MICA: Pervasive Middleware for Learning, Sharing and Talking

Mohammed Waleed Kadous
School of Computer Science & Engineering
University of New South Wales
Email: waleed@cse.unsw.edu.au

Claude Sammut
School of Computer Science & Engineering
University of New South Wales
Email: claude@cse.unsw.edu.au

Abstract

MICA (Multimodal Interagent Communication Architecture) is a middleware layer for pervasive computing that is especially well suited to sharing of information between users, learning user’s preferences and interacting with the user through many devices and modalities. It is based on the idea of a blackboard: a global shared memory which acts as both a communication and storage mechanism. It uses an extremely simple API that is easy to program, but still small enough to fit on PDAs and mobile phones. It has been used to implement several systems, including a context-aware application that learns the user’s email preferences.

1. Introduction

Previous work by the Natural Adaptive User Interfaces group of the Smart Internet Technology CRC built InCA, a system that allowed a user to interact with a PDA using a natural speech input and output [1, 2]. InCA provides access to personal information (such as e-mail and appointments), as well as Web services (such as exchange rates, translations, weather). Experiments into multimodal interaction with speech and gesture for map navigation were also undertaken.

However, this initial prototype lacked flexibility and became difficult to extend. InCA led to the development of an architecture called MICA that overcame some of the weaknesses of InCA. MICA was designed for simplicity in mind for both the API and the implementation.

In designing MICA, the design goals were to:

- Operate in heterogeneous environments, in particular, it should be able to run on low-power and mobile devices. This also means that it should support low-level programming languages such as C.
- Allow users to use multiple input and output modalities and devices, such as speech, gesture, audio and video; at the same time allowing both multimodal input fusion and multimodal output generation. For example, the user should be able to begin answering an e-mail on her PDA using speech, then when she gets to her office continues working on the same e-mail on her desktop from where she left off.
- Support learning of user’s preferences and patterns of usage.
- Use a simple abstraction that developers could learn and use quickly.
- Support both security and privacy measures.
- Act as an interpersonal communication mechanism, as well as allowing all of one person’s communication tools to communicate with each other.

This article will first discuss the basic design of the MICA architecture, before discussing the implementation and applications built using MICA. It then compares the MICA architecture with other middleware platforms, before drawing some conclusions and future work.

2. MICA Core Architecture

MICA’s is based on the blackboard architecture [3]: group of experts use the blackboard to share knowledge and communicate. When something is written to the blackboard, each expert examines it and sees if he can make a contribution to solving a problem. The experts are known as knowledge sources. A similar idea can be used for as a generic communication mechanism.
In MICA, we update the idea of a blackboard by supporting distributed execution and network communication, using an object-oriented data representation and adding security and privacy mechanisms.

2.1. Entities

There are three entities used in the MICA design:

2.1.1. The Blackboard The blackboard is the core of MICA. The blackboard is similar to a telephone switch with memory. All interactions between agents flow through the blackboard, in much the same way that all phone calls in a town go through a switch.

Much of the power of the blackboard comes from its ability to allow many agents to share information. It is expected, that in use, there would be one blackboard for each person. All agents and services related to that user would execute through the blackboard. It is also expected that forum blackboards, for groups of individuals, could also be created.

2.1.2. Agents Agents are entities that wish to access and contribute to the information on the blackboard. They take many shapes and forms, but the two main types are interaction agents that convey information to and from the user such as GUIs, proxies for telephone connections and devices in the environment, and computational agents that provide services such as speech recognition, web access and e-mail.

Once connected to a blackboard, agents can read, write and query objects on the blackboard. In addition, they can register for new objects written on the blackboard. When a new object of interest to the agent is written to the blackboard, the agent is informed of its arrival.

2.1.3. MICA Objects MICA Objects (or mobs for short) are the basic unit of information in MICA. Mobs are the things that agents actually read and write from the blackboard.

Mobs are very similar to objects in an object-oriented framework, but also share some characteristics with frames. Each mob has a type defined by the user – similar to a class in an object-oriented language. In addition, mobs have slots, which are similar to fields in object-oriented programming – each slot has a name and one or more values. Currently, only string values are supported, but support for other types of values is planned.

An example of a typical mob is shown in Fig. 1, output from the MICA implementation. Its name is line-103 and it has a type line.

2.1.4. Type system MICA supports multiple inheritance: a given type can have several superclasses. Unlike languages like Smalltalk where each object can have multiple types a single mob can only have one type.

To simplify management, there is a universal supertype, called mob. Every type is assumed to inherit from mob. When an object is created on the blackboard, the blackboard also creates two slots, each with a single value: creationTime, the time at which the agent was created; and creator, the agent that created a particular mob. The registration system heeds inheritance; if an agent registers for a given type, the agent is informed if any subtype is also written to the blackboard.

This is an extremely useful feature. For example, to write a debugger agent that displays everything on the blackboard it need only register for mobs and it will be informed of any mob written to the blackboard.

2.2. Communication model

Fig. 2 shows how these entities connect. The arrows represent flows of requests; for example that a mob be written to the blackboard, or that an agent be informed when mobs of a given type are written to the blackboard.

The diagram shows two agents connected to one blackboard. Agents and blackboards do not communicate directly, but through a transport layer. This allows implementations of MICA to use different transport layers. For example, some possible transport layers are: local function calls, XML over a TCP/IP connection, or ASCII messages over Bluetooth. For each
transport, a pair of classes – an agent transport and a blackboard transport – need to be developed. Agents typically talk to the Agent Transport layer through method calls or callbacks (rather than network connections), depending on the language; similarly for the blackboard.

This gives flexibility in the domains MICA can be applied to; and allows developers to write agents that do not need to be aware of the underlying transport layer.

In order to clarify how requests are passed, consider what happens when Agent 1 writes a mob to the blackboard, which is of type `line`. Also assume that Agent 2 has registered for any mobs whose type is `line`. Agent 1 would tell its transport to an object to the blackboard. Agent 2 would then be informed of the new object.

Note that neither Agent 1 nor Agent 2 are explicitly aware of one another and that different agents can register for mobs of a given type. If six agents were registered for `lines`, then all six of them would be informed. Also note that the arrows are “two way” arrows. At times the blackboard will initiate communication with agents.

2.3. Security, privacy and authentication

Security and privacy in MICA are based on nyms. A nym is an aspect of the user’s identity that the user wishes to reveal and represents a spectrum of access rights. For example, anonymity means that minimal details of the user are to be disclosed, whereas verynymity is certainty of the user’s identity. Typically, a user will have multiple nyms, depending on the type of information the user wishes to disclose.

When an agent connects, it is associated with a nym. A nym is authenticateed using standard public-key encryption systems; by means of a challenge from the blackboard to the client. Several agents with the same nym can connect at once.

For privacy, the nyms are used to control visibility of the information on the blackboard. The core of this idea is to realise that not every agent need see the same information on the blackboard; and generally, each nym sees a different version of the blackboard. Agents are not informed when a mob they are not supposed to see is written to the blackboard, and similarly, should they do a query, only the mobs they are supposed to see are returned.

The visibility of the information on the blackboard is governed by a security policy. The security policy consists of rules for each nym that specifies what types of mob are visible to what nyms. The security policy can also specify constraints on the values of slots.

For example, consider the office of the future where on the front door is an LCD panel that says whether the owner of the nym is in or not. This LCD panel would connect to the user’s blackboard with say, a nym of `anon`. If the person in the office is in, but does not wish to be disturbed, the security policy may prevent the LCD panel from seeing mobs that indicate the owner is inside.

2.4. Interblackboard communication

Our vision is that the blackboards are not only to be used as global servers in the way that similar systems have been used in pervasive computing, but rather that each individual has their own personal blackboard that tends to their multi-modal and multi-devices communications.

MICA uses the concept of a forum blackboard. A forum represents a “shared space” for blackboards to connect, and represent either a physical or a virtual group of blackboards. A forum blackboard may also have services associated with it other than being a meeting point. For example a meeting room may have a blackboard associated with it. This forum blackboard would give access to any resources in the room, e.g. speakers, panels etc. However, a forum may also be a totally virtual space.

It is envisaged that initially each user’s personal blackboard will be carried around with them, or run on their home machines. Forum blackboards for physical spaces, such as meeting rooms, would be embedded in that environment. Forum blackboards for virtual spaces would operate within the Internet in much the same way as a web server runs today.
This design is shown in Fig. 3. To support interblackboard communication, Interblackboard Agents (IBA) are used. One complication of interblackboard communication is the handling of identity; and how this fits with the nym concept. A connection between two blackboards can be considered an exchange of nyms; when two IBAs initiate contact, they exchange their opposite nyms; to ensure proper access rights. Once the IBAs are connected, a mirroring policy is constructed, much like the security policy previously mentioned. If a mob is written to a blackboard and if it is covered by the mirroring policy, the IBA will also write it to the companion blackboard. The mirroring policy interacts closely with the security policy to ensure that users’ privacy is not violated.

3. MICA implementation

The current implementation of MICA consists of an implementation of the blackboard and the agent library in Java. It is functional, and has been used for several small projects, but not all the functionality above has been fully implemented. Currently, object delivery is complete and the type system is functional. Two transport layers are also implemented: XML-over-TCP and a local method-call based protocol.

3.1. Performance

Some simple evaluations have been done on the speed of MICA. With the current unoptimized Java implementation, the system can distribute approximately 200 messages a second with the XML-over-TCP transport layer. There is a latency of about 50 milliseconds from when an agent writes an object to the blackboard and if it is covered by the mirroring policy. We have tested with up to six clients connected to a single blackboard at once over Ethernet and it still “feels” responsive.

We have used a real-time whiteboard-style application that naively writes each drag by the user directly to the blackboard (without doing any kind of consolidation of lines, or reduction in data). We have tested with up to six clients connected to a single blackboard at once over Ethernet and it still “feels” responsive.

The MICA jar file, including blackboard, agent libraries, demonstration programs, is just over 100K. Importantly, the API is extremely simple. To implement an agent, a developer needs to write two methods or callback functions: \texttt{init()} to start the agent, and \texttt{handleMob()} to handle any mobs the agent has registered for. The API for accessing the blackboard from the agent’s perspective consists of only 8 API function calls. The C API is similar and is approximately 20K in size.

3.2. Applications

Two test applications have been written using the existing infrastructure.

3.2.1. SharedPad

SharedPad is a simple “whiteboard” style application. Users can scribble on a SharedPad and the information is written to the blackboard. This then communicates the information to all the other attached SharedPad in real time. By making a MICAQL query on startup, a SharedPad can also retrieve the existing scribbles made before it connected. Thus, the user can doodle on their PDA (in our case, a H3870 running Familiar Linux and GPE), then connect to the blackboard from their desktop and continue doodling from where they last stopped doodling on their PDA. SharedPad is surprisingly simple. The Java implementation is approximately 135 lines, with approximately 35 lines relating to the blackboard.

3.2.2. WhizBang

The WhizBang application demonstrates coordination between several agents to learn the user’s e-mail reading preferences in a variety of environments. The main agents involved in the application are:

- A GUI that allows the user to set certain environmental aspects that are a substitute for future intelligent environments, such as the user’s location, who else is around, and the background noise level. It also allows users to type information into a terminal.
- A learning agent that uses Weka [5, 6], a learning library written in Java. In this particular case, it is used to decide whether to read out a user’s email using text to speech or display it as a list.

WhizBang can learn rules such as “when the user is in the car, read out emails, but if the user is at home and there are more than 15 e-mails display it on a computer screen; otherwise read them out.” If the system learns an incorrect rule, the user can correct the system by saying that it did the wrong thing; and the correction is used to learn new rules.

Perhaps the most interesting agent is the learning agent. Other agents tell the learning agent to learn given the current context; without the other agents being necessarily concerned with the context itself. This takes advantage of the visibility of the context through the blackboard. The context and correct action (or the
class in the machine learning nomenclature) are then used to construct a model of the user’s preferences.

This demonstration illustrates how easy it is (i) to build systems with MICA with a variety of agents (ii) implement a system that learns user’s preferences through the “openness” of the blackboard.

4. Alternatives and previous work

Recently, several types of middleware have been developed specifically for use in pervasive computing. MICA is similar to some of these. In spirit, MICA is very similar to T Spaces [9, 10]. There are some key distinctions compared to T Spaces: The data model is object-oriented, rather than being built on tuple spaces; MICA is not locked in to either the Java language nor to TCP/IP as the transport layer. MICA already has two transport layers (local method calls and XML over TCP) as well as several implementations of the agent library (C and Java); The API is significantly simpler: without complex operations like the rhonda operation, or blocking wait commands.

It is also similar in many ways to the Event Heap being developed as part of the iROS [11]. In addition to the services provided by T Spaces, the Event Heap provides some platform and language independence.

The main distinction with existing work is how blackboards are used in MICA: it is envisaged that every user has their own personal blackboard, and that these personal blackboards can connect to one another using forum blackboards. Further, our work is distinguished by its security and privacy model.

5. Future work

Our immediate plans are to complete the implementation of MICA. We then plan to reimplement an extended version of InCA, the prototype discussed at the beginning of the article, on top of MICA.

6. Conclusion

MICA is a simple abstraction for pervasive computing that is small and elegant. It functions as both a medium of communication as well as storage that is well-suited to supporting multimodal and multi-device interaction. It also supports privacy and communication between different users and environments, as well as a demonstrated ability to employ learning to learn user’s preferences.

It has been tested on two small demonstration domains, where it has shown promise. Further explorations of its usefulness for more complex domains are in progress.

Acknowledgment

The authors would like to thank Dr John Zic for his assistance with the development of the security model. They would also like to acknowledge the support of the Smart Internet Technology CRC.

References