

DESIGNING EFFICIENT LOCATION-AWARE MIDDLEWARE FOR PERVASIVE USE IN MOBILE APPLICATIONS

ABSTRACT

An important design consideration for pervasive mobile computing is allowing applications to seamlessly run in the background. This enables, for example, an application to continually communicate with a server to exchange location information at the same time as allowing the user to check their email. The user should be unaware that the background task is executing. Although the mobile operating system may provide this functionality it is up to the application designers to develop their applications with certain constraints in mind. Obvious mobile development constraints include the limited resources of the hardware. Less obvious constraints include the nature of the communication networks used to exchange data. In this paper we present ongoing work on the Thumbtribes project - an open source project that attempts to provide proximity awareness to mobile applications in a way that facilitates pervasive use.

KEYWORDS

Pervasive mobile computing, location-aware middleware

1. INTRODUCTION

Thumbtribes is a low-bandwidth proximity service built on top of the Inter-Process Communication (IPC) technology D-Bus (Palmieri, 2005). The software has been designed to continually run with minimal user interaction as a background process commonly referred to as a daemon. Proximity is defined as the distance between the mobile device and other devices using the system. A typical use case might be to obtain the distance of another device and trigger an alert when that device is within a certain proximity. Because the daemon is designed to continually operate it attempts to minimize the amount of traffic generated. This is achieved by using encoding techniques based on the Packedobjects project (Moore, 2007a). Reducing the amount of data sent across high latency mobile networks can help improve response times. In addition it has the added benefit of reducing costs on metered networks typically found with prepaid phone contracts. Other benefits which may become apparent from taking light-weight approaches include reducing the memory footprint required to process data as well as reducing the amount of processing time required. Packedobjects adopts a lean approach which avoids dynamic memory allocation and uses one allocated area of memory for both the encoding and decoding of data. This is possible due to the simple synchronous nature of the communication protocol used (Moore, 2006).

Thumbtribes is based around one core operation. More specifically, the daemon will "ping" the device location to a central server. The server responds with a list of devices and their distances which are then cached locally. The "ping" process has been designed to encourage location updates rather than be dominated by network lookups. This self-clocking operation means the more frequently you send in your location the more up-to-date your location cache will be. It also follows that frequent updates provide other devices with a more accurate view of your position. Having updated your local cache of proximity information you are able to query it multiple times without incurring any network penalty. A simple $O(1)$ lookup can take place to translate unique identifiers into distances. The unique identifier may be something private that you share with a few others or might be something more public like an email address. Thumbtribes does not enforce any standard for this and therefore requires no account creation or validation. For example, a lookup for "john@doe.com" might return a distance of 100 kilometers. Updating your local cache will involve network

activity. However, as previously mentioned, a key design consideration is that the data transferred is encoded efficiently. To place some additional limits to the amount of information that is transferred a distance filter can be used. For example, by specifying a distance filter of 1000 kilometers your local cache would be limited to devices within that distance after a "ping" operation takes place.

2. INTRODUCTION TO D-BUS

D-Bus is a form of Inter-Process Communication (IPC) which allows applications to talk to one other over a bus. It is used extensively in both desktop and mobile Linux systems. Two categories of bus exist: the system bus which has a tight coupling with the operating system and the session bus which is employed on a per user basis. The latter provides a way for application developers to utilise IPC. D-Bus takes an object-oriented approach. This allows a direct mapping to object systems such as GLib (Warkus, 2004). Figure 1 depicts a simple form of IPC between two application processes. An application will create objects during its lifetime. Some of those objects may be native to the application whilst others may be proxy objects. A proxy object provides a simple way to call methods belonging to a remote object. Exactly what kind of methods can be invoked by a remote application is defined through an interface. Introspection can be used to find out the capabilities of other processes registered on the bus. In addition to messages, D-Bus also uses signals. A signal may be emitted by an application process to notify other application processes that a particular condition has occurred. Multiple applications can receive the same signal.

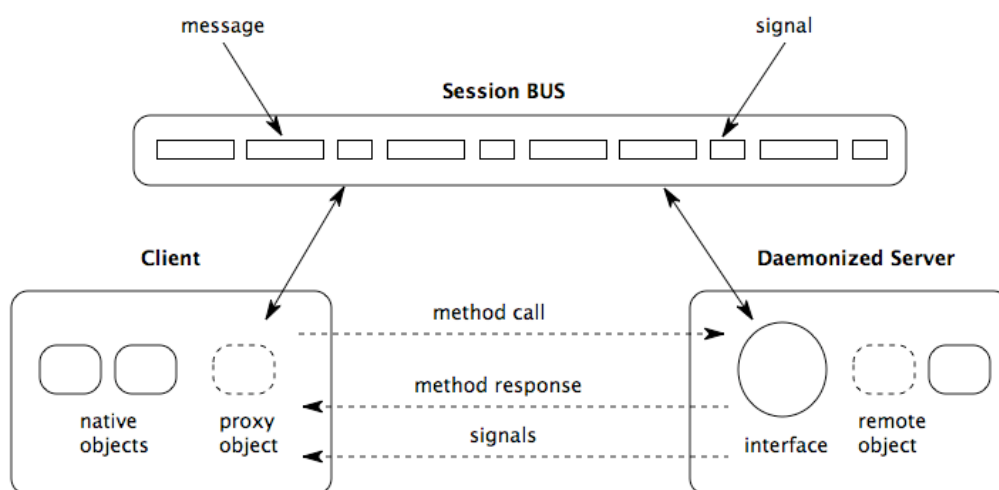


Figure 1: D-Bus architecture

When an application registers on the bus it may daemonise its process. This has the effect of running as a server. Other application processes can then communicate as clients. Object systems such as GLib provide ways of calling methods both synchronously and asynchronously. An application which invokes a method synchronously will need to wait until the remote object responds. If the client application is a user interface this may cause the application to appear frozen. If the remote application requires some time to handle the request an alternative is to use an asynchronous call. GLib handles this by allowing a callback function to be defined which will eventually be triggered when the remote application responds. In this case the calling application can return immediately rather than hang waiting for the response. Alternative approaches using threads could also be employed.

3. THUMBTRIBES DESIGN

Figure 2 illustrates the Thumbtribes architecture as a pipe. D-Bus messages are shown as being absorbed into the pipe through its outer layer. GLib provides the object-oriented interface for this communication. The middle layer of the pipe is where the embedded Scheme language is used to drive the encoding and decoding process. At the core of the pipe is the C encoder and decoder responsible for bit manipulation. Binary encoded data is sent across the Internet to the Thumbtribes server. Similarly, binary data from the server must be received and decoded by the layers of the pipe. The protocol is described using an s-expression in the Scheme programming language. An s-expression is a syntactic form which allows complex data structures to be represented more concisely than XML (Moore, 2007b). The ability to represent the protocol as code means that no further transformation is required to manipulate the data. If we contrast this to XML, the protocol would need to be transformed from its description into an abstract data type, such as a tree, before it could be manipulated. Even though s-expressions are concise, Thumbtribes goes one step further and transforms the textual representation of an s-expression into binary before transmitting it over a network. Data is encoded in a very efficient bit packed structure using the Packedobjects project (Moore, 2007a). Results showed that this form of compression produced on average a 36% reduction in message size in comparison to the popular GZIP compression standard (Moore, 2007b) (Deutsch, 1996).

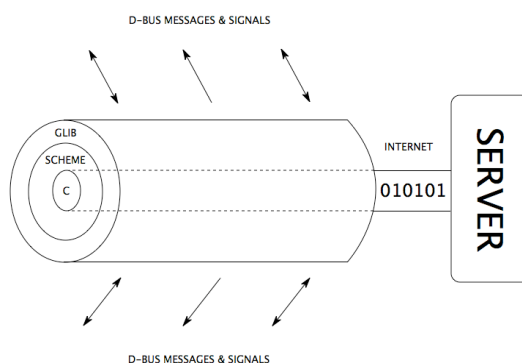


Figure 2: Thumbtribes architecture

To have a better understanding of how the described Thumbtribes architecture might function, we can create a typical use case. The Thumbtribes daemon has been designed to be part of a larger geo-information service. In particular, Thumbtribes requires location information to be supplied in the form of decimal coordinates. An obvious way to supply coordinates would be from a Global Positioning System (GPS). However, using a GPS may not always be practical. Firstly, GPS use can quickly consume a mobile phone battery. Secondly, a GPS does not function well or even at all indoors or in urban environments. Thirdly, turning on and off the GPS when required is not practical because the GPS may take minutes to obtain a fix. For these reasons, Thumbtribes does not solely depend on a GPS for location. Rather it has been designed to operate as part of a larger location providing system. Such a system already exists and is known as GeoClue (Preston and Kuosmanen, 2006). GeoClue is a geo-information service built on top of D-Bus. Its aim is to provide a simple way for application developers to create location-aware applications. GeoClue uses the notion of providers which can act as a source of location information. Various providers exist such as gypsy to obtain coordinates from a GPS device (Holmes, 2007). Less accurate providers that simply map the IP address to coordinates can also be employed. The developer can interact with a master provider which abstracts these different underlying providers and can choose the best method based on the client's requirements. For example, if accuracy is important and there is a functioning GPS, the information may come from the gypsy provider. In this architecture, Thumbtribes could also be considered to be a provider dependent on other providers. It is able to provide proximity information but requires decimal coordinates from another provider to function.

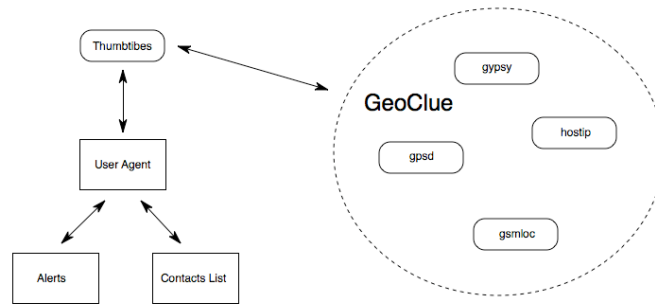


Figure 3: GeoClue

For the target application to appear pervasive it is perceived that there will be no obvious user interface. Figure 3 depicts one such scenario. A User Agent (UA) will run in the background and obtain location information from GeoClue. It will then "ping" in this information to update the local Thumbtribes proximity cache. The UA will then try and match information from different sources such as the phone's contact list against the proximity cache. If a match is found, the UA will use the phone's underlying features to alert the user accordingly. For example, the phone may vibrate and display a window to suggest that "John from your contacts" is nearby. A user of the Thumbtribes system may be a person with an email address or a person using a unique hash. It may even be feasible that the unique identifier used in the system has been automated by the device. Therefore, the information in the proximity cache may have different applications. The obvious choice is to use email addresses which can be compared to other email address aggregate sources as previously described.

4. FUTUREWORK

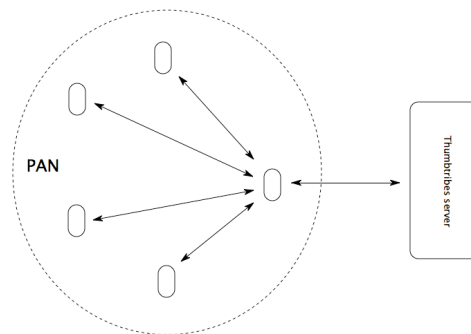


Figure 4: Personal Area Network

The binary has been successfully built and run on OpenMoko's Neo 1973 developer's phone (OpenMoko, 2007). This device has 128MB of RAM which is fairly common place amongst current smart phones. Analysing memory usage on the Neo 1973 showed the Thumbtribes binary running had negligible impact on overall memory usage. However if memory is at a premium, an alternative approach would be to re-write the embedded Scheme code in C. The approach taken by the Thumbtribes project is to prototype with Scheme with the intention to replace with C if justified.

In terms of feature enhancements some novel possibilities exist. Each device contains its own proximity cache. If it is possible to share this information with nearby users it would be possible to do simple proximity lookups without requiring any Internet connection. The lookup could take place over a Personal Area Network (PAN) powered by technologies such as Bluetooth. Figure 4 illustrates this concept. Only one user has an Internet connection and is therefore able to obtain proximity information from the Thumbtribes server. The other users are then able to query the cache stored on the single device. Other more elaborate sharing possibilities exist.

5. CONCLUSION

Thumbtribes has been designed around the principle of a self-clocking proximity cache. The term self-clocking describes the usage pattern of supplying location information to obtain more up-to-date information in the cache. The proximity cache is a local store on the device which contains the relative distance of other users or devices in the Thumbtribes system. A simple API exists to allow a User Agent (UA) to query the cache by supplying a unique identifier in return for a distance expressed in meters. This mapping can be used in applications to provide proximity alerts. For example, the UA may automate querying the phone's contact list and then in turn query the proximity cache to see if any contacts are within a certain distance. The system has been designed to be independent of existing online services which require account creation and validation. A key design principle is that all data used to build the proximity cache has been encoded in an efficient manner. This has been achieved by using binary encoding techniques based on Packedobjects. The Packedobjects system provides a more light-weight alternative to encoding data via XML or applying standard compression techniques to text-based protocols. Another key design principle is the software has been built to run with minimal user interaction. The Inter-Process Communication (IPC) system known as D-Bus has been used as the underlying infrastructure to help achieve this goal. The Thumbtribes software runs as a background process known as a daemon and provides an object oriented interface accessible via D-Bus. The interface can easily be integrated with other location-aware D-Bus services found such as those available within the GeoClue project. Together these key design principles help facilitate developing pervasive mobile applications.

REFERENCES

- Deutsch, P., 1996. *GZIP file format specification version 4.3, Request for Comments: 1952*. <http://www.ietf.org/rfc/rfc1952.txt>.
- Holmes, I., 2007. *Gypsy - A GPS Multiplexing Daemon*. <http://gypsy.freedesktop.org/wiki/>.
- Moore, J., 2006. Packedobjects. In *WINSYS 2006 International Conference on Wireless Information Networks and Systems*.
- Moore, J., 2007a. *Packedobjects*. <http://packedobjects.com>.
- Moore, J., 2007b. Thumbtribes: low bandwidth, location-aware communication. In *WINSYS 2007 International Conference on Wireless Information Networks and Systems*.
- OpenMoko, 2007. *OpenMoko Neo 1973*. <http://openmoko.com/products-neo-base-00-stdkit.html>.
- Palmieri, J., 2005. Get on D-BUS, *Red Hat Magazine*. <http://www.redhat.com/magazine/003jan05/features/dbus/>.
- Preston, K. and Kuosmanen, T., 2006. *GeoClue: The Geoinformation Service*. <http://www.freedesktop.org/wiki/Software/GeoClue>.
- Warkus, M., 2004. *Gnome 2.0 the Developers Guide*. No Starch Press.