

Enabling Distributed Applications for Smart Phones

Nick Palmer
nick@sluggardy.net

Vrije Universiteit Amsterdam

Masters in Parallel and Distributed Computer Systems

Advised by Professor Henri Bal and Professor Thilo Kielmann

August 31, 2007

Dedication

With humble gratitude for all of you have given me I dedicate this work:

to my family:

Byron: For sparking my interest in computers so many years ago, giving me my first lessons in programming, and always being there to bounce ideas around with. Without you I would not be working with computers today.

Chris: For your loving support in my darkest of hours and having faith in my abilities. I am most proud of myself when you are proud of me.

Max: For being my exemplary model of a hard working, dedicated, responsible person with a big kid's heart. (Nobody would ever suspect it is part cow.) If I were half the man you are I would be so much greater than I am.

Ed: For encouraging me to return to academia and supporting me at every step of the journey. Without you I could not have started or finished this program.

Helen: For all your efforts over the years which have helped me with my English; from reading to me, to working with me on spelling, to editing this and so many other papers, you have always been my teacher.

Cyndi: For all your loving packages which saw me through the dark Amsterdam winters.

Nettle: For showing me the power of living by your values. My admiration knows no bounds.

to my advisers:

Professor Henri Bal For giving me the chance of a lifetime and enough rope to hang myself with. I will always be thankful for the opportunities, freedoms and support you have given me.

Professor Thilo Kielmaan For pushing me to do better and encouraging me to think more critically about the problems at hand.

to my friends:

Scott, Julia, and Grace for a very special summer and so, so much more; American Tom for all your experienced advice and support, and for helping me to identify what I want to do with my life; Alessandra e Reno per produrre la mia luce ed dando sostegno la nostra vita insieme; Erik and Chiara for being such good, geeky friends; Kel and Tina for reminding me of my value; Brian for all your work building the business that saw me through the last year; Cassidy for keeping my things safe over the summer; Anna and Jenny Rae for helping me find my spirit; and finally, to the many more friends I lack the space to list. You have all helped me more than you can possibly imagine and I owe you all a debt of gratitude for your friendship, support, and kindness over the years.

and in particolare alla luce della mia vita:

Viola: Siete la mia stregghetta, mio pesciolino, mio scopritore del nomi, ma principalmente mia stella guidante. Siete la luce sul mio percorso. Mi avete insegnato l'amore appassionato, indicato me dove ho bisogno del miglioramento e come posso adattarsi e così tanto tanto molto più. Il mio futuro è riempito con vostra magica luce. Grazie mille sempre.

Contents

Dedication	1
Contents	2
1 Introduction	4
1.1 The Promise of Smart Phones	4
1.1.1 Approach	5
1.2 The Problems with the Platform	5
1.2.1 Distributed Applications	5
1.2.2 Resource Limitations	6
1.2.3 Security	6
1.2.4 Hardware Diversity	6
1.2.5 Operating System Diversity	7
1.2.6 The Walled Garden	8
1.2.7 Summary	8
1.3 Why Not Use Web Technology?	8
1.3.1 Problems with the Browser	8
1.3.2 Non Client-Server Applications	9
1.3.3 Screen Sizes	9
1.4 Understanding The Problems	10
1.4.1 Contributions	11
1.4.2 Analysis	11
1.4.3 The Application	12
1.4.4 The New API	13
2 Related Work	14
2.1 Smart Phone Technologies	14
2.1.1 Operating Systems	14
2.1.2 J2ME: Java Micro Edition	15
2.1.3 The Need for Serialization	15
2.1.4 The Code-Signing Problem	16
2.2 Ibis: Distributed Communication API for Grids	17
2.3 OSGi: Open Services Gateway Initiative	18
2.4 ETC: Equip Component Toolkit	19
3 Katrina: A Case Study	20
3.1 Introduction	20
3.2 Technological Impact: Lack of Connectivity	20
3.3 Possible Applications	22
3.3.1 People Finder	22

3.3.2	Situation Reporting	23
3.3.3	Bus Management	24
3.3.4	Evacuation Management	24
3.3.5	Shelter Management	24
3.3.6	Summary	25
3.4	Implications for Technology	25
4	Sahana: Web-Based Disaster Management	27
4.1	Sahana Modules	27
4.1.1	Sahana Missing Person Registry	28
4.1.2	Sahana Volunteer Registry	28
4.1.3	Sahana Organization Registry	28
4.1.4	Sahana Shelter Registry	29
4.1.5	Sahana Request/Aid Management System	29
4.1.6	Sahana: Possibilities for Smart Phone Integration	30
4.2	Criticisms of Sahana	31
5	People Finder Application	32
5.1	The Approach	32
5.2	Design	34
5.3	Implementation	37
5.4	Analysis	37
5.5	Limitations	38
6	RAVEN: A New API for Distributed Applications	39
6.1	Design	40
6.2	Storage API	40
6.3	Communication API	43
6.4	Callback API	44
6.4.1	Generic Methods	45
6.4.2	Storage Methods	45
6.4.3	Communication Methods	46
6.5	Implementation	48
6.5.1	Storage System	48
6.5.2	In Memory System	48
6.5.3	JSR 75 System	49
6.5.4	Communication System	49
6.6	Analysis	50
7	Conclusion	52
7.1	Our Contributions	52
7.2	Future Work	53
7.2.1	Serialization	53
7.2.2	Wireless Internet Support	54
7.2.3	Versioning	54
7.2.4	Security	55
7.2.5	Component Integration	56
7.2.6	On Phone Development	56
	Bibliography	57

Chapter 1

Introduction

1.1 The Promise of Smart Phones

Smart Phones are devices that combine the functionality of cellular phones with those of personal digital assistants, media players and other hand-held computers. They contain an increasingly large set of features including internet connectivity, cameras, audio recording and playback, global positioning services (GPS), accelerometers and similar sensors, as well as the traditional phone features. There are an increasingly large number of applications being deployed on this platform, many of which take advantage of the features offered and many of which look nothing like traditional phone applications. The market has shown that people seem to be willing to carry this kind of converged device with them, with growth of 42 percent in 2006 [17] in this market with 64 million smart phones shipped in 2006[19]. Other hand-held computing platforms have not experienced nearly the level of acceptance that smart phones offer, and sales have declined as consumers have moved to purchasing smart phones[18]. Smart phones are positioned to be the next big wave of computing and are already being hailed by many as the beginning of a new era of mobile computing.

One of the distinct advantages of this new platform are the location-aware applications that the combined communication and GPS functionality provide, so-called convergence of features. These features raise significant questions of privacy that must be addressed[20, 5, 26]. Nonetheless, this combination of features opens up novel applications such as traffic monitoring and

analysis, portable directions, emergency and rescue applications, location of friends and family, location aware distributed games. These applications are only possible on a platform that combines location awareness with easy communication. The other distinct advantage of this platform is its distributed nature. A great deal of literature is dedicated to so called “Ubiquitous Computing,” where computers are embedded in everything we interact with and distributed everywhere. The distribution of smart phones is a first step in this direction offering a distributed platform that is everywhere. There are many unique applications that can take advantage of this distribution like distributed gaming and location-aware applications such as guided tours. Although many of these applications are interesting, they are just the “tip of the iceberg” in what is expected to be a flood of new applications for smart phones.

1.1.1 Approach

In order to achieve the long-term goal of creating a development environment for distributed smart phone applications, it is necessary to better understand what is needed by applications desiring to use the smart phone platform. Of particular interest are applications which take advantage of the distributed nature of the platform while also taking advantages of unique features they offer such as location awareness. In order to better understand the needs of applications we have chosen to examine a single application that can take advantage of these features. This gives us a top-down view of what a development environment should provide. This knowledge can then be applied to the creation of an application development framework in order to provide the features application developers are likely to require.

1.2 The Problems with the Platform

1.2.1 Distributed Applications

Although the hardware platform is promising and offers many unique advantages and although applications are already being developed for the platform, it is well known that building distributed (networked) applications is uniquely challenging. Distributed applications are notorious for not only being difficult to write but even more challenging to debug.¹ Without simple ways

¹Distributed application state is not, in general, directly observable, which brings to mind Heisenberg’s Uncertainty Principle. This in and of itself should be a warning about the complexity of dealing with distributed debugging.

for developers to take advantage of the distributed nature of the platform, many applications which could harness this distribution may never be built.

1.2.2 Resource Limitations

Smart phones have the additional challenge of being resource-limited platforms in terms of processing power, network bandwidth, memory, screen size, and battery life. The typical desktop computer has access to only a single network, virtually unlimited electricity, and a large monitor. Contrast this with smart phones, which have traditional GSM communication, bluetooth, and wireless internet networks to work with, all of which is powered by a single battery and displayed on a relatively small screen. There is also the question of user interaction. Some phones have a touch screens with a pen and some have a soft key interface, while still others have a complete QWERTY keyboard, or even only a touch screen as is the case with the iPhone from Apple. All of these resource limitations further complicate development for this platform.

1.2.3 Security

Security, authentication, and authorization are of paramount importance for many applications, which further complicates the process of building novel applications. It is well known that the original cell phone standards did not consider security closely and broadcast both the phone number and the password in the clear, allowing phones to be easily cloned[15]. Security is also notoriously difficult to get right. Many deployed security protocols, for example, Kerberos version 4, have been deployed only to have serious problems discovered after deployment[31]. When deployment consists of hundreds of thousands of phones, fixing these kinds of problems can be prohibitively expensive. Thus it is important to have security that is easy for application designers to implement. This security often involves cryptography, which can be expensive in terms of processor power and is thus at odds with the resource limitations of the platform.

1.2.4 Hardware Diversity

One of the major challenges with developing for the smart phone platform is the diversity of the hardware on which applications must run. Smart phones have a variety of processors. Any development environment that wishes to run on a wide variety of phones must either compile

code for a wide variety of processors or run inside a virtual machine such as the one provided by Java. Diversity also exists in terms of screen size, which significantly increases the challenge of screen layout. There is also diversity in terms of input methods including the number of so-called soft keys that are available on a particular phone. Finally there is diversity of connectivity methods: phones feature a variety of methods from short range like infra-red and bluetooth to Wi-Max and other broad band for cell phone technologies. There is diversity in GPS availability: some phones have real GPS features, some have assisted GPS where the cellular signal is used to assist the GPS unit in getting a location, some offer external bluetooth devices for GPS features, and some phones offer no GPS features at all. Finally, there is diversity in on-board sensors such as accelerometers or temperature and pressure sensors. All of this diversity significantly increases the barrier to entry for novel applications that need to work on a variety of phones.

1.2.5 Operating System Diversity

There is also considerable diversity in the operating system used on smart phones. The operating system with the greatest market share is Symbian OS, but the reality of the situation is more complex since the operating system itself has considerable issues and so vendors have written different middle-ware systems that use Symbian as a kernel. Writing for these systems is in no way the same. Furthermore, Symbian isn't the only operating system on smart phones today. There is also Windows CE, which has considerable market share; Linux, which is gaining ground; Research In Motion (RIM), which powers the popular Blackberry and Palm; not to mention the new iPhone from Apple, which runs a stripped-down version of OS X. Having so many operating systems creates a significant barrier to innovation and prevents many smaller software developers from entering the market or restricts their market to a subset of available phones. Often those that do create novel applications pick a single operating system based on the kind of user they are targeting: Windows CE for North American or Business Users or Symbian for European and non-business users. This fragmentation further hinders development of novel applications. Developers are faced with a choice between complex development that targets multiple operating systems or reduced market penetration.

1.2.6 The Walled Garden

One other significant issue with the platform is the so called “Walled Garden” that many providers have elected to use to protect their revenues[23]. Many providers have elected to limit users’ access to new features like the GPS, Wireless Internet, or Voice-Over-IP features on their phones in order to keep these features from competing with existing services and to try to squeeze every last drop of revenue from their customers. Although some vendors, such as the company “3”, have elected to open their networks, it is clear that some providers are not yet ready to be as open with their networks. Open source systems like those based on Linux may have an impact on this[22], but it is unclear yet how big an impact given the lower market penetration of Linux-based devices. Nonetheless it is clear that carriers’ efforts to control the revenue from these devices is stifling innovation and competition from outside developers and prevents certain applications from even being developed. A good example is Voice-Over-IP features which are a threat to the core of many carriers’ business models. Voice-Over-IP features are more common in Europe but are almost non-existent in the more tightly controlled United States market.

1.2.7 Summary

The combination of the inherent challenges of building distributed applications, diversity of hardware features, diversity of operating systems, and the constraints of the platform seriously hampers the development of new applications. Therefore what is needed is a development environment that makes creation of applications for smart phone platforms fast and easy and can provide broad market penetration. Currently it is unlikely that the fragmentation seen in the market is going to go away, so tools must be developed which can deal with the fragmentation that currently exists. Without these tools the promise of the smart phone platform may not be realized.

1.3 Why Not Use Web Technology?

1.3.1 Problems with the Browser

Many people argue that simply having browsers on smart phones is all that is required for smart phone applications. This strategy has been adopted by Apple’s iPhone, which allows

development of only web based applications on the phone[16]. Although there are certainly many applications where this may be true, as is clearly demonstrated by the many web based applications offered by Google, there are many issues with simply offering a browser on smart phones. Chief among these problems is application's that do not fit the client-server model but screen size and processor power also offer a significant problem for browser-based solutions. Another problem that we are concerned with is lack of connectivity. Web-based applications are only accessible when the network is available, which may not always be the case.

1.3.2 Non Client-Server Applications

The foremost issue for a web-application-only environment is that it limits the design of applications to client-server-style applications. Truly novel peer to peer applications do not fit into the client-server model that a browser-based solution forces. Furthermore, browser applications cannot interact with unique phone features such as a camera and GPS via a browser. The client server model is not applicable to many forms of applications and limits access to the very features that make the platform so compelling. The most obvious of these may be distributed gaming where users participate in a game on their smart phone via the bluetooth connection and anyone may join or leave the game at any time. Having a single phone act as a central server would mean that phone could never leave the game; it would also considerably tax the phone's resources, preventing the application from scaling properly. Such peer to peer style applications are well known with the growth of the various file-sharing networks. Although these networks are often currently used to share media this is not intrinsic to peer to peer networks. Many novel applications may be built that do not follow the client server model if an API existed that made it easy for application designers to do so. Other good examples include context-sensitive applications where features of the environment may be detected by the phone and activate new applications and features on the phone. For example the phone might detect a projector in the room and offer to send a presentation file to the projector. These applications do not fit well into a browser.

1.3.3 Screen Sizes

Second among issues with a web only design is screen real-estate. Web pages are usually designed for larger screen sizes than cell phones have. Gone are the days when web pages ensured that

they looked good at 640 by 480. Many sites are now designed with resolutions as high as 1024 by 768 in mind. Such high resolutions are unlikely to be supported on cell phone screens and, even if they are, any text on the screen is likely to be far too small to be legible. Furthermore, it has been shown that screen size affects user navigation behavior and perceptions significantly[8]. Thus sites need to be designed with cell phone screen sizes in mind. Although some sites may do this, it is unlikely that a large enough percentage will keep users from being frustrated with the surfing experience on cell phones. Although zooming and page overviews can help to alleviate the problem, fundamentally they do not address the needs of the users of the devices.

Furthermore, because of the limited processing power of the typical smart phone, the use of Javascript on pages accessed by cell phones may prove problematic. More and more websites are designed in such a way that they do not work in browsers that lack Javascript functionality and are often designed for wide screens. One technology that attempts to address this issue is the Wireless Application Protocol (WAP) based devices that make it possible to access Wireless Markup Language (WML) based services with mobile browsers. However, WAP requires sites to recode all pages using WML and run another service to serve up these pages. Adoption is not widespread as content providers do not wish to maintain two copies of all content. Proof that WAP is a non-starter is the use of a full browser in Apple's new iPhone. The phones have adapted to the web instead of the other way around as WAP proponents had hoped.

1.4 Understanding The Problems

In order to better understand the problems this exciting new platform faces we have chosen to approach the problem in both a bottom up and a top down fashion. This means that we have written both an application targeted at the platform as well as a lower level API which the application needed. In this way we can understand the problem more completely and further our goal of making distributed applications for smart phones easy to implement.

One of the areas that promises high utility for new applications of distributed smart phones is emergency response. Emergencies pose unique challenges in terms of the need for a distributed response to the emergency event. They also may take advantage of the location awareness of the platform in particularly useful ways. In the case of large-scale disasters, such as a hurricane,

an earthquake, or a tsunami, which have all made headlines recently, the response has to be spread across a large geographic area. Developing an application of use during such an event poses the further question of what kinds of applications are appropriate for disasters. What happened during the recent landfall of hurricane Katrina when it struck the Gulf Coast of the United States and New Orleans significantly illuminated the issues. We also examine an existing application for disaster management called Sahana in order to understand what kinds of features already exist and how smart phones can play a role in this area.

Based on this examination of the requirements of a disaster we have come up with a distributed application to assist in disaster management based on a diffusion of data concept. This application can be used to collect data in a distributed fashion with the goal of building eventual consistency of the data at each individual node through exchanges between nodes when connectivity can be established. This application necessitates a new API for dealing with the data exchanges that has been partially implemented.

1.4.1 Contributions

Our contributions to this area include a significant analysis of the needs of disaster management applications and several novel ideas about the role that smart phones may play in such a scenario as well as an example of such an application for tracking lost and found people. This application is easily extensible to tracking additional types of data that might need to be collected during a disaster; it proves that enabling users to decide what data is appropriate to collect is certainly within the power of existing smart phone platforms. A simple API has been developed for managing both the storage of the collected data as well as the exchange of that data on smart phones. The usefulness of smart phones to disaster management has thus been demonstrated as well as the need for more useful tools on smart phones for developing distributed applications. Although the implemented API is far from feature complete, improvements will be undertaken soon.

1.4.2 Analysis

Our analysis of existing smart phone technologies has identified several areas that need to be addressed in order to truly realize the promise of the smart phone platform. Although work

is being done on these areas, they are far from mature. Chief among our concerns is enabling distributed communication where the cellular network is unavailable. An API that enables such data sharing is the primary goal being addressed out of this analysis.

The analysis related to the needs in a disaster management situation has suggested several applications that could be deployed on smart phones in future disasters. We are unaware of any similar analysis that targets smart phones as enabling technologies for disaster management. Many of the applications the analysis has suggested could be quite simple to implement but would prove invaluable to emergency response teams. There are also several needs for application development environments, chief among which is the need for rapid development of new applications to meet needs that were unknown before a disaster struck. This need for rapid application development has driven our desire to make it possible for application users to define what an application does. We plan in future work to enable on phone development of new applications by non-programmers. However, our current application takes a first step in that direction by allowing users to define the data model they wish to use.

1.4.3 The Application

The application consists of a Java MIDlet that can be deployed on Java-enabled smart phones and exchanged between MIDP 2.0 phones in the field. Using this platform means that Java enabled smart phones that are already in the field can be pressed into service as long as they meet up with a phone on which the application is already deployed. The application can also be deployed over the air if a cell network or Wireless Internet services are available.

The communication features can be used on smart phones that have the Java Bluetooth API defined in JSR 82 and the storage features can be used on phones that contain the Java Personal Information Management (PIM) API defined in JSR 75. The application has been tested on the emulator provided with the Sun Wireless Toolkit as well as between a Nokia N80 and a computer running the emulated code.

1.4.4 The New API

The API consists of a simple data storage system that can store Hashtables of Strings and exchange this data easily with other smart phones running the API. The API is considerably easier to use than the underlying JSR 82 and JSR 75 systems, hiding the implementation complexity from application designers and freeing them to focus on their application. The API is designed to allow other communication and storage implementations based on other technologies, allowing the API to be used in the future to exchange data over cell or wi-fi networks as well as use other storage systems available on other platforms.

Chapter 2

Related Work

There is a great deal of work being done in this area. Smart phone technology itself has been in development for a great deal of time now leading to a multitude of operating systems. There is also the Java Micro Edition which is designed to remove some of the challenges this diversity in operating systems creates. There are bundling systems designed to assist in building applications out of reusable components that we will also examine.

2.1 Smart Phone Technologies

2.1.1 Operating Systems

There are multiple smart phone operating systems on the market today including Symbian, Linux, Microsoft, Access, RIM, and others.[4] A recent Canalysis report [4] on smart phone market share showed that Symbian has a dominating 71.7 percent of the market; however, Linux is proving to be a popular platform with 14.3 percent of the market and Microsoft's platform is growing quickly with 265 percent growth between Q1 2006 and Q1 2007. Unfortunately, having all these operating systems means that there is fragmentation within the market as application designers have to pick an operating system that they wish to target or else deal with the complexities of porting to multiple operating systems. The only alternative to working directly with the operating system is Java technology, which can run on Symbian, Linux and other systems. However, it is important to note though that Microsoft's operating system does not support Java because of Microsoft's desire to push their own .NET development environment.

This lack of support means that there is no single technology that can be used to develop applications for all smart phone platforms.

2.1.2 J2ME: Java Micro Edition

Java Micro Edition (J2ME) is an implementation of the Java platform for mobile devices. Java offers unique advantages for developers because it uses a byte code format that is device independent. This format means that in theory a Java application can run on any smart phone operating system. There are however many issues with this in practice that prevent the “Write once, run everywhere” goal from being little more than a dream. First and foremost is the considerable fragmentation in the market in terms of support for the platform. There are multiple “Profiles” and versions of those profiles within J2ME allowing hardware manufacturers to define what features they will support. This causes considerable issues for application designers wishing to use Java because in the end they have to target a particular profile for their application.

Sun Microsystems has attempted to address this to some degree with the recent Java Technology for the Wireless Industry (JTWI, JSR 185) and the Mobility Services Architecture (MSA, JSR 248). These addresses some of the issues; however, the market is further fragmented by the various optional JSR packages that provide additional functionality such as access to the file system or bluetooth networking on the device. The situation is further hampered by the ongoing battle for control of the lucrative cell phone market by Microsoft, Sun, and others. Microsoft does not include Java with its Windows CE operating system. Thus as previously mentioned, Java runs only on Symbian-based and Linux based phones. Despite these issues, there is a growing market for Java software on phones, and Java remains the only really open solution for developing applications. As such, it is the most appropriate for academic research and thus has been selected for developing this application and API.

2.1.3 The Need for Serialization

Java faces some significant limitations on J2ME that do not exist in the Standard Edition (SE) version of Java. Most notable for our needs is the lack of object serialization that significantly simplifies the development of distributed applications using Remote Method Invocation (RMI) that is available with other versions of Java. This lack of support means that application imple-

menters are required to handle a much lower level of detail than would be required if serialization and RMI were already available on the smart phone platform. In order to enable smart phone platforms to easily develop smart phone applications, a serialization implementation would be invaluable. The Enterprise Edition (EE) version of Java has, in large part, been driven by Serialization and RMI. Lack of support for these features in J2ME is certainly a major issue for programmers who are used to working with versions of Java that have these features. The loss of the Serialization abstraction is a serious problem that needs to be addressed.

2.1.4 The Code-Signing Problem

Another major barrier to innovation using Java technology is the code-signing system specified in the Mobile Information Device Profile (MIDP) specification. The MIDP specification is intended to prevent malicious code from damaging the phone or its data by warning users not to install unsigned applications and asking them to confirm when the MIDlet accesses sensitive portions of the system. The protection offered by code-signing is a worthwhile goal but the security model is fundamentally broken because the roots of trust are not in the hands of the users but rather in the hands of device manufacturers and carriers.

The effect of the current specification of this system is to stifle innovation and shake down developers for cash and does not add a great deal of security. The issue is that untrusted applications that are not properly signed merely prompt the user to trust the application each time the application accesses a secured area. Because the action can still be allowed and users believe that the application needs access the system does not actually increase security on the device. The users are likely to enter yes to such prompts since they wish to use the application they have installed on the phone and either do not understand the security implications or willfully bypass them. The frequent prompts, however, are likely to annoy users to the point of not wishing to use applications that are not properly signed. The model even restricts user's abilities to install new root trust certificates, thus hampering developers abilities to even develop applications for the platform without a code-signing certificate to test with. Educated users are not free to decide what signing authorities they wish to trust.

This code-signing system stifles innovation by third party developers who cannot afford to pay for code signing certificates and does not benefit users very much at all. The manufactures limitations on root certifiat installation creates an artificial scarcity in certificate availability and thus artificially inflates the cost of certificates. Free certificates from organizations like CaCert cannot be installed on the phone and used to verify software and some root certificates that an educated user may not wish to trust cannot be removed either.

There are currently only three ways to get a signed MIDlet for Nokia phones, the current market leader. The first is to submit the MIDlet to the Java Verified program for testing, which requires that the MIDlet be fully developed. This testing costs hundreds of dollars per MIDlet and phone model and is thus not feasible for open source developers, hobbyist, and academic researchers. The other two options consist of purchasing code-signing certificates from either Verisign (at a cost in 2007 of \$499(US) per year) or Thawte, a slightly cheaper but still prohibitive cost in 2007 of \$299(US) per year. Furthermore because Verisign owns Thawte, there is no real competition in the certificate market. Because unsigned MIDlets can still be run but at great annoyance to the user this system is little more than a security certificate shake down foisted on developers which stifles innovation and takes control away from consumers.

2.2 Ibis: Distributed Communication API for Grids

Ibis[30, 29] is a distributed communication API targeted at grid applications written in Java. It is relevant to our current work in that it targets distributed communication using Java and has solved some of the same issues that we face. However, while our current work targets distributed communication and uses Java, it differs in several important respects. First and foremost, Ibis is a port-oriented API and uses connection oriented communication to achieve its goals. Although work is being done on a non-centralized version of the Ibis registry the join and leave semantics within Ibis are not appropriate for the applications currently targeting. Ibis operates at a lower level in many ways than the API we have developed in the sense that our API could be made to run on top of Ibis, a task that we intend to undertake soon.

Ibis does not currently support a disconnected model of communication, largely expecting that all nodes participating in the system are well connected. A connected model is certainly appro-

appropriate for a grid environment and similar environments where connectivity is largely assured but it cannot easily handle the kind of applications currently targeted by our research. Although a communication-oriented interface such as Ibis on smart phones makes sense for certain applications this is not the current focus of our work. However, Ibis also has a custom serialization interface that performs better than the native Sun implementation. In the future we hope to take advantage of this work to bring a serialization API to smart phones using Java to remove the String-based data limitation currently required by the new API.

2.3 OSGi: Open Services Gateway Initiative

OSGi is an “independent non-profit corporation comprised of technology innovators and developers and focused on the interoperability of applications and services based on its component integration platform[6].” It is a service-oriented, component-based system targeted at managing the software life-cycle and defines some basic services such as HTTP and logging. It is primarily targeted at embedded markets but has been widely adopted by the open source community and there are many projects based on the framework including the very popular Eclipse Integrated Development Environment.[1]

OSGi provides services for packing and delivering components using the Java runtime system. It facilitates installation and management of simple component “bundles.” These “bundles” are dynamically loaded to form applications of interconnected services. Services are defined via Java interfaces and stored in a service gateway. There may be multiple instances of services all of which fulfill the contract defined by the interface. The service gateway acts as a conduit between service providers and service users, giving service users access to an implementation of a particular service interface.

OSGi is not in any way a complete system. It is a low-level system designed to make software life-cycle management easy for developers but requires considerable complexity to be handled by those developers because of the low level of the API. It is no way usable by the average user that could be targeted for application development and modification. It is, however a good low level API for managing inter application functionality for mobile devices. As such we anticipate integrating our API into a bundle that is compatible with the OSGi initiative.

There are multiple open source implementations of the OSGi specification including Eclipse Equinox, Knopflerfish, Apache Felix, Osxa, and Oscar, all with varying levels of support of the core standard and optional bundles[9]. Nokia and Motorola have also combined and submitted a reference implementation of OSGi Revision 4 to the Java Community Process (JCP) as JSR 232 for use on mobile devices[12]. The main issue with using these technologies in smart phones is legacy support for the already-released CLDC 1.0 MIDP 2.0 devices in the market. Most existing devices lack support for OSGi. As such, it is not yet widely deployed enough to make it a viable platform for software today.

Although our work does not address the same issues as OSGi, we anticipate bundling our low level API as an OSGi compatible service in the future in order to more easily enable its use in OSGi-based applications.

2.4 ETC: Equip Component Toolkit

The Equip Component Toolkit (ETC)[13] is a toolkit for building applications on top of the EQUIP data space toolkit. This system is targeted towards making it easy for non-programmers to be involved in the creation of applications which is one of our long term goals as well, However, it relies on the concept of an “Installation Master,” a centralized component in each installation that manages the Linda style tuple space. It also replicates the tuple space onto every node for performance. Our system is similar in that it attempts to replicate all of the data onto all nodes, however, our system differs in that a node may decide which tuples it wishes to store. Replicating portions of the total tuple space is preferable to storing the entire tuple space because if the tuple space is very large then it may not fit within the constraints of a smart phone and also because it is useless to consume space on the device for data that will never be used by that device. Furthermore, security is provided via the use of a pre-shared secret key which is not appropriate for the kinds of applications we are targeting where it is desirable for a user to enter the system at any time without sharing an out of band secret. Although our system does not yet have support for security, we intend to add a PGP-style “Web of Trust” security model in the future to address the security issue, as we require a more distributed form of data security where data is authenticated as originating from a particular user but where data from unknown users can still be considered by the application.

Chapter 3

Katrina: A Case Study

3.1 Introduction

The recent Hurricane Katrina disaster in the United States was chosen as a case study in order to learn the needs for a disaster management application for several reasons. First, it is a particularly useful event for us to study because it is well known that the response to the emergency was inadequate; second, it is recent enough to take advantage of existing technologies and third; it is well documented. On September 15th, 2005, the United States House of Representatives created the “Select Bipartisan Committee to Investigate the Preparation for and Response to Hurricane Katrina,” which was charged with making a full and complete investigation into the issues surrounding the disaster. This committee generated a document detailing the issues that arose in the response, “Katrina: A Failure of Initiative[28],” which we used to study the disaster in order to understand how smart phones could be of assistance in such an event.

3.2 Technological Impact: Lack of Connectivity

The first thing to note about the disaster is that the hurricane knocked out many of the communications systems that state and local authorities and first responders rely upon to communicate with each other. These systems included not only the plain old telephone systems (POTS) but also the cell towers. Up to 2,000 cell towers were knocked out by the storm. While some cell towers lasted for some time on battery backups, in New Orleans, where there was considerable

flooding because of the levee breach, the towers lost primary power. Although the battery backup systems and generator systems ran out of power, the cell network died and could not be restored until the flooding was dealt with and crews could get fuel to the backup generators. Even the New Orleans Police Department's communication system failed and was inoperable for three days. This failure meant that they had to communicate on only two radio channels, forcing them to wait to communicate vital information.

Although many might think that the loss of cell towers implies that cell phones would be of little use in these areas, the emergence of phones with wireless internet connections and the expectation of the development of ad-hoc networking systems, as well as the ability to quickly deploy battery-based mesh networking equipment, may allow smart phones to have considerable utility in future disasters. It has been suggested that peer to peer networks comprised of cell phones could be used to allow (possibly limited) communication in such an event[21, 7]. It has also been suggested that manufacturers create "cell towers in a barrel", access points that could be battery-powered for considerably longer than a cell tower and could be easily dropped into extreme environments to build a network[25]. Manufacturers have also developed truck-based cell towers that can easily be deployed in place of land-based cell towers that have lost power in the event of emergencies[24, 27]. Thus our expectation is that the communication issues may be addressed by these new technologies. Even so there are applications that don't even require communication mediums to be of great utility.

The lack of communication mediums during the Katrina disaster compromised situational awareness and command and control and thus hindered the response. What is needed is a distributed communication system that does not rely on existing infrastructure and can be easily moved into place. Although the mesh and mobile cell towers may provide some solution to the problem the disadvantage of these systems is that they require action on the part of carriers to deploy them. However, smart phones themselves may offer a unique alternative in areas where such deployments cannot be quickly made via ad-hoc networking protocols. Such networks are often called Mobile ad-hoc networks to distinguish them from mesh networks composed of fixed location wireless routers[11].

Mobile ad-hoc networks that are based on smart phones offer an alternative in situations where mesh networks and cell towers are not in place. These networks can be made up entirely of smart phones which relay communication from one to another. Such networks could easily be seen to be of great utility in various disaster and rescue situations where cell towers and other infrastructure may not be available. Although these forms of networks are not fully developed there is already a IETF Work Group working on standardizing the technology in order to bring it to market[10]. As such it makes sense to consider developing a platform for creation of applications which can use such a network in order to speed their adoption once the technology is ready.

3.3 Possible Applications

Our analysis of the Katrina disaster has revealed a number of possible applications that could be of great utility in disaster scenarios. These include people finding, situation reporting, bus management, evacuation management, and shelter management. We will look briefly at each of these possible applications and their implications for the technology.

3.3.1 People Finder

In disaster situations individuals and groups often get separated from their families and friends. People often spend considerable effort trying to find lost loved ones during and after a disaster. What is needed is a comprehensive system to register both people whom somebody is looking for as well as people who are safe. The utility of this information is limited by the ability to access the data collected: smart phones could play a unique role in overcoming this limitation. By allowing access to the database in the field, people may be able to find the location of missing loved ones earlier than might otherwise be possible. By better enabling collection of information the database will be more complete earlier enhancing its utility. Thus an application that allowed individuals with phones to collect the required data as well as search the existing database would greatly enhance the ability of families to reunite, and possibly also find people with particular skills who could provide assistance in the disaster.

The fact that cell phones combine GPS, camera, and communication facilities makes them an ideal platform for gathering this kind of data. Individuals could easily snap a photo and use

the GPS to automatically associate the location. Workers could then enter the data about the person using the keypad on the phone and send it to the database if communication is available. If communication is absent, the phone could store the data until communication is restored; then it could send the collected data making room for more data collection. Individuals looking for relatives could also search the database, if connectivity is available, using their own phone; and display the photos on the screen and then use the GPS to help them go to the location where the individual was last seen as well as update their own location in the database based on the location provided by their phone. A system with these features would free disaster workers from having to do the search for individuals so they could attend to other matters of importance.

Furthermore, if volunteers with smart phones can register both their areas of expertise as well as their location via GPS and be contacted via the phone, they can then be more rapidly connected with an aid organization that is working in the area. The technology of the smart phone could allow the individual to do this registration from a remote location without having to have a worker enter the information, thus freeing up vital resources for other tasks.

3.3.2 Situation Reporting

Smart phones also offer other unique features that would have been of great assistance in other aspects of the disaster response than just basic communication. For example, the report of the levee breach in New Orleans was delivered to the White House some 12 hours before it was confirmed and action was delayed waiting for confirmation of the report. It was a Federal Emergency Management Agency (FEMA) worker who made the initial report, but the chaotic nature of information flowing into command and control in disaster situations created some doubt as to the veracity of the report. Had the worker had a smart phone with a camera, he would have been able to take a photo of the breach and send it to command and control, providing proof beyond a doubt that the report was true. This information may have had a dramatic effect on the government response to the disaster thus saving lives and minimizing suffering. A system that used the GPS features in addition could have provided the location the photo was taken, as well making deployment of forces to deal with the breach easier since command and control would already know the exact location that needed to be addressed. The data could be cryptographically signed by the phone to ensure the source and integrity of the information

reducing the need for independent confirmation.

3.3.3 Bus Management

Another situation for which smart phones would have been uniquely suited was the location and communication of buses in service to do evacuations. The evacuation of the Superdome in particular was greatly hindered by the fact that the buses were often flagged down by evacuees on the way to the Superdome and so were often full before they even got to the designated pickup point. The large number of buses proved to be a logistical nightmare since the location of the buses was not known to command and control and the fact that the buses were being filled before even getting to the Superdome was not known. There were reports of buses taking 4 hours to unload because of traffic jams at the unload points, but again this information was not available to command and control. Had they known they might have deployed forces to better manage traffic at the drop off points and deployed even more buses to effect the evacuation. The GPS and communication features of the smart phone platform could have been utilized to provide both location awareness and communication of that location to command and control so that the bus situation could have been better managed.

3.3.4 Evacuation Management

The evacuation before the hurricane made landfall was problematic as individuals often waited at designated pickup locations for long periods of time but none of the buses evacuating people knew where they were waiting and drove by several times without stopping. Many evacuees ended up walking to the Superdome in New Orleans because of the lack of communication about where they could be picked up. If buses had had GPS cell phones that could receive messages from evacuees with their location, the evacuation itself may have been more complete and saved considerable suffering and loss of life.

3.3.5 Shelter Management

There was also no comprehensive database of shelters. Louisiana had some 563 shelters set up by the state or American Red Cross with another 10 special needs shelters but it is likely that there were many private shelters, available from churches and other community groups making

the total number of shelters much higher. Despite the large number of shelters individuals had a very hard time getting information about where shelters were and certainly had no way to know if a shelter was already at capacity or was likely to be full by the time people arrived. For example, the Superdome greatly exceeded capacity as it was the only official shelter of last resort for New Orleans. The Civic Center was opened out of necessity, but officials in command and control positions were unaware that this had happened for several days. When the city flooded, “the clover-leaf,” a conjunction of highway interconnects, became a shelter out of necessity so that people could escape the rising waters. Because of the unexpected need for resources at this location some resources intended for the official shelter had to be diverted. A service that alerted individuals to the location and available capacity of the shelters would help them to get assistance. If the system also allowed the shelters to get a feeling for the number of people that would arrive then command and control could better position required supplies for the shelters as well as alert them to the need to open additional shelters. Furthermore, a system to alert command and control to new ad-hoc shelters would allow for better resource planning and distribution. The data and input for this application needs to be handled in a distributed fashion as new shelters may be opened at any time during the disaster and ad-hoc shelters may arise out of necessity.

3.3.6 Summary

To summarize the possible applications we have discovered in our analysis we provide Table 3.1 which shows each problem and how smart phones could be used as part of a solution.

3.4 Implications for Technology

The examination of the Katrina disaster has revealed several possible applications that would greatly assist relief efforts in a disaster. Although these applications are interesting in and of themselves, they are not the most important lessons to take away from this analysis. Perhaps chief among the knowledge gained is that the need for specific applications was not clear until the disaster started. Thus, an emphasis needs to be placed on a development environment in which applications for smart phones can be implemented and deployed extremely quickly.

Problem	Smart Phone Solution
Individuals get separated. Volunteers announce themselves everywhere.	Smart phones collect data on lost and found people and volunteers in the field. Data is distributed as connectivity allows.
Information to Command and Control needs confirmation of origin.	Images, text, and location data taken on phone in the field can be cryptographically signed to ensure authenticity and origin and sent to command and control as connectivity allows.
Location of buses involved in evacuation are unknown to command and control.	Drivers with GPS enabled smart phones can pinpoint bus locations via the smart phones.
Evacuees in need of pickup are hard to locate and count with impacts on management of required resources.	Evacuees with GPS enabled smart phones can notify authorities of their location allowing them to be evacuated efficiently and evacuation resources to be better managed.
Shelter locations and capacities are unknown to individuals. Shelters have no feeling for how many individuals are coming.	Evacuees with smart phones could download a list of shelters and notify shelters that they are coming to help the shelter estimate required resources.

Table 3.1: Problems and Smart Phone Solutions

We can also clearly see that the combination of location information and data input, including photographic and/or video data, produces a compelling combination for smart phone as data entry device. The bus application also demonstrates the value of smart phones as sensor nodes providing location information of distributed assets. Although there are other devices that may contain these features the smart phone offers a ready-made, relatively cheap platform that may already be deployed widely in the disaster area, just waiting to be used. Thus it is important that it be easy to combine location information with other information and communicate that data easily for use by command and control and service providers, highlighting again the need for rapid deployment tools in order to be able to press available smart phones into service.

The possible use of mobile ad-hoc networking and/or the use of mesh networking techniques which can run with lower more distributed power than a cell tower and can thus replace lost cell towers or even be deployed in areas where cell towers are not available are also clearly very useful and making programming for the use of these features easy is obviously important.

Chapter 4

Sahana: Web-Based Disaster Management

Sahana[3][2] is a free, open source application for disaster management that has been in development since 2004. Originally conceived during the 2004 Sri Lanka tsunami, Sahana development has been led by the Lanka Software Foundation. Sahana has a rich set of tools for managing various aspects of a large-scale disaster. It attempts to address several problems common to disasters including reuniting families, coordinating aid, locating temporary camps and shelters and managing requests and offers of assistance. It is designed to be deployed on platforms from a laptop to a large cluster of servers depending on the needs of the disaster and available resources. In disasters, power is often a very scarce resource. Sahana has been designed to run on a single laptop powered by a 135 Watt solar panel until power can be restored. Data from multiple such laptop deployments can later be integrated into a larger system when power and internet connectivity can be restored. Sahana was built from the ground up with usability, adaptability, and resilience in mind in order to make it applicable to as many disasters as possible. It is designed as a web-based application and some might argue that this is the only interface required.

4.1 Sahana Modules

Sahana is organized around the concept of modules where each module manages some aspect of the disaster. This organization allows Sahana to adapt to the needs of various disasters.

Although the modular nature of the architecture is desirable, the coupling between modules within Sahana is difficult to handle in a clean way. Nonetheless, we have examined a number of these modules in order to understand the role that cell phones could play in this model.

4.1.1 Sahana Missing Person Registry

The missing person registry is designed to help people find each other during and after a large disaster. It captures data about individuals who are missing or found including data such as what they were wearing when they were last seen, hair and eye color, location where they were found, as well as photographs of the individual and relationship information between individuals. The relationship in particular is powerful because it allows the system to unite two members of the same family through the relationship tree.

4.1.2 Sahana Volunteer Registry

The volunteer registry portion of Sahana is designed to keep track of all volunteers who are working in a given disaster region. It is designed to capture the services a particular volunteer is offering and where they are offering these services are being offered. This allows not only an idea of how services and support are being distributed in the region but also allows command and control centers to determine where aid services are not being provided. By registering ad-hoc volunteers it is possible to take better advantage of particular skills offered by volunteers during the disaster.

4.1.3 Sahana Organization Registry

The organization registry is designed to help government agencies and non-government organizations (NGOs) as well as the general civil society to coordinate aid in the event of a disaster. Lack of coordination between these groups can cause problems with too much aid going to some areas while others get none, as well as issues with supply chain management where a route to a remote shelter becomes jammed with too many aid organizations trying to get there. It is designed to capture both the places where organizations are active as well as what kind of aid they are providing. It is also designed to register ad-hoc volunteers who can then be contacted by organizations in need of volunteer labor. This feature in particular might benefit from the

use of smart phones.

4.1.4 Sahana Shelter Registry

As we saw in the analysis of Katrina, there are often many shelters that spring up to provide needed services to disaster victims. The distributed and often ad-hoc nature of shelter creation poses unique challenges that the Sahana Shelter Registry is designed to address. The registry is designed to capture data about these shelters including location, facilities, and size and also to manage the needs of the facilities and provide a geospatial overview of the shelter situation. Again this application could easily benefit from the location data provided by GPS-enabled smart phones. Efforts are already underway to develop applications for existing GPS receivers, and the smart phone could easily be used to not only input GPS data but also needed supplies and update capacity. However, one distinct advantage that laptops may not provide is distribution of shelter information to disaster victims with smart phones as we mentioned before in the analysis of Katrina.

4.1.5 Sahana Request/Aid Management System

Often after a disaster there is a huge pledge of aid from various organizations. Distribution of that aid is, however, a huge problem as many of the organizations pledging aid have no supply chain with which to distribute it and requests for aid often end up going to an organization that does not offer the needed assistance while some other might. The Request/Aid Management System is designed to deal with this problem by cataloging both the pledges of aid and of requests. This application again could benefit from the synthesis of location information and aid requests, making it much easier to request aid for a specific location. Although this may not be as compelling as some of the other applications, the general principles of deployment and availability of the phone as input device of course still apply here. The location data available on the cell phone could also be used, as in the bus-tracking system suggested in our analysis of Katrina, for supply chain management to track if supply trucks are actually getting to their intended destination and could be used to provide direction information to workers who may not be familiar with the area.

4.1.6 Sahana: Possibilities for Smart Phone Integration

We would argue that there are many areas where smart phones could easily be used as a primary interface tool for the Sahana platform and in fact offer distinct advantages over a more traditional laptop/browser deployment. Even smart phone-based browsers will not offer the ability to take advantage of the rich set of features of the smart phone easily and seamlessly. Furthermore, the cost of the hardware is reason enough to consider a smart phone deployment. A typical smart phone is far cheaper than an average laptop. Distribution of hardware is also a major factor to consider. If the promise of the smart phone platform plays out as the pundits predict, it is much more likely that people in a disaster area will have smart phones than laptops. Even those who do have laptops are unlikely to have laptops with cameras and even fewer are likely to have GPS features on their laptops.

Furthermore, in an emergency it is probable that individuals will take their phone with them as they evacuate and will leave their laptops behind. Individuals often have their phones on their persons and the communication features give compelling reasons to do so. Individuals are much more likely to leave their bulky heavy laptop behind as they rush to safety. Thus we believe that the smart phone platform may prove to be much more prevalent and ready for use within a disaster zone than laptops will and can thus be pressed into service much more quickly and in far greater numbers.

Another factor to consider is available power. A laptop requires a great deal more power than a smart phone and electricity is often one of the first services to be lost in a disaster situation. The average laptop has a battery life in the range of a few hours and consumes an order of magnitude more power than a smart phone. The average smart phone has at least as much active talk time and a great deal more time if a cell signal is not being broadcast, a power hungry operation. They also consume considerably less power for the same amount of running time, allowing many more to be charged with the same solar panel it would take to run a single laptop. There are even hand-crank cell phone chargers available which can provide enough power to charge the phone even while in use, allowing them to be used easily where power is not available at all. The combination of longer battery power, richer features, and likely existing distribution in the disaster area makes them a compelling input device for use in disaster situations. In order

to understand how the features of smart phones could assist the existing Sahana platform, we analyzed each application of the Sahana platform in turn.

The Sahana platform clearly offers several applications where the unique features of the smart phone would ease data input in terms of GPS, photo, and even text data. The use of smart phones already in the disaster area provides a compelling reason to develop such applications, but questions of deployment are again raised. What is needed is a way to combine the rich features of the smart phone like location and photo data with more traditional text data and deploy such an application quickly onto smart phones already in the disaster zone.

4.2 Criticisms of Sahana

It is important to note that Sahana is in essence a centralized solution to data collection. Although interface applications for smart phones offer the opportunity to improve data collection, the centralized nature of the Sahana platform necessitates connectivity to the central node. Although Sahana can operate without a complete network by making data transfers via disks when possible this process is far from completely developed and is likely to cause large problems with data consistency. Even if data consistency can be handled properly, a majority of the data will not be available at all sites and synchronization must be managed in an ad-hoc way, making it unlikely that it will be consistently done correctly.

Furthermore Sahana often requires modifications to make it suitable for a particular disaster. Although this may be acceptable in a centralized system, having distributed interface applications makes this considerably more challenging and necessitates code changes to both Sahana itself and to the interface applications. Providing a distributed interface with the current implementation of Sahana is further complicated by the implementation of Sahana as a PHP/SQL application. There is currently no way to easily modify both the schema and the interface and doing so requires considerably more technical expertise than is likely to be possessed by the average aid worker. The demonstrated need of rapid application development in disaster scenarios is not very well met by the current implementation.

Chapter 5

People Finder Application

5.1 The Approach

What is really needed to address the issues raised so far is a system designed from the ground up to work in a completely distributed fashion that can be easily modified in terms of the data that is collected. Thus it is desirable to allow the smart phone to collect data in the field such as descriptions of individuals in shelters and then exchange that data with other phones when communication is possible and with command and control centers when a phone can establish communication with such a center. In this way data can be diffused throughout the entire disaster area. This communication might be done via bluetooth or wireless internet from the phone. The desire to be able to use Bluetooth, cell, and wireless internet once again highlights the need for an API that can use multiple network services to accomplish the same task.

An application designed in this way offers significant advantages over the centralized model of Sahana. For example searches against a volunteer database can be done on each phone against the subset of data it has collected and gathered through data exchanges with other phones. Then, locating a volunteer with a particular skill-set does not have to wait until communication is available to the central server; rather it can occur along with the data collection process. As more data is collected and exchanged between the phones, each phone can build up a more complete database that can be searched locally. The search done will be against an incomplete database, but as connectivity is restored the data exchanges between phones will bring new data

to each phone allowing search of a larger data set. A search against incomplete data that results in a hit is certainly more valuable than no search at all. Searches can be repeated when more data is collected as volunteers are likely to be searched for repeatedly until the need is met. In this way the collected data becomes useful in a much shorter time than is possible with a centralized system where information utility is limited by connectivity.

The idea of storing a more complete copy of all the data on each phone raises considerations of storage space and power. We argue that there are smart phones today with expandable memory slots that can hold gigabytes worth of data. This storage is certainly a large enough data store for purely text based data and can hold thousands of low resolution photos. The exchange of data between phones could be limited based on the kind of data the particular cell phone user is interested in providing access to. A worker at a shelter may care about only lost people and shelter locations and may not gather data about aid requests at all but might insert an aid request into the network of data based on needs. In this way the application is much more adaptable while limiting data storage to data that will be useful on the phone in question.

With this kind of system, phones can enable rapid data gathering and distribution in a distributed fashion. Furthermore, the limited power available to a centralized laptop in the disaster area can be extended. The system could also use an outside command and control center where power is available and simply take a phone with all the collected data to this location and use the same system to exchange the data with the command and control system.

The exchange of data between phones in this manner raises the issue of multiple copies of the data being present and the need to be able to handle merging of these multiple copies. We solve this issue in our current application by giving each data item a globally unique ID and only allowing data to be written once. This is not the most flexible system, but it eliminates the need for a version subsystem that can handle multiple versions of the same data item and can reconcile the versions into a single copy. Future work will address this issue.

5.2 Design

The design of our application is relatively simple. The application allows a user to define data types that consist of an array of string keys and basic type for each key. We pre-load this system with the data types for volunteers as well as lost and found individuals. We loosely based the data to collect for each of these three types of people on the data that Sahana currently tracks for these individuals. The advantage of our system over Sahana is that the types need not be fixed to these three types. Users are free to add new types to the system. The user can then add an entry of a given type to the database. At any point the user can list the entries of a particular type. While we have currently limited the types to write only future versions of the application may allow modification of a particular type if new data about that type should be collected. This on phone creation and modification of types makes a much more flexible system than is currently available.

We approached the design from a Graphical User Interface (GUI) perspective by first designing the screens that make up the system and the connection between them using the Netbeans IDE enhanced with the Netbeans Mobility pack. The development environment provided by Netbeans provides a graphical editor for creating applications for mobile phones on Java. These applications are called MIDlets (`javax.microedition.midlet.MIDlet`), the analog of a Java Applet for cell phones. This development environment allows rapid development of the various on-screen components but still leaves a great deal of code to fill in the functionality of the various screens.

MIDlets provide a way for the mobile device to manage the lifetime of the application. The API requires that applications implement a defined interface for construction of the application, pausing the application and termination. Although the Visual MIDlet designer from Netbeans is quite powerful, it is not a tool that is targeted at allowing ordinary individuals to create applications. A programmer is still required to fill in the connections between the graphical user interface and the underlying system which the graphical interface is driving. Thus our application is specifically designed to allow users to define new types and specify the data that they wish to collect about that type. This design is a first step towards creating a system that allows non-programmers to create applications on the fly.

On the initial screen the user is presented with a Form (`javax.microedition.lcdui.Form`) that contains a splash image, a gauge (`javax.microedition.lcdui.Gauge`) that runs continuously and a text string that changes as the application initializes. If there is an error while the application is initializing, then an alert message is displayed and the user is prompted to exit the application. If initialization is successful, the application moves on to displaying a list of types. The visual midlet designer in Netbeans does not have a way of representing an error form which is displayed when a condition occurs nor does it have a way of representing a form.

The List (`javax.microedition.lcdui.List`) displays the known data types and allows the user to scroll through the types and select one. Choosing one brings up the screen which lists entries of that type. It also has several on screen commands for creating new types, turning network listening on and off and initiating a transfer of data and exiting the application. Unfortunately the Netbeans designer does not allow the same command to be assigned to two screens, which means that, even though Java will allow a single command to be shared between screens the designer is unable to create such an application. Because of this limitation an application has more instances of commands than are needed and also means that the logic to deal with commands is overly complex. If two commands require the same logic, there is no easy way to handle this using the Visual MIDlet designer.

The screen to create a new data type is the most complicated screen out of our application because it is the portion of that application which gives users the power to modify the data that the application creates. It contains a text area to specify the data type, a table to display the added data fields and fields to add a new field to the table and remove existing fields and on screen commands to save the fields or cancel type creation all together.

The screen to display a particular piece of data has to be created on the fly based on the fields in the particular data type. Thus the graphical components that are required cannot be represented by the screen designer. This deficiency demonstrates one of the limitations of the Visual MIDlet designer in that it has no easy way to express dynamically created portions of the user interface.

Although the Visual MIDlet designer clearly has quite a bit of power to assist developers in creating applications, it gains that power by giving up some flexibility that leads to extraneous code and logic. These deficiencies are an issue for the smart phone platform because the devices are resource constrained. Additional commands and screens require additional storage for the added code, additional memory to load them, and additional processing power to execute them on the phone. Although Visual MIDlet Designer allowed us to quickly sketch out the interface the inability to specify visual components that needed to be created on the fly prevented us from designing the applications GUI entirely with the visual designer.

The final design of the screen flow can be seen in Figure 5.1. Note that not all flows are represented by lines because of the limitations of the Visual Midlet designer.

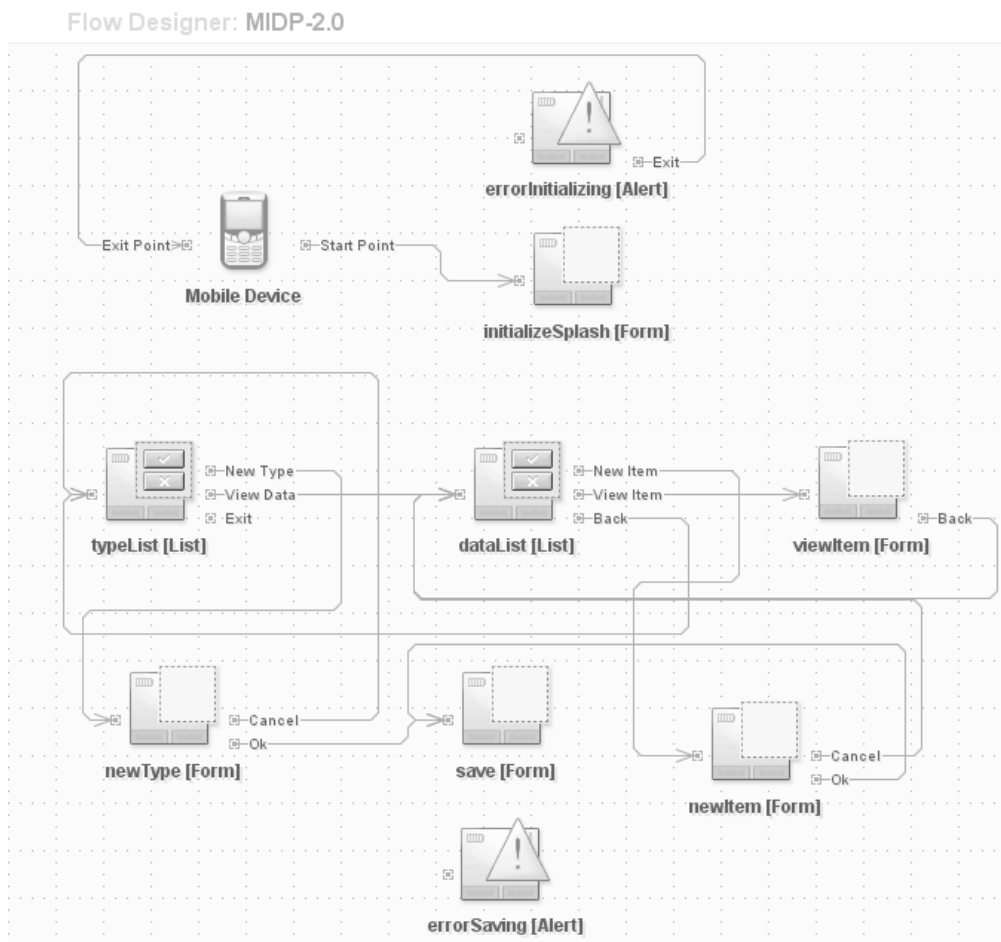


Figure 5.1: Screen Flow

5.3 Implementation

Our goal was to implement as simple an application as possible while still providing the functionality to fill the demand determined by our analysis of the disaster management interface. Having designed the GUI that drives our application in the design phase, we next moved to implementing the actual work that the application performed. This step required balancing the needs of the application with the desire to simultaneously create an API to hide the low-level tasks. Because our application is primarily data oriented, we first implemented the data model that our API would offer to the application based on our understanding of what the application required. This implementation is discussed more in Chapter 5.

With the data API in place we began to fill in the places in our application where it makes use of this API. We choose to store the data for the types that are supported in our storage layer as well as the data for each type. The ability to store application and user oriented data in the same system demonstrates the flexibility of the system to work as an arbitrary data store. Each type gets a type item that is excluded from the list of data items when they are displayed, but these items can be transferred by the application when exchanging types. This means that a type which is unknown to a particular phone can be added to that phone via the usual transfer mechanism.

Because the API is so simple the application requires a surprisingly small amount of code to deal with any operation. A great deal of the code of the final application is dedicated to catching the `IOException` that the API can throw when there is a problem storing the data.

5.4 Analysis

We tested our application using the emulator that comes with the Sun Wireless Toolkit. The emulator allows us to simulate more than one phone and the data connections between them. This simulation allowed us to verify that our code should work properly on remote devices. We were able to verify all aspects of our application and our API using the emulators. We did, however, run into issues testing some aspects of the lower level API on our real target device, a Nokia N80. The code that worked fine on the emulator did not work on the real target device. This failure

to work properly on a target device highlights one of the major problems with the emulators; they are not true emulators because they do not exactly mimic the behavior of the application on the device. This deficiency is a major obstacle to the “write once, run everywhere” system that developers would like to use. We will discuss the trouble we encountered more completely in the section on the lower-level API.

Benefits

The benefits of our application over existing systems is clear in that it runs on a platform which is likely to be widely deployed in a disaster area. It allows users to participate in designing the data model for the application and also allows them to track new kinds of data quickly and easily. It enables a new form of distributed application where portions of the total data are stored on each device with the ability for each device to build up a more complete view of all the data in the field when networking is possible and to ferry that data to a command and control center as required and to ferry data from command and control back into the field to be distributed to devices there.

5.5 Limitations

Our application does not currently support the modification of a type once entered because of current restrictions in the API we have developed to handle the data storage at this time. We intend to expand the API to allow for such modifications in the future and expand the application to take advantage of these features when they are available. It also does not currently support complex type validations as all data is stored as a String within the data storage system again due to restrictions in the lower level API we have developed but we have plans to address this issue in the API in future work and expand the system to store additional types. See the Related Work section in the Conclusion for more information on how we intend to address these limitations.

Chapter 6

RAVEN: A New API for Distributed Applications

Our New API is based on the concept of diffusion of data throughout a periodically connected network with the idea that nodes can choose which data they are interested in collecting. As such we require a method of exchanging data between the nodes. We selected Bluetooth as the network of choice because the disaster management application we targeted building assumes that the cell network is unavailable. We selected Bluetooth over Wireless Internet connections because bluetooth is much more widely supported on mobile devices than Wireless Internet services, which appear on only a handful of devices. Thus, we required the Bluetooth API for Java (JSR 82) in order to be able to make the data exchanges.

Our API requires the Connection Limited Device Configuration (CLDC 1.0) and Mobile Information Device Platform (MIDP 2.0) support as well as support for the Java bluetooth API (JSR 82) to do the data exchange. Data persistence is done using the Java Personal Information Management API (JSR 75). Both of these APIs are available on most modern smart phones including our test phone — a Nokia N80.

6.1 Design

The design of our API was driven primarily by the needs of our application as discussed in Chapter 4; however, we paid special attention to keeping the API general enough that it could be taken off the mobile phone platform and used on desktop computers. We also worked hard to ensure that the API was general enough to be built on top of various different communication and storage systems. We next discuss in turn both the storage API and the Communication API that we have designed.

6.2 Storage API

Java ME has only two general-purpose data storage types in the `java.util` package. These are the `Hashtable` (`java.util.Hashtable`) and the `Vector` (`java.util.Vector`) classes. These classes do not have the usual interface versions in the form of `Map` (`java.util.Map`) and `List` (`java.util.List`), respectively, that are common with the standard edition of Java.¹ This means that the interface for our storage API is limited to these two generic types. While we would prefer to have defined the interface to the storage system in terms of Interfaces available on Java Standard Edition so that Standard Edition Implementations would be free to use more complex types available in the `java.util` package this simply was not possible due to these limitations.

Furthermore, as discussed in Chapter 3, Java ME lacks serialization support. In order to address this issue, we chose to restrict our API to only handling Hashtables with Strings in them for both keys and values. While we could have implemented a system to at least allow all of the primitive data [`Integer` (`java.lang.Integer`), `Long` (`java.lang.Long`), `Boolean` (`java.lang.Boolean`) etc.] using the `instanceof` operator, we felt that using this implementation was still little more than a hack. The real need is to be able to store arbitrary objects and exchange them, which requires serialization. We intend to address this shortcoming of the Java ME platform in future work instead of adding such a hack to our existing API. Given that user input always comes from the `TextField` (`javax.microedition.lcdui.TextField`) as a `String` anyway, we felt that this was a reasonable restriction until we can add serialization support and thus support objects of any

¹Presumably the lack of interface is simply because they would serve no purpose with only one available implementation on the phone but add to both code space requirements and complicate `instanceof` checks. The code space and run-time speed was a bigger issue when Java ME was first designed but could probably be dropped on modern phones to allow APIs such as ours to work more easily on both ME and SE editions of Java.

kind.

In order to keep the communication API from having to exchange objects that are already on a particular phone, we decided to have the storage API assign a globally unique ID to each data object when it is added to the system. This ID is created in collaboration with the communication API because communication systems often already have a unique ID such as an address. This ID is combined with a timestamp for the time the object is added and a serialized ID for the storage subsystem instance. Although the ID may not be globally unique the probability of collision is extremely low using this system since the address is usually intended to be unique and the chance that two instances add new objects with the same timestamp and instance ID is extremely low.

We considered creating an object wrapper to store this data but rejected this in order to keep object construction and memory usage down for the current simple system. Instead, we chose to disallow the application from adding keys to the data hash that begin with “raven.reserved.” For our current test application this limitation is sufficient; however, as the system becomes more complex, we may revisit this decision and create a simple object wrapper that can hide the version and identification information from the system. For the moment, the current API is sufficient and the added complexity of a wrapper object is not required.

We wanted the communication system to be able to exchange subsets of the total data so users can store only data on their phone that might be of use. Of course, users are still able to accept all data if they wish to ferry data to a command and control center or they are interested in data of all types. In order to effect this, we decided to introduce the concept of hierarchical types to the API. That is, each data object that is put into the system has an associated type. The hierarchy of types is created using a dotted notation to split a particular type up. For example, with our people-finder application, users might register that they are interested in any data object in the “people” category, which would include both subcategories of “people.lost” and “people.found.”

This design of the system means an application can add a Hashtable to the system while passing in a String that specifies the type. The API then places the data into the type and each of the super types of that type. So using an example from our application, if we added a Hashtable

with the type “people.lost,” a fetch of either the type “people” and the type “people.lost” will return the added data item among the data returned. The type system provides an easy way for users to filter both the data they want to see and the data they want to exchange based on these types.

The API also provides methods for dealing directly with the types in the system. An application is free to add types to the system without adding a data item in order to prime the type system if required. The application may also retrieve the types that have been added to the system.

When a data item is added a value is automatically added to the hash in the reserved namespace “raven.reserved” with the unique ID of the data item. This ID is used as a key into the system to retrieve the item later. This ID is also returned to the application when the data item is added so that the application can track it if it needs to.

When the user asks for the content of a particular type, the system returns a Vector of these IDs. The user can then use these types to fetch the Hashtable associated with the given ID from the system. It should be noted that the semantics of what happens when the user changes the returned Hashtable are undefined. An implementation may pass back the only copy of the Hashtable and modifications to that hash are backed by the actual Hashtable stored by the system or an implementation may pass back a copy of the original data Hashtable. Similarly, whether an implementation of the API reflects changes made to the Hashtable after it has been added to the system is also undefined. The design allows an implementation the freedom to either store the data in memory or write it out to long term storage, or a combination thereof as is appropriate.

The storage system we have created is extremely simple, offering only 5 calls that are very similar to the existing Hashtable interface. These are the following:

```
/**
 * Returns a Vector of IDs as Strings that are in the specified type.
 **/
Vector get(String type) throws IOException;
```

```
/**
 * Returns the data Hashtable for the given ID key.
 */
Hashtable getId(String id) throws IOException;

/**
 * Adds the specified type to type storage
 */
void putType(String type) throws IOException;

/**
 * Returns the subtypes within a given type. (Use null to get all types.)
 */
Vector getTypes(String type) throws IOException;

/**
 * Puts the data into a particular type returning the ID for the item.
 */
String put(String type, Hashtable data);
```

The last required part of the storage API is a way to get some required information from an application making use of it. In particular, we would like the system to be able to store the data for multiple applications — all of which make use of it — on the same phone without requiring that applications intermingle the data. We also need to allow applications to choose where to store the data on the filing system, in particular if there are different storage roots as is common on smart phones with removable memory cards and Windows-based PCs. We get this information using call backs to the application, which we will discuss in the Callback API section below.

6.3 Communication API

The goal for the communication subsystem is to make it as easy and seamless as possible for users to exchange the data they have in the storage API. However we have to consider the power requirements of the platform we are targeting. Because, we would like the system to be able to use Bluetooth and/or Wireless Internet we need to ensure that the API is designed in such a way as to not require the radio to be on all the time listening for a connection. Thus, the API requires calls to turn on and off listening for a connection. Once the system is listening it will accept incoming data exchange connections.

In order to exchange information with another system that is listening the application needs to find a system to exchange data with. The application simply requests the API to perform a search for suitable systems. The results of this search are returned to the application via a callback discussed below. The application selects one of the remote systems found after making the search call and calls into the system requesting a transfer and telling the system which types it is willing to send to the remote system.

When the system accepts a connection, the two systems exchange the types they are interested in receiving and match that up with the types the other party is willing to send and then exchange the data they have in the subset of matching types. This data is automatically added to the storage system for retrieval. This system hides from the application designer the complexities of this exchange entirely and makes the new data available through the existing storage API.

The design makes the inbound interface for the communication API extremely simple consisting of only three calls:

```
/**
 * Turn on or off searching
 */
void listen(boolean listen);

/**
 * Request a search
 */
void requestSearch();

/**
 * Request a transfer with another party.
 */
void requestTransfer(RemoteRaven raven, String[] sendableTypes);
```

There is some additional complexity to this interface that is handled by the callback system discussed next.

6.4 Callback API

Our API operates largely asynchronously which means that the API needs to be able to make calls back into the application. Callbacks are handled by forcing the application to provide a

listener to handle the callbacks when it initializes the system. Java APIs often allow multiple listeners to listen for the same events, but our listener API is not strictly an event system because it provides data to the API. Thus, we have restricted our API to a single-listener set at initialization of the system.

The callback interface has some methods that are dedicated to the storage system and some methods that are dedicated to the communication API.

6.4.1 Generic Methods

Because our API is largely targeted at applications that are designed to be used with a GUI and large parts of it run asynchronously, it is important to have a way to give feedback to the user about what the subsystem might be doing. To accomplish this we have added a generic method to the application API that can be called with activity messages. This can be useful to inform the user how the search is proceeding when running a long running search that consumes all the resources on the device. This one method looks as follows:

```
/**
 * Send an informational message to the application.
 */
void informActivity(String message);
```

6.4.2 Storage Methods

There are three storage callback methods in the communication API. The first allows the storage system to get an ID for the application in order to allow the storage system to store data for each application in different places. The second method allows the storage system to ask the application what to use as the storage root for long-term storage. The last callback is used during initialization to let the application know that there was an error loading the data from stable storage at initialization. These three calls as follows:

```
/**
 * Return the id for the application
 */
String getApplicationId();
```

```

/**
 * Select a storage system
 */
String requestStoragePath(Vector roots);

/**
 * Inform the application of an error loading the data
 */
void informLoadError(String error);

```

6.4.3 Communication Methods

Because the communication system is largely asynchronous, the number of callbacks it requires is considerably higher; however, they are far from onerous for the user of the API. These methods are grouped based on the call into the system that triggers the callback and are illustrated in the following three diagrams.

In Figure 6.1 we see the flow of calls related to the listen call which include the callbacks `informTransferError()`, `transferComplete()`, `getTypePermissions()`, and `getTypeInterests()`.

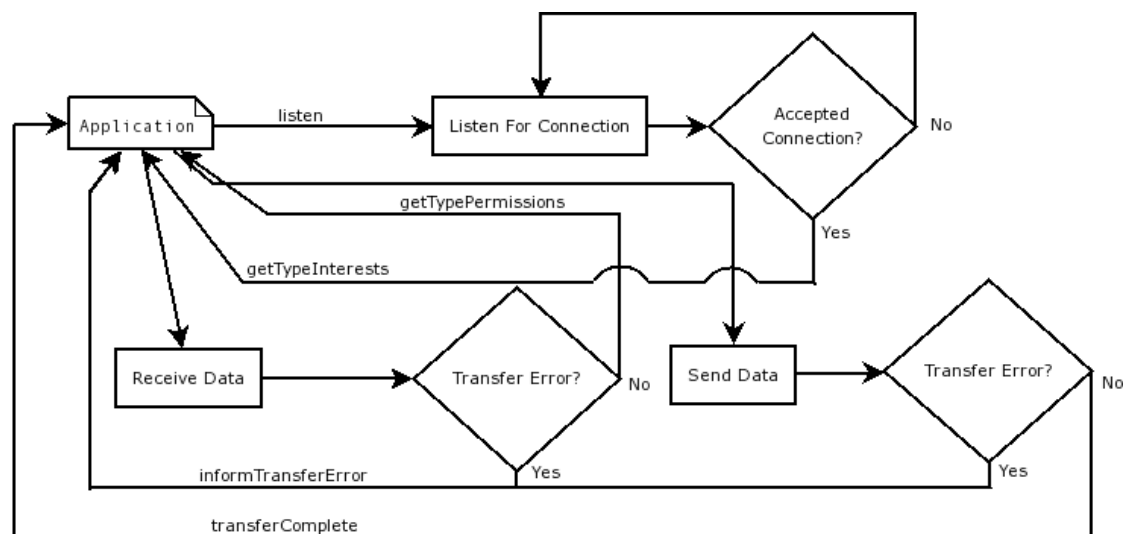


Figure 6.1: `listen()` Flow Chart

In Figure 6.2 we see the flow of calls related to the `requestSearch()` call made by the application which include the callbacks `informSearchError()` and `searchResults()`. Note that the results of this call are required in order to initiate with the `requestTransfer()` call detailed in diagram 6.3.

In Figure 6.3 we see the flow of calls related to the requestTransfer() call made by the application which include the callbacks informTransferError(), transferComplete(), and getTypeInterests(). Note that there is overlap in these callbacks with those required by the listen() flow reducing the number of callback methods that an application must implement.

The callback methods are as follows:

```

/**
 * Inform the application of an error during a search
 */
void informSearchError(String message);

/**
 * Inform the application of search results
 */
void searchResults(Vector records);

/**
 * Inform the application the transfer with the given id is complete

```

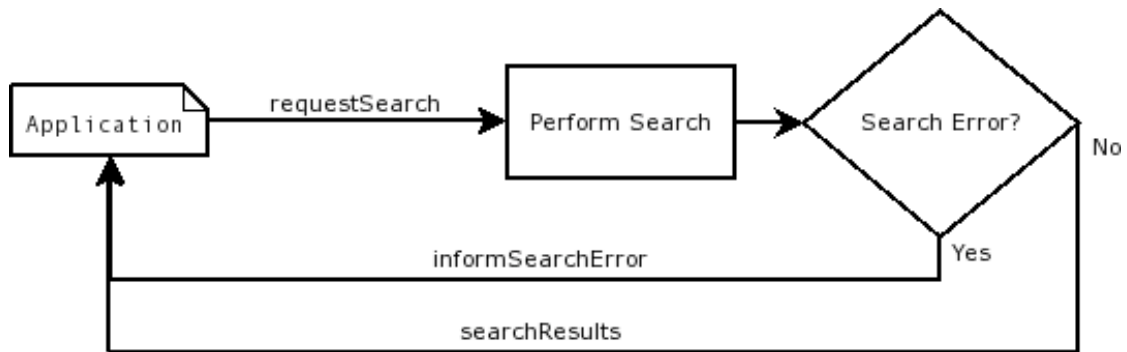


Figure 6.2: requestSearch() Flow Chart

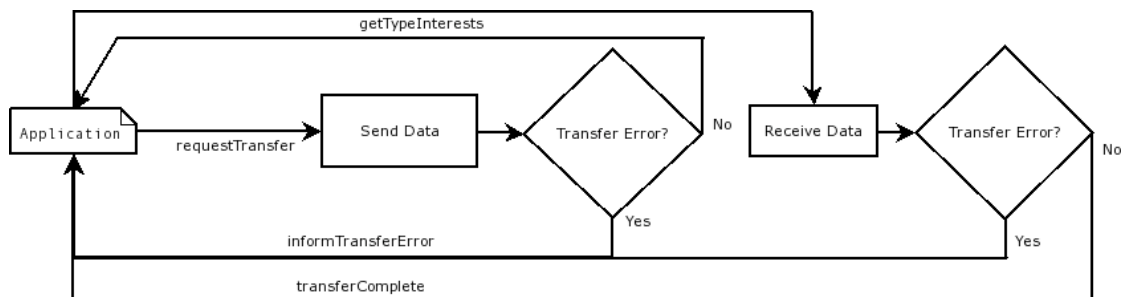


Figure 6.3: requestTransfer() Flow Chart

```
*/
void transferComplete(int transId);

/**
 * Inform the application of an error during a transfer
 */
void informTransferError(int transId, String message);

/**
 * Request the types that the application is interested in receiving
 */
Vector getTypeInterests();

/**
 * Request the types that the application is willing to send to a remote system
 */
Vector getTypePermissions(RemoteRaven remoteSystem);
```

6.5 Implementation

The implementation of both the storage system and the communication system followed directly from the design of the API. The ease of implementation encountered in this process is indicative of the simplicity of the overall design.

6.5.1 Storage System

We have implemented two versions of the storage API. The first version was implemented using only in memory storage. The in memory system is relatively simple because it does not store data off to long-term storage. Because of the lack of long-term storage the quantity of the data to be stored is limited by the amount of memory available to the application on the device and that the data will be lost as soon as the application is terminated. We have also implemented a version on top of the JSR 75 storage API for long term storage of data on the phone.

6.5.2 In Memory System

The initial implementation of the storage API was designed to allow us to move on to the communication system, which is the portion that we are most interested in. As such we implemented a very simple version of the API that uses three Hashtables to store all the data. One Hashtable stores the types within a particular super-type using a Vector for each type. The

second Hashtable stores the type to id Hashtable storing a Vector for each type filled with the ids of all data elements in that type, and the last Hashtable stores the ID to data mapping. These three Hashtables are all that is required to implement the API.

6.5.3 JSR 75 System

The second implementation of the storage system was added after getting communication working in order to effect long term storage on the phone. This version uses the filing system on the device to store the equivalent structure as the Memory System. It uses names of files as keys on the filing system and stores the contents of the Vector or Hashtable that would be in the in memory system within the file in a very simple format. Currently reading and writing files is done for every request in the system as there is no caching. We intend in future work to implement a Least Recently Used (LRU) caching system to reduce the number of reads and writes required by the system; however, we were more interested in making it work with JSR 75 at this time than making it work quickly.

6.5.4 Communication System

The communication system was considerably more complex to implement than either storage system largely because of the complexity of the underlying Bluetooth system and the need to implement a protocol to handle the data exchange.

We began the implementation by coming up with an abstract version of the wire protocol that could be used by all implementations. This class uses a thread to handle either the client or the server side of the protocol. The two halves are run as two instances of this abstract class in order to allow outgoing connections without having to turn on the inbound listening connection. When the class is created, the implementation must tell it if it is running the client side or the server side of the protocol. The protocol is the same on both sides with the order of operations reversed to avoid deadlock with both sides reading.

This class handles all details of the protocol except conversion of the request into the required input and output streams. Thus, an implementation has to only implement two abstract methods of the protocol `runServer` and `runClient` which constructs an `InputStream` (`java.io.InputStream`)

and an `OutputStream` (`java.io.OutputStream`) from the data passed in and then call either the `doServerProtocol` or `doClientProtocol` methods to finish the protocol. The implementation then simply has to add a connection to the protocol and it will be serviced by the protocol's thread in turn calling on the subclass to do the conversion to streams as required. This means that an implementation of the communication system is free from worrying about the wire protocol and threading issues and can focus on the issues specific to that communication system.

The next step was to implement the subclass of this protocol and the other classes required to implement the interface fully. We divided the Bluetooth system into two subsystems, one to handle the client side and one to handle the server side. The system first initializes the JSR 82 system and then kicks off each of the sides. The server side is responsible for opening a port for listening and accepting connections from the other side. The client side accepts requests from the application and passes them on to the protocol. Although portions are more complex than the storage system, they fell out of the design fairly easily.

6.6 Analysis

We tested our API using the application we built as discussed above. We were unable to complete testing of the JSR 75 implementation of the storage system on our device test platform, the Nokia N80. We believe this result was caused by the code signing issue since the code works properly on the emulator. The application throws an exception that cannot be caught by any catch block, including a `catch(java.lang.Throwable)` block and instead displays a message indicating that an uncaught exception was thrown. We have traced the origin of this throw to the first time we attempt to write to the filing system on the device. We believe that the restrictions on the phone may not allow MIDlets that have not been signed to write to the filing system. Note that the system does not ask for the users approval for the write, though calls to read from the filing system do cause prompts. This emphasizes both the problems with differences between emulators and actual devices and the code signing problem that we discussed earlier. We hope to resolve this issue in future work.

Our API provides a relatively simple data store and communication system that can be easily used by applications to implement a novel new system of data transfer. It is specifically designed

to be able to run on top of multiple communication and storage systems, thus hiding the complexity of those systems from application designers. This flexibility will allow us to move the API to desktop and server computers in order to be able to offer new forms of communication between those systems.

Chapter 7

Conclusion

7.1 Our Contributions

Our contributions include an analysis of the needs of disaster management applications that led us to suggest several novel ideas about the role that smart phones may play in disaster scenarios. This includes many applications that we have not built but which could prove invaluable if they were implemented and deployed properly on smart phones in a disaster area.

We have implemented an example of such an application for tracking lost and found people in order to understand the needs of such an application. This application is flexible and user extensible in terms of what data is collected proving that user participation in application design as well and rapid application development are possible on phone.

The very limited code size of our application to achieve this end demonstrates the power of our data-oriented API in achieving this end.

The API we have developed for managing both the storage of the collected data as well as the exchange of that data on smart phones is, as far as we know, unique. The ability of the API to hide both the communication and persistence layers creates an environment in which application designers are free to focus on the needs of the application while providing a relatively flexible system. Future enhancements to this system will add even greater flexibility.

We have also identified several areas where future work will greatly enhance not only our API but also other APIs that we discuss below.

7.2 Future Work

Obviously there is still a great deal of work to be done on both our application and our API. Although we have shown that our data diffusion concept is a workable system for enabling certain types of applications, it isn't appropriate for all applications. Sporadically connected networks are the exception not the norm and as such we need to improve on our API to better handle the normal connected scenario. We also need to address the limitations our API imposes on data types, the data versioning issue and issues of security and component bundling. Finally, we would like to extend our application environment to enhance the system so that ordinary users have even more power to build new applications on phone.

7.2.1 Serialization

Java is known for being a good platform for doing distributed programming because of the tight integration of synchronization and threading primitives in the language as well as support for communication primitives. Remote Method Invocation (RMI) is one of the most popular paradigms for doing distributed computing on Java; however, RMI does not exist on the smart phone platform because of the lack of serialization on which RMI relies to be able to transport arbitrary data objects across the network[14].

The current specifications for Java ME do not include reflection support, the ability to introspect objects, which is required for the current Java serialization system[14]. Without reflection support, the native Java serialization system cannot work. However Java ME already requires a pre-verification step in order to reduce the overhead of verifying the Java classes on the phone. There is no reason that an analogous step couldn't be used to provide a pre-serialization system where classes that implement the proper interface are rewritten at compile time to not require the runtime type inspection that reflection requires.

Conveniently, work on this problem has already been done by the Ibis project[29], which implements exactly this kind of serialization system. Our intention is to combine the Ibis serialization

system with the API in order to bring efficient serialization to the Java ME platform. We believe that porting this system to Java ME will bring significant benefits by enabling the familiar serialization paradigm on smart phone platforms and thus enabling more distributed APIs to be built for the platform.

7.2.2 Wireless Internet Support

As previously mentioned, sporadically connected networks are the exception not the norm. Our group already has an excellent system for enabling distributed applications on connected networks in the form of the Ibis[30, 29] system. Our intention is to bring support for connected networks like desktops to Raven by writing an implementation of our RavenCommunication system on top of Ibis and then bringing that to smart phones by porting Ibis to the smart phone platform.

The resulting system will enable novel distributed applications on smart phones that can easily communicate with computers running Ibis. This combination will enable new forms of collaboration on traditional computing platforms using our new API. We feel that this will benefit both traditional desktops and the smart phone platform by enhancing existing applications with the ability to add data to the system on both platforms even when they cannot communicate. For example, a users calendar could be registered as a type within our API. New events could then be added to both the phone and the desktop even if the two are out of communication. When they exchange information again, they will both have a complete view of the calendar. We believe that our API running over Bluetooth and/or Wireless Internet could have a huge impact on these more traditional applications as well as open up development for even more non-traditional applications that have yet to be imagined.

7.2.3 Versioning

Our API does not yet address the issue of modification of data, limiting the model to a write only data store. Although this protects our API from the need to handle multiple versions of the data it is not very realistic. What is needed is a way to track multiple versions of the data and to resolve differences between versions with the help of the users.

The lack of versioning is a significant problem, the complexity of which must not be underestimated. Our wish to keep the size of total data down to consume less of the limited resources on the platform adds to the complexity on our platform. We anticipate a system that can compare versions of data and reduce them to a single version with conflicts stored as a part of the data. This is further complicated by our desire to have security on the data. Because a user may modify only a portion of the data the system will need to address how to keep the signature on the data intact as we merge fields from individual versions. It is important to be able to keep the identity and signature on a portion of the data generated by one user intact on the portion of the data that is not being modified by the current user. This is at odds with the need to keep signing overhead low since it implies that each field is signed individually or else that the old version is kept around to verify the signature on it which increases storage requirements.

7.2.4 Security

Security is vital in this modern world and particularly in disaster control scenarios where bad data can lead to bad decisions. Data collected by our system needs to be integrity protected and signed so that command and control centers can verify the collector of the data as trustworthy and verify the fact that the data has not been tampered with. Neither our API nor our Application addresses this issue in any way. Making addressing this issue even more challenging is the resource-constrained nature of the application. However, it is important to note that the security model for Java MIDlets on smart phones includes cryptographic signatures on the code that stand as proof that verification of signatures is feasible on the platform. Nonetheless the generation of signatures and the verification of those signatures needs to be both efficient and straightforward.

We feel that the signing of the data needs to be as transparent as possible to the application programmer. As such, our future work will investigate adding integrity protection and digital signatures to our API in as simple a way to program as possible. We anticipate a system where the API accepts a signing key for the current user and then transparently signs all data that is placed into the data store by the user. Furthermore, the signature on data received from other users must be easy to verify as valid. This operation, because of the computational expense, should be performed as infrequently as possible.

The model of trust is paramount to the usefulness of this system. We anticipate using a PGP-style “Web of Trust” system to build up a network of trust. This model can include a central signing key if that is most appropriate for the organization in question; however, we feel that the anarchistic nature of “Web of Trust” style systems most closely matches the domain that we are working in. We would like to enable addition of new users to the system where trust of their collected data can be verified using the signature on their key generated by the individual who enabled them to begin participating in the system.

7.2.5 Component Integration

Our API has not yet been bundled in such a way as to be compatible with component-based architectures such as the OSGi platform. With wide support for OSGi-based systems from major phone vendors, we see significant advantages to bundling our API as a reusable component within the OSGi framework. This bundling will more easily enable component oriented systems to use our API.

7.2.6 On Phone Development

One of the things that came out of our analysis of the Katrina disaster is that many applications are not even imagined until a disaster is underway. Because of this unknown need we believe that rapid development of applications, preferably by non-programmers, is highly desirable. Although we have demonstrated that it is possible to build a system where users can specify the kinds of data that they are interested in collecting on the fly, we would like to extend this work even further and give users even more power to develop new applications on their phones. In the future we intend to explore new ways that users can participate in application development on their phones directly.

Bibliography

- [1] Osgi: the footings of the foundation of the platform. URL <http://www.eclipse.org/osgi/>.
- [2] Sahana foss disaster management system. URL <http://en.wikipedia.org/wiki/Sahana>.
- [3] Sahana: Free open source disaster management. URL <http://www.sahana.lk/>.
- [4] Symbian: Fast facts. URL <http://www.symbian.com/about/fastfacts/fastfacts.html>.
- [5] A. Acquisti. Protecting privacy with economics: Economic incentives for preventive technologies in ubiquitous computing environments. *Workshop on Socially-informed Design of Privacy-enhancing Solutions, 4th International Conference on Ubiquitous Computing (UBICOMP 02)*.
- [6] OSGi Alliance. Welcome to the osgi alliance (<http://osgi.org>). URL <http://osgi.org>.
- [7] David Brin. Designed to let us down... our deliberately frail cell phone system. URL <http://davidbrin.blogspot.com/2007/01/designed-to-let-us-down-our.html> .
- [8] G. Buchanan, S. Farrant, M. Jones, H. Thimbleby, G. Marsden, and M. Pazzani. Improving mobile internet usability. *Proceedings of the tenth international conference on World Wide Web*, pages 673–680.
- [9] Piero Campanelli. Status of open source osgi containers, January 2007.
- [10] Ian D. Chakeres and Joseph P. Macker. Mobile ad hoc networking and the ietf. *SIGMOBILE Mob. Comput. Commun. Rev.*, 10(1):58–60, 2006. ISSN 1559-1662.
- [11] C. Cordeiro and D. Agrawal. Mobile Ad Hoc Networking. *Tutorial/Short Course in 20th Brazilian Symposium on Computer Networks*, pages 125–186, 2002.
- [12] Venkat Amirisetty Motorola Jon Bostrom Nokia Corporation. Jsr 232: Mobile operational management. URL <http://jcp.org/en/jsr/detail?id=232>.
- [13] C. Greenhalgh, S. Izadi, J. Mathrick, J. Humble, and I. Taylor. Ect: A toolkit to support rapid construction of ubicomp environments. *System Support for Ubiquitous Computing Workshop, UbiSys04*.
- [14] William Grosso. *Java RMI*. O’Riley, October 2001.
- [15] J. Hynninen. Experiences in mobile phone fraud. URL <http://www.niksula.cs.hut.fi/~jthynnin/mobfra.html>.
- [16] Apple Computer Inc. iphone to support third-party web 2.0 applications. <http://www.apple.com/pr/library/2007/06/11iphone.html>, 2007 2007. URL <http://www.apple.com/pr/library/2007/06/11iphone.html>.

- [17] Ryan Kairer. Converged mobile device market grows 42 percent in 2006. URL <http://www.palminfocenter.com/news/9277/converged-mobile-device-market%-grows-42-in-2006/>.
- [18] Ryan Kairer. Handheld market continues downhill slide. URL http://www.palminfocenter.com/news/9411/handheld-market-continues-down%_hill-slide/.
- [19] Ryan Kairer. Report: 64 million smartphones shipped in 2006. URL http://www.palminfocenter.com/news/9247/report-64-million-smartphones-%_shipped-in-2006/.
- [20] M. Langheinrich. Privacy by design-principles of privacy-aware ubiquitous systems. *Proceedings of Ubicomp*, 2001.
- [21] Brian McConnell. Sms relay – an idea for fault-tolerant communications. URL <http://www.oreillynet.com/pub/a/wireless/2001/09/28/relay.html>.
- [22] Paul Nowak. Will enterprise open source scale the walled gardens of the cellular network providers? URL <http://opensource.sys-con.com/read/400928.htm>.
- [23] Andrew Orlowski. Through the (walled) garden gate... URL http://www.theregister.co.uk/2006/11/17/three_analysis/.
- [24] Steve Rosenbush. Tackling swamped communications, AUGUST 2005. URL http://www.businessweek.com/bwdaily/dnflash/aug2005/nf20050831_5227_db%_094.htm.
- [25] Jon Schull. Mesh-networking cellphones. URL http://www.socialtext.net/recovery2/index.cgi?mesh_networking_cellphon%_es.
- [26] M. Spreitzer and M. Theimer. Providing location information in a ubiquitous computing environment (panel session). *Proceedings of the fourteenth ACM symposium on Operating systems principles*, pages 270–283.
- [27] E-N-G Mobile Systems. High performance mobile cell sites – when and where you need them. URL http://www.e-n-g.com/pages/mobile_cell_cell_on_wheels.html.
- [28] Select Bipartisan Committee to Investigate the Preparation for and Chairman Response to Hurricane Katrina. Tom Davis. A failure of initiative. URL http://katrina.house.gov/full_katrina_report.htm.
- [29] R.V. van Nieuwpoort, J. Maassen, R. Hofman, T. Kielmann, and H.E. Bal. Ibis: an efficient java-based grid programming environment. *Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande*, pages 18–27.
- [30] R.V. van Nieuwpoort, J. Maassen, G. Wrzesinska, R.F.H. Hofman, C.J.H. Jacobs, T. Kielmann, H.E. Bal, et al. Ibis: a flexible and efficient java-based grid programming environment. *Concurrency and Computation Practice and Experience*, (7-8):1079–1107.
- [31] T. Yu, S. Hartman, and K. Raeburn. The perils of unauthenticated encryption: Kerberos version 4. *Proc. Network and Distributed System Security Symposium*, 4, 2004.