device is reached, the agent is informed. In case the owner’s device is discovered by the neighbor discovery service while the movement service moves the agent towards the home zone, the agent is directly sent to it. In the current implementation agents can reach their originator’s device faster since the device leaves a trail of links to its actual position.

3.4 Map computation

In order to allow a more reliable communication among negotiating agents, a marketplace has to be in a geographical region with a high device population. Thus, devices need information about the current geographical distribution of all devices forming the ad-hoc network to enable useful decisions where to position a new marketplace.

The map computation service divides a geographically bounded area surrounding the device (e.g. 500m $\times$ 500m) in small rectangles of size 50m $\times$ 50m for instance. A distributed algorithm continuously approximates the actual number of devices inside each rectangle. To achieve this, each device permanently updates the number of all devices inside the rectangle where it is located itself. Periodically, it broadcasts its current map state to all neighbors. Conversely, it listens for all incoming map states and averages its current map version with the incoming ones.

3.5 Hotspot detection

By using the map computation service, the hotspot detection mechanism enables devices to decide where to position a new marketplace. When a new marketplace is initiated, the service checks for a map rectangle with a high number of mobile devices. Additionally, the initiator is allowed to constrain the hotspot detection to put new marketplaces only in his proximity. The agent used to start the new marketplace is then sent towards the destination rectangle. To avoid an accumulation of marketplaces in one rectangle, the actual number of marketplaces in a specific rectangle can be considered when creating a new marketplace. This value is also be disseminated by the map computation service described above. When an agent arrives at the destination rectangle, it is allowed to start the new marketplace.

To be more flexible to dynamic changes regarding the device distribution, intermediate nodes are allowed to change the destination rectangle, if they know about a "better" rectangle to place the new marketplace. This, however, might lead to possible loops. To avoid this, an agent manages a set of rectangles it has passed by now. An agent is not allowed to start a marketplace on rectangles from this avoidance set.

3.6 Geographic routing

In general, position-based greedy routing algorithms [3] have to use a recovery mechanism to cope with the greedy routing failure when packets are stuck in a local minimum (i.e. no neighbor is closer to destination than the current forwarding device). In particular, in a large scale ad-hoc network this greedy routing failure might occur frequently since there is no uniform distribution of mobile devices (e.g. high device population in a public place and no devices on a lake).

By using the map from section 3.4, "holes" in the current ad-hoc network can be detected in advance and avoided during message transport. This keeps the possibility of greedy routing failures to a minimum.

Traditional greedy routing will drop packets and possibly send an error notification to the originator if sender and recipient are located in different network partitions. The application class which is intended to use the marketplace communication pattern has no hard time constraints on agents arrival at the marketplace. Thus, it is a good idea to use device mobility as an additional "slow" message transport mechanism, which allows agents to pass network partitions.

Each device moving inside a network partition might be a possible candidate for agents to leave this partition. The map service provides devices with information about other network partitions. This allows them to place agents on a moving device leaving the actual network partition in a certain direction.

3.7 Marketplace communication

At marketplaces, agents are able to communicate with each other. This communication generally needs only a few hops and is limited by the marketplace border. Device density at marketplaces is commonly higher as in other parts of the network. This leads to a low probability for unresolveable message losses. Two communication types are provided by the current prototype: a marketplace-wide broadcast and a unicast addressing a specific agent. All communication takes place between agents not between single devices, because agents may change their hosting device during communication.

Marketplace broadcasts are geographically limited, i.e. messages are discarded outside the marketplace. Thus, no network load is produced outside the marketplace. To reduce redundant messages generated by simple flooding, a neighbor knowledge broadcast [4] is used. This kind of broadcast needs only a part of the devices to rebroadcast the messages but still reaches all devices on the marketplace. Marketplace broadcasts are commonly used by agents to discover their yet unknown communication partners.

Marketplace unicasts use topology-based routing [5] between two communication partners. So, only a few message forwards are needed to reach each other. Routes can be built with marketplace-wide route discovery broadcasts, but more commonly, routes already exists due to the fact that agents need to discover their communication partner first.

3.8 Localization service

In order to enable agents to find their corresponding marketplace, information about current marketplace positions and applications running at them has to be disseminated over the complete ad-hoc network. In order to avoid redundant information exchange, each device uses the same hash function to compute a hash value from its currently stored marketplace information. This value is periodically broadcast to all one hop neighbors. Each incoming value is compared with the value stored on the receiving device. Only if a received value differs, information about marketplaces has to be exchanged and updated.

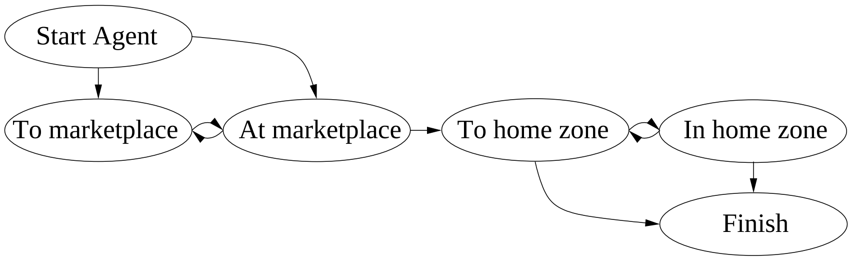

Figure 3: Application agents lifcycle

3.9 Application agents

Application agents generally have a lifecycle as depicted in figure 3. After being started by the user, an agent inquires the marketplace position of its application by using the marketplace localization service. The agent starts its movement service by adjusting it to the marketplace position. When the marketplace is reached the agent is informed by the movement service. Agents at marketplaces change to application specific states (e.g. bidding in the UbiBay example) and are able to communicate with others using the marketplace communication mechanism. Applications running at the marketplace have to take into account that agents can always temporarily be carried out of the marketplace. The movement service informs the agents about these events. After the marketplace specific parts of the application are finished, the agent starts moving to its homezone or to other targets e.g. other marketplaces. In order to detect and resolve a possible loss of the last application protocol message, an agent is required to wait a certain time on the marketplace after it has finished its negotiations.

After setting the homezone as the target for the movement service, the agent will be informed when the owner of the agent is reached. Now the agent is able to communicate its results, delete itself or go on with application specific tasks.

Agents are informed by the transport protocol when the movement of an agent results in a potential duplicate. This enables the agent to take precautions, if needed, when it reaches its current target. To cope with agent duplication the duplication elemination service agent (see section 3.10) can be used.

If necessary, an application is allowed to create a new marketplace (e.g. if all known marketplaces are too far away or no marketplace exists for this application). The application agent chooses a new marketplace location with the aid of the hotspot detection service. After reaching the selected target, the agent must propagate the creation of the new marketplace. This is achieved by the marketplace localization service. Additionally, all necessary service agents have to be created.

3.10 Service Agents

Agents not only represent users at distant locations. Additionally, agents can be used to realize a location based service. Such service agents are not assigned to any user, thus they must be created in a self-organizing manner. Every user agent detecting a missing service agent is able to create a new one. Since service agents are assigned to fixed locations (e.g. marketplaces) it is quite simple to detect a duplicated service agent caused by multiple creation or transmission failures.

Load balancing

A marketplace covers a fixed geographic region. Thus, it is possible that the network is saturated by marketplace application messages or, even more probable, agents are no longer able to enter the marketplace because of resource restrictions on each device forming the marketplace. This problem is solved by “splitting” the marketplace: create a new marketplace at a different geographic location, assign a subset of the agents to it and propagate the new marketplace.

This decision is made by a service agent, monitoring the load at a marketplace. When it detects an overload, a new marketplace must be opened and a new load balancing agent for this marketplace is created. The new agent searches a location for the new marketplace with the aid of the hotspot detection service. This new location should not be too far away because a subset of the agents from the original marketplace needs to migrate to the new one. Additionally, marketplace regions are not allowed to overlap. After the new load agent has reached the new marketplace it propagates its applications by using the marketplace localization service. The decision which applications with their according agents have to migrate to the new marketplace is made by the original load balancing agent. It is told to the new load balancing agent at creation time.

When a marketplace has been split, underload must also be considered. When underload is detected a marketplace must be closed. The applications at that marketplace must be assigned to another known marketplace and this assignment must be propagated. After this assignment is committed by the load balancing agent responsible for the other marketplace, the agent at the closed marketplace can delete itself. Communication between agents at different marketplaces is based on geographic routing.

Load balancing is not only possible by assigning different applications to marketplaces. Agents of one application are classifiable and can be assigned to different marketplaces. It is up to the application to define such a classification.

Duplicate elimination

Since the agent transport protocol allows agent duplication, precautions must be made at a marketplace to detect the other duplicates there. Duplicates must not be able to make different agreements. So, if an agent marked as possibly duplicated reaches the marketplace it must first ask a caretaker agent responsible for duplicates. If a duplicate is already registered there, it is up to the asking agent to delete itself. If not, it registers itself as the first one and is now able to make agreements with other agents. No other duplicate can do this from now on.

Billboard agent

Marketplaces can be used as public billboards. Information can be placed there for a given time period and can be inquired by agents. The persistence of the information cannot be guaranteed due to limited and changing resources at a marketplace. Each device at a marketplace provides resources to store public data. Information must be assigned to specific devices, reassigned when the device leaves the marketplace and be deleted. Deletion is needed when the time to live has expired or all available resources are fully used and other information has higher priority (e.g. is newer). This information organization is done by a service agent.

Ticket agent

This agent provides tickets unique for its marketplace and with a requested time to live. Application agents requesting a ticket must pass a ticket id unique for its application. It is guaranteed that the ticket agent provides only one ticket for this id. All following other requests for this id are denied. For example applications are able to elect one agent out of a group to represent the whole group.

4 Related work

In general, permanent network partitions are inherent to large scale ad-hoc networks, which prevents the provision of transparent end-to-end communication. Additionally, the bandwidth in an ad-hoc network decays with $\frac{1}{\sqrt{n}}$ where n is the number of nodes [6]. This shows the necessity of new communication paradigms like the marketplace-based approach[7] followed by the proposed middleware platform. Communication on marketplaces does not suffer from these problems, since marketplaces define small subsets of the entire ad-hoc network.

Traditional middleware systems such as CORBA[8], Microsoft (D)COM[9] or Java RMI[10] are not suitable for mobile ad-hoc networks because they rely on central infrastructure like naming services and assume the reachability of all network nodes. These assumptions cannot be matched by mobile multihop ad-hoc networks. Additionally, traditional middleware approaches are too heavyweight for mobile devices.

Many adaptions have been made to traditional middleware to apply them in mobile settings such as OpenCORBA[11] or NextGenerationMiddleware[12]. These extensions provide mechanisms for context awareness, but cover mainly infrastructure networks and one-hop mobile communications.

An increasing number of middleware systems is developed specifically for mobile ad-hoc networks. XMIDDLE[13] allows the sharing of XML documents between mobile nodes. Lime[14] and L^{2}imbo[15] are based on the idea of tuple-spaces [16], which they share between neighbored nodes. But due to the coupling of nodes, these approaches are not well-suited for highly mobile multihop ad-hoc networks. MESHMdl[17] employs the idea of tuple-spaces as well, but avoids coupling of nodes by using mobile agents, which communicate with each other using the local tuple-space of the agent platform. Proem[18] provides a peer-to-peer computing platform for mobile ad-hoc networks. STEAM[19] limits the delivery of events to geographic regions around the sender which is similar to the geographically bound communication at marketplaces. STEAM provides no long distance communication, it is only possible to receive events over a distance of a few hops.

Mobile agent frameworks exist in numerous variations, Aglets[20] or MARS[21] may serve as examples. These frameworks were designed for fixed networks and thus the above mentioned problems of traditional middleware approaches apply to them as well. The SWAT infrastructure[22] provides a secure platform for mobile agents in mobile ad-hoc networks. This infrastructure require a permanent link-based routing connection between all hosts and thus limits the ad-hoc network to a few hops and thus is not applicable to the marketplace-based communication pattern.

5 Summary

This article gives an overview of a middleware architecture intended to be used by distributed applications in mobile multihop ad-hoc networks. The design follows a communication pattern based on the marketplace metaphor. Marketplace-based communication needs no external administrative overhead, is resource saving and copes with permanent network partitions. Herewith, the concept of marketplaces is a promising candidate for practically realizing self-organizing distributed applications in large scale ad-hoc networks.

Parts of the presented middleware architecture have been implemented in a simulation environment. The applicability of this platform has been evaluated by means of example applications. The UbiBay application and its corresponding agents presented in section 2 have been implemented for the current platform version [23]. Additionally, an electronic ride board for arranging lifts among students is based on components of this middleware platform. Beside simulation evalution the applications based on the middleware were tested in an emulative environment were real devices are connected to the simulation. Furthermore also smaller tests with real hardware using IEEE 802.11 and GPS were accomplished.

The communication abstraction is completely put into practice. Additionally, agent movement service, agent transport protocol and flooding based marketplace communication are part of the implemented system. They have been evaluated in [7]. The impact of topology-based routing on efficient marketplace communication is part of current work. All presented service agents are ready to use.

The map computation service is currently developed and using this service to improve performance of geographical position-based routing is subject to further work. Evaluation of the marketplace communication pattern is currently done by defining a fixed set of marketplaces in advance. Dynamic creation of new marketplaces in regions with high device population and localization of these marketplaces is currently not part of the platform but will be added soon. Efficient methods allowing an agent to find its originator's device by using trails of positions left by the device are part of current work.

References

[1] M. Weiser, "The computer for the 21st Century," *Scientific American*, vol. 265, pp. 94–104, September 1991.

[2] J. Hightower and G. Borriello, "Location systems for ubiquitous computing," *IEEE Computer*, vol. 34, pp. 57–66, August 2001.

[3] M. Mauve, J. Widemer, and H. Hartenstein, "A survey on position-based routing in mobile ad-hoc networks," *IEEE Network Magazine*, vol. 15, no. 6, pp. 30–39, 2001.

[4] B. Williams and T. Camp, "Comparison of broadcasting techniques for mobile ad hoc networks," in *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC)*, pp. 194–205, 2002.

[5] E. M. Royer and C.-K. Toh, "A review of current routing protocols for ad-hoc mobile wireless networks," *IEEE Personal Communications*, pp. 46–55, April 1999.

[6] P. Gupta and P. Kumar, "The capacity of wireless networks," *IEEE Transactions on Information Theory*, Mar. 2000.

[7] D. Görgen, H. Frey, J. Lehnert, and P. Sturm, "Marketplaces as communication patterns in mobile ad-hoc networks," in *Kommunikation in Verteilten Systemen (KiVS)*, 2003.

[8] Object Management Group, "Common object request broker architecture: Core specification," technical report version 3.0.2 editorial update, December 2002.

[9] N. Brown and C. Kindel, "Distributed component object model protocol - dcom/1.0," internet draft, January 1998.

[10] Sun Microsystems, "Java remote method invocation specification," tech. rep., Sun Microsystems, 2002.

[11] T. Ledoux, "OpenCorba: A reflective open broker," *Lecture Notes in Computer Science*, vol. 1616, pp. 197–??, 1999.

[12] G. S. Blair, G. Coulson, P. Robin, and M. Papathomas, "An architecture for next generation middleware," in *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, (London), Springer-Verlag, 1998.

[13] S. Zachariadis, L. Capra, C. Mascolo, and W. Emmerich, "XMIDDLE: Information sharing middleware for a mobile environment," in *ACM Proc. Int. Conf. Software Engineering (ICSE02). Demo Presentation.*, (Orlando, FL, USA), May 2002.

[14] G. P. Picco, A. L. Murphy, and G.-C. Roman, "LIME: Linda meets mobility," in *International Conference on Software Engineering*, pp. 368–377, 1999.

[15] N. Davies, A. Friday, S. P. Wade, and G. S. Blair, "L^{2}imbo: A distributed systems platform for mobile computing," *ACM Mobile Networks and Applications (MONET) - Special Issue on Protocols and Software Paradigms of Mobile Networks*, vol. 3, pp. 143–156, Aug. 1998.

[16] S. Ahuja, N. Carriero, and D. Gelernter, “Linda and friends,” *IEEE Computer*, vol. 19, pp. 26–34, Aug. 1986.

[17] K. Herrmann, “MESHMdl - A Middleware for Self-Organization in Ad hoc Networks,” in *Proceedings of the 1st International Workshop on Mobile Distributed Computing (MDC’03)*, May 19 2003.

[18] G. Kortuem, “Proem: a middleware platform for mobile peer-to-peer computing,” *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 6, no. 4, pp. 62–64, 2002.

[19] R. Meier and V. Cahill, “STEAM: Event-based middleware for wireless ad hoc networks,” in *22nd International Conference on Distributed Computing Systems Workshops (ICDCSW ’02)*, (Vienna, Austria), July 2002.

[20] D. Lange and M. Oshima, *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley, 1998.

[21] G. Cabri, L. Leonardi, and F. Zambonelli, “MARS: A programmable coordination architecture for mobile agents,” *IEEE Internet Computing*, vol. 4, no. 4, pp. 26–35, 2000.

[22] E. Sultanik, D. Artz, G. Anderson, M. Kam, W. Regli, M. Peysakhov, J. Sevy, N. Belov, N. Morizio, and A. Mroczkowski, “Secure mobile agents on ad hoc wireless networks,” in *The Fifteenth Innovative Applications of Artificial Intelligence Conference*, American Association for Artificial Intelligence, Aug. 2003.

[23] H. Frey, D. Görgen, J. K. Lehnert, and P. Sturm, “A java-based uniform workbench for simulating and executing distributed mobile applications,” in *Proceedings of FIDJI 2003 International Workshop on scientific engineering of distributed Java applications*, (Luxembourg), Nov. 27–28 2003.