Developing Mobile Computing Applications with LIME

Gian Pietro Picco¹, Amy L. Murphy², Gruia-Catalin Roman²

¹ Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza Leonardo da Vinci, 32
20133 Milano, Italy
picco@iet.polimi.it

² Department of Computer Science
Washington University in St. Louis
Campus Box 1045, One Brookings Drive
St. Louis, MO 63130-4899, USA
{alm,roman}@cs.wustl.edu

ABSTRACT
Mobile computing defines a very dynamic and challenging scenario for which software engineering practices are still largely in their initial developments. LIME is a middleware designed to enable the rapid development of dependable applications in the mobile environment. The model underlying LIME allows for coordination of physical and logical mobile units by exploiting a reactive, transiently shared tuple space whose contents changes according to connectivity. In this demonstration, we report about initial experiences in developing applications for physical mobility using LIME.

1 INTRODUCTION
Mobile computing is becoming increasingly popular, partly due to the wealth of mobile devices that are enabling untethered communication as people move through physical space. Additionally, mobile code and mobile agents which roam the logical space of network hosts are being exploited as a new design paradigm for distributed applications. Nevertheless, models, architectures, and technologies for mobile computing are still in their early stages of development, and are only beginning to define and address the complex challenges posed by mobility. As mobile components migrate, they must continuously adapt in order to access the newly available resources (e.g., data) and to interact with other components (mobile or not) as connectivity becomes available. Collectively, these environmental properties form the context of a mobile component. Our goal is to devise an abstraction of the context which provides conceptually clean access to the changing environment, thus allowing the mobile application programmer to focus on the application itself rather than the details of the changing environment. This modeling goal is realized as a mobile middleware called LIME (Linda in a Mobile Environment). In this paper, the discussion of two applications involving physical mobility is the opportunity to outline the LIME model and show how it is exploited by each application. Finally, we briefly report about the status of the current implementation and draw some conclusions about this experience.

2 MOBILE COMPUTING APPLICATIONS
Applications for the mobile environment can be classified roughly into two typologies: those which focus on data sharing and those that are concerned with transient interactions with other components as the context changes. In this section, we present two mobile application scenarios and challenges that must be addressed for a successful implementation.

Figure 1: ROAMINGJIGSAW. When a player is disconnected (left), only the pieces previously selected can be assembled. Upon reconnection (right), the assemblies made meanwhile by other players become visible in the player’s workspace.

ROAMINGJIGSAW: Accessing Shared Data
Our first application, ROAMINGJIGSAW as shown in Figure 1, is a multi-player jigsaw assembly game. A group of players cooperate to assemble a jigsaw puzzle in a disconnected fashion. Play begins when a player opens the box of puzzle pieces and makes them available to other players. Players may connect, select the pieces they want to own and work with, and then disconnect. Thus, assemblies can be constructed independently and intermediate results, together with previously hoarded pieces, can be shared again when connectivity is reestablished. Although disconnected players can work only with the pieces they previously selected, the workspace displays also the other pieces they have seen but are
Currently owned by other players, who can manipulate them independently. Hence, the perspective displayed in each player's workspace is only weakly consistent with the global state, as it represents only the last known information about each puzzle piece. Full consistency is prevented by disconnections in the mobile scenario. The challenges that must be addressed to develop ROAMINGJIGSAW involve enabling the access to selected pieces after disconnection, the propagation of assemblies and selections when connectivity is available, and the reconciliation of the workspace with the connected players upon reconnection.

Although ROAMINGJIGSAW is only a game, it nevertheless exhibits the characteristics of a more general class of applications in which data sharing is the key element. Thus, the design strategy of ROAMINGJIGSAW may be adapted easily to applications where the nature of data is different, e.g., sections of a document in a collaborative editing application or paper submissions to be evaluated by a program committee, but the patterns of shared interaction are the same.

![Figure 2: RedRover. The main console of RedRover, and the most recent camera image of a connected player.](image)

RedRover: Detecting Changes in Context

Our second mobile application is a spatial game we refer to as RedRover, where individuals equipped with small mobile devices form teams and interact in a physical environment augmented with virtual elements. This forces the participants to rely to a great extent on information provided by the devices and not solely on what is visible to the naked eye. The primary objective of RedRover is to search for clues in an unknown environment, discover the flag of the other team, and cluster around the player who finds the flag. Each player is equipped with a digital camera that can be used to share a snapshot of the current environment with team members who may be physically separated by walls or other barriers, but still within communication range. Players know their location in space, and sharing of this information with all connected players allows maintenance of an image of the playing field displaying the relative location of all participants. This view is the dominant element of the display, and maintaining its consistence represents the biggest implementation challenge. The application must know which other players are around, and be able to take actions, namely update the screen, as soon as mobile components arrive or depart.

Again, RedRover exhibits similarities to real world scenarios, such as the exploration of an unknown area by a group of people or robots. Our current efforts include the incorporation of a mapping mechanism which will allow users to recognize landmarks in their environment and share this information as they meet other users.

3 Linda in a Mobile Environment

The model fostered by Lime [3] aims at identifying a coordination layer that can be exploited successfully for developing applications that exhibit either logical or physical mobility, including those presented above. To achieve this goal, Lime borrows and adapts the communication model made popular by Linda [1].

In Linda, processes communicate through a shared tuple space, a repository of elementary data structures, called tuples, that can be accessed concurrently by several processes. Each tuple is an ordered sequence of typed data. Tuples are inserted using the out(t) operation on the tuple space, and can be removed by executing in(p), where p is a template used to identify tuples based on pattern matching against their content. Tuples can also be read from the tuple space using the rd operation. Both in and rd are blocking. A typical extension to this synchronous model is the provision of the asynchronous primitives inp and rdp, called probes, that allow non-blocking access to the tuple space.

Linda characteristics resonate well with the mobile setting. Communication in Linda is decoupled in time and space, i.e., senders and receivers do not need to be available at the same time, and mutual knowledge of their location is not necessary for data exchange. Decoupling is of paramount importance in mobility, where the parties involved in communication change dynamically due to their migration.

Nevertheless, when mobility is fully exploited, as with ad hoc networks, there is no predefined, static, global context for the computation, as assumed by Linda. Rather, the current global context is defined by the transient community of mobile units that are currently present, to which each unit is contributing its own individual context. Since these communities are dynamically changing according to connectivity and migration, the context changes as well. This observation alone leads to the model underlying Lime. Although still based on the Linda notion of a tuple space, Lime exploits it in a radically different way.
The Core Idea: Transiently Shared Tuple Spaces
In the model underlying Lime, the shift from a fixed context to a dynamically changing one is accomplished by breaking up the Linda tuple space into many tuple spaces, each permanently associated to a mobile unit, and by introducing rules for transient sharing of the individual tuple spaces based on connectivity.

One way to visualize this concept is by imagining a global, virtual tuple space containing the individual tuple spaces of all mobile units. Based on current connectivity among sets of mobile components, different projections of this global space are available to the mobile components, forming what we refer to as federated tuple spaces. For example, in ROAMINGJIGSAW, the global virtual tuple space contains all puzzle pieces, but only the pieces selected by players in contact are part of the federation and can be manipulated. As players arrive and depart, the set of available puzzle pieces changes.

From the perspective of the mobile unit, access to data is uniform regardless of the current connectivity. In other words, while the available contents may change, the mobile unit’s style of interaction does not. In fact, basic interactions occur using the same primitives as in Linda but they operate over the federated tuple space. The process of growing the shared data when connectivity is established is referred to as engagement and occurs as a transaction. When components disengage, the accessible portion of the data space shrinks to remove those tuples and reflect the new sharing.

This idea of transient sharing of tuple spaces is a very powerful abstraction. It provides a mobile unit with the illusion of a local tuple space that contains all the tuples coming from all the units belonging to the community, without any need to identify explicitly each one.

Degrees of Context Awareness
Thus far, Lime appears to foster a coordination style that reduces the details of distribution and mobility to changes in what it is perceived as a local tuple space. This view is very powerful for simplifying applications such as ROAMINGJIGSAW, where data is accessed uniformly and independently of its location. Nevertheless, this view may hide too much in cases where the designer needs a more fine-grained control upon the portion of the context to be accessed. For instance, in REDROVER a player can ask for the camera image from a given player. This should be accomplished by looking into that player’s tuple space, with no need to query other tuple spaces for data whose location is known. Lime provides this control by extending the Linda operations with tuple location parameters that allow access to projections of the transiently shared tuple space.

The out[\lambda] operation extends out with a location parameter representing the identifier of the agent responsible for holding the tuple. The semantics of out[\lambda] involve two steps. The first step is equivalent to a conventional out(t), the tuple t is inserted in the local tuple space of the agent calling the operation, say \omega. At this point the tuple t has a current location \omega, and a destination location \lambda. If the agent \lambda is currently connected, i.e., either co-located or located on a connected mobile host, the tuple t is moved to the destination location. The combination of the two actions are performed as a single atomic operation. On the other hand, if \lambda is not currently connected, the tuple remains with \omega. This “misplaced” tuple, if not withdrawn, will remain such unless \lambda becomes connected. In this case, the tuple will migrate to the tuple space associated with \lambda as part of the engagement transaction. Hence, using out[\lambda], the caller can specify that the tuple, albeit shared, is supposed to be placed within the tuple space of agent \lambda. This way, the default policy of keeping the tuple in the caller’s context until withdrawn can be overridden, and more elaborate schemes for transient communication can be developed. Location parameters also provide variants of the in and rd operations that allow access to a slice of the current global context. In Lime, these operations are annotated as in[\omega, \lambda] and rd[\omega, \lambda], where the current and destination locations defined earlier are used. Finally, disengagement utilizes tuple location in order to separate the tuples owned by the departing mobile unit from the remainder of the tuples in the federated tuple space. It should be noted that, in practice, no tuple transfers are needed during disengagement as the tuples remain physically distributed and co-located with the agents that own them.

It is interesting to note that the extension of Linda operations with location parameters, as well as the other operations discussed thus far, foster a model that hides completely the details of the system (re)configuration that generated those changes. For instance, if a rd[\omega, \lambda] probe for a camera image fails, this gives no information about whether the agent is not present, or it is present and an image is not available. Without awareness of the system configuration, only partial context awareness can be accomplished. For example, ROAMINGJIGSAW maintains a weakly consistent view of the puzzle pieces while the main display of REDROVER requires that all currently connected components be displayed and all components that were once connected be represented as “ghost images”—a much stronger consistency guarantee. Lime provides awareness of the system configuration using the same set of tuple space abstractions discussed thus far but through a separate LimeSystem tuple space, a system-maintained, read-only tuple space whose data represents the currently connected components. The combination of the transiently shared tuple spaces and the LimeSystem tuple space enable the definition of a fully context aware style of computing.
ACKNOWLEDGMENTS

The authors wish to acknowledge Jason Chander, Brian Mesh, and Brian Payne for their contributions in the implementation of the RoamingCrews software, and Andrew Pincus for his assistance with mobile computing tasks.

REFERENCES


5. CONCLUSIONS

Our experiences with application development using Lime and Lime RT provide further evidence that mobile computing is an important area of research. The Lime platform, in particular, offers a unique opportunity to explore the potential of mobile computing, and we look forward to seeing the results of future work on this topic.

4. CURRENT IMPLEMENTATION

Lime is currently implemented in Java, and it provides support for the Java platform. The Lime platform can be used to develop applications that run on a variety of mobile devices, including personal digital assistants (PDAs), cellular phones, and Internet-capable devices.

The Lime platform provides a number of key features, including:

- Support for Java applications
- Support for mobile devices
- Support for network connectivity
- Support for mobile data management
- Support for mobile security
- Support for mobile user interfaces

In addition, Lime provides a number of tools and utilities for developing mobile applications, including:

- A Java compiler
- A Java debugger
- A Java profiler
- A Java testing framework
- A Java security framework

Overall, Lime provides a powerful and flexible platform for developing mobile applications, and it is well suited to the needs of developers working in this exciting field.