UMEÅ UNIVERSITY
Department of Computing Science
Jon Hollström
Per Nordlinder
{jon, per}@cs.umu.se

2003-01-13

RRApex

An APEX Implementation for the RoadRunner Toolkit

Advisors: Pedher Johansson & Jonas Borgström
Examiner: Per Lindström

"The APEX Program makes parents aware that they have a responsibility to be actively involved in their child's education at various levels."

The ASPIRA Association APEX Internal Implementation Guide, 1991

Abstract

This report describes our master's thesis work: a BEEP/RoadRunner implementation of The Application Exchange Core (APEX). BEEP and APEX are parts of a new IETF approach, aiming to standardize and simplify application level communication protocol development. About 20 years of protocol development and sharp testing, primarily on the Internet, have resulted in loads of experiences, all giving a clue on how to design a common framework for future protocols.

We present the background to the task, the design issues and implementation details. We also look at a realistic application of APEX, namely instant messaging, the techniques used today on the Internet and finally compare them to the APEX Core when used in an instant messaging library.

The result is a RoadRunner profile (library), implementing most parts of the APEX standard. An instant messaging tool, *Apecis*, have also been written to test and show the functionalities in the library.

Contents

1 Introduction **1**
- 1.1 Task specification . . . 1
- 1.2 Overview . . . 2

2 Background **3**
- 2.1 Instant messaging today . . . 3
- 2.2 Another approach . . . 3
- 2.3 BEEP – Blocks Extensible Exchange Protocol . . . 4
- 2.4 APEX – Application Exchange . . . 6
- 2.5 Earlier work . . . 7
  - 2.5.1 The APEX Working Group . . . 7
  - 2.5.2 RoadRunner . . . 7
  - 2.5.3 Netlab research . . . 8

3 Implementation **9**
- 3.1 Environment . . . 10
- 3.2 Implementation details . . . 10
  - 3.2.1 Overview . . . 10
  - 3.2.2 Threads . . . 10
  - 3.2.3 Data structures . . . 10
- 3.3 Algorithms . . . 12
  - 3.3.1 Algorithms for relay nodes . . . 12
  - 3.3.2 Algorithms for endpoints . . . 14
- 3.4 Tests . . . 15
  - 3.4.1 Apecis . . . 15
  - 3.4.2 Tests with Netlab implementation . . . 16
- 3.5 Result . . . 16
  - 3.5.1 Limitations . . . 17

4 Discussion **18**
- 4.1 19 years of application protocols . . . 18
- 4.2 The nature of BEEP . . . 20
- 4.3 BEEP as a common framework . . . 20
- 4.4 Add APEX to BEEP, what do we get? . . . 20
- 4.5 The need of an instant messaging standard . . . 21
- 4.6 APEX as part of an instant messaging standard . . . 21
- 4.7 Future work . . . 21
- 4.8 Conclusion . . . 22

5 Acknowledgment **23**

A Tools and libraries **25**
- A.1 GLib . . . 25
- A.2 GTK+ . . . 25
- A.3 Libxml . . . 25
- A.4 Other tools and environment . . . 26

B License for the RRApex library **27**

C API documentation for RRApex and RRApexPresence **28**

List of Figures

1 Example of a typical BEEP/APEX protocol stack. 4
2 Classic TCP/IP socket approach 5
3 New BEEP approach 5
4 The structure of an APEX system. 6
5 Well-known endpoints on top of the APEX core. 7
6 RRApex and the surrounding interfaces. 9
7 Class hierarchy for APEX implementation. 11
8 The Apecis GUI. 15

List of Tables

1 Tests with Netlab implementation 16
2 Problems for a generic application protocol 19
3 The foundations of BXXP (BEEP). 20

1 Introduction

The last 20 years, many new Internet application protocols have been designed and reached *Request For Comments* (RFC) status. An RFC is a document describing an Internet standard. All RFC documents are authorized by *The Internet Engineering Task Force* (IETF) [9]. Many of the application protocols solves about the same problems. Each protocol in its own way, which means, every protocol has its own methods and principles for sending control information and data, providing authentication and privacy, etc.

To make application protocol design more efficient and simple, Blocks Extensible Exchange Protocol (BEEP) [17] has been designed. BEEP specifies a framework for application protocol design and is suitable for every application protocol in need of time-insensible, connection-oriented, asynchronous interactions, e.g., message passing between network peers. The designer has only to focus on the part of the protocol specific to the actual application. Details about data encoding and sending, authentication and privacy are of less importance for the designer since BEEP takes care of these parts.

As an attempt to make design easy, and to make BEEP usable, Codefactory AB[1] has implemented BEEP and developed a toolkit for building protocols using it, namely RoadRunner [2]. Many experts around the world (even Dr. Marshall T. Rose, the designer of BEEP) agree that RoadRunner is the best BEEP implementation available today. RoadRunner is written in object oriented C.

The Application Exchange Core (APEX) [20], is a new standard for application communication. APEX is built on top of BEEP and provides connection-oriented communication, a request/reply-pattern for messages and support for asynchrony. Hence, *one* of its natural areas of use is *instant messaging*. There are some differences though, between instant messengers today, like *ICQ* [6], *AOL Instant Messenger* [1], *Jabber* [4] and so on, and an APEX instant messenger.

To begin with, APEX is an open standard, written by Marshall Rose and authorized by the IETF, which makes it usable for open source programmers (and others) all over the world. APEX is also very general in the sense that messages can contain arbitrary MIME [10] content, which means that one can send almost any data over an APEX channel, for example binary data or streaming video. This means that APEX is useful in a rich number of other applications than instant messaging.

1.1 Task specification

The Master's Thesis project task is to implement and evaluate the new APEX specification [20], by using the technique in a message-delivery system for instant messaging and presence information. The message delivery system shall be written to provide about the same services as Jabber and *Oscar* (the protocol

[1] http://www.codefactory.se/, an open source software consulting company with head office located in Umeå

behind ICQ [6] and AOL Instant Messenger [1]).

The implementation is of *proof-of-concept* type. The intention is not to write a complete instant messaging tool, but to implement the underlaying APEX relay-mesh library. We will also aim to answer these questions:

- ⋆ What service and functionality does the application programmer want from the APEX library? What does a general APEX interface look like?
- ⋆ How shall the APEX layer be implemented to be a profile for BEEP in general and RoadRunner in particular? And what parts of APEX should me modularized?
- ⋆ How do we keep an implementation portable to all platforms? Is there any special requirements to make the library work on for example a Microsoft® Windows™platform?
- ⋆ What standards do we have/need on top of APEX to get a complete message system? What applications are relevant and interesting?
- ⋆ What does APEX give us? Is there need for yet another instant messaging standard, or is APEX more than that?

In this report, our implementation of APEX is presented. It is the result of a joint work between the Department of Computing Science[2] at Umeå University and Codefactory AB. It is performed as a Master's Thesis project. The students are Jon Hollström and Per Nordlinder, currently at the Master of Science in Computing science and Engineering program at Umeå University. Pedher Johansson (department of Computing Science, Umeå University) and Jonas Borgström (Codefactory AB) are advising the project.

1.2 Overview

In section 2 we present background information on the area and take a look at earlier work. A detailed description of our work and implementation details can be found in section 3. Finally, we discuss standards in general, the need of an open standard for instant messaging, advantages with our implementation and what is left to do in the future in section 4 below. In the same section we also try to formulate answers to the questions asked above.

The APEX implementation will be written as a profile for RoadRunner. More information about RoadRunner can be found in section 2.5.2. Information about other tools and libraries we have used can be found in Appendix A.

[2]Information available at http://www.cs.umu.se/

2 Background

Instant messaging is an interesting application of the APEX technology. Therefore, we will now take a brief look at the techniques and protocols used for instant messaging today, and then describe a new APEX approach.

2.1 Instant messaging today

Today, a few large companies compete with different technologies for instant messaging. The big players are:

- ⋆ **Mirabilis ICQ [6]**, first version released in 1996 by a small company in Israel. The company was bought by America Online in 1998. America Online is today part of the large AOL Time Warner Corporation. Today the ICQ network have about 150 million registered users. The users are connected to an ICQ server. The server keep track of the users presence information (online, away, offline, etc.). Messages can be sent between two ICQ users in two ways: peer-to-peer direct connection or via the ICQ server.
- ⋆ **AOL Instant Messenger (AIM) [1]**, a message service for the 35 million registered AOL users. AIM use the protocol *Oscar*. Lately, ICQ changed protocol and are now using Oscar as well.
- ⋆ **Microsoft MSN Messenger [5]**, Microsoft's answer to ICQ and AIM. Support for shared white-board, streaming voice and video. Version 1.0 of the protocol is described in the (expired) IETF draft *MSN Messenger Service 1.0 Protocol* [16]. The draft is written by Microsoft and IETF do not have the right to publish it as RFC.
- ⋆ **Jabber [4]**, an open XML protocol for exchange of instant messages and presence information. Jabber is based on the client-server model. Messages between users normally pass at least one Jabber server. Since Jabber is an open protocol, based on XML, it is also the most extensible and flexible of the protocols mentioned so far.

The protocols above are built direct on top of TCP/IP. They handle all parts of the message passing; framing, encoding, reporting, authentication, privacy, end user routing and so on.

2.2 Another approach

It all started back in 1998. Marshall Rose and Carl Malamud where sitting down, reviewing protocol architectures. They came to one conclusion; We need a general protocol to solve the common problems and provide the common protocol services.

Rose wrote down their ideas about the structure of a such protocol in a document. The document was released as a draft document, and later reached RFC

status. It is now known as RFC 3117 [18]. We will talk more about this document in section 4.1. The ideas presented in the document, later came to be the fundamentals of BEEP, described below.

BEEP can use different transport level protocols, for example the *Transmission Control Protocol* (TCP), and it is *on* BEEP, that APEX enters the scene. Figure 1 illustrates this protocol hierarchy.

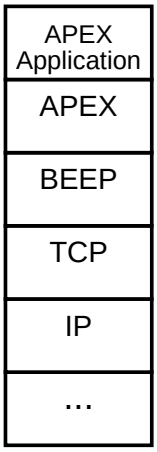

Figure 1: Example of a typical BEEP/APEX protocol stack.

2.3 BEEP – Blocks Extensible Exchange Protocol

In [18], *On the Design of Application Protocols*, design principles for a generic application protocol framework, developed to support connection-oriented asynchronous interactions, are presented. BEEP, described in [17], is such a framework (or kernel). BEEP can be described as a framing mechanism that permits exchanges of messages between network peers.

Exchanges can be done simultaneously and independent of each other. The messages are sent via BEEP channels and are arbitrary *MIME* (Multipurpose Internet Mail Extensions) ([10]) content structured using XML, eXtensible Markup Language [3]. A BEEP connection has one or more channels and every channel has a profile attached to it. This is how the multiplexing (many logical connections over one transport level connection) is done in BEEP.

The channel profile handles syntax and semantics of the messages exchanged. The BEEP RFC also defines a channel management profile, a *TLS* (Transport Layer Security) profile and the *SASL* (Simple Authentication and Security Layer) family of profiles.

Application programmers implement, for the task specific protocol parts, as add-on profiles to BEEP. Even APEX, described below and specified in [20], is implemented as a profile for BEEP. Figure 2 illustrates the classic TCP/IP socket approach. Figure 3 illustrates the BEEP approach.

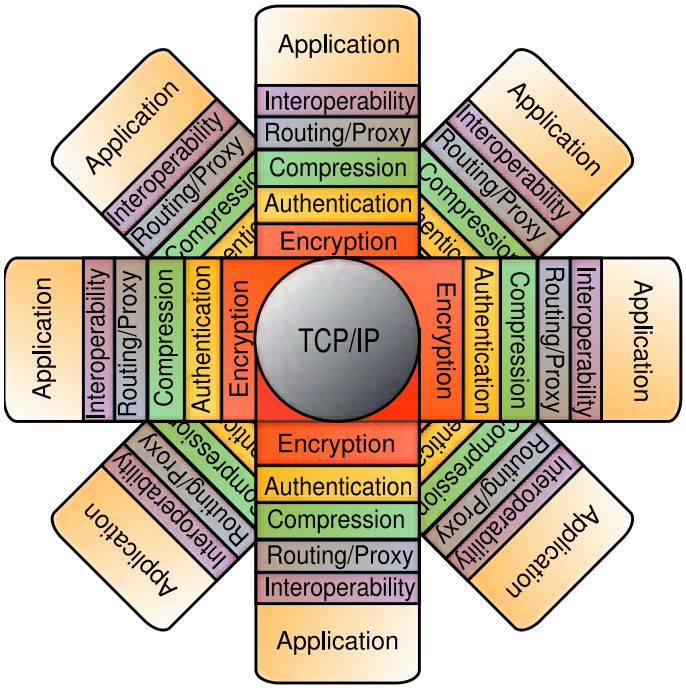

Figure 2: Typical design layers for TCP/IP application protocols. Each protocol imeplements its own functionality layers, reinventing the wheel every time.

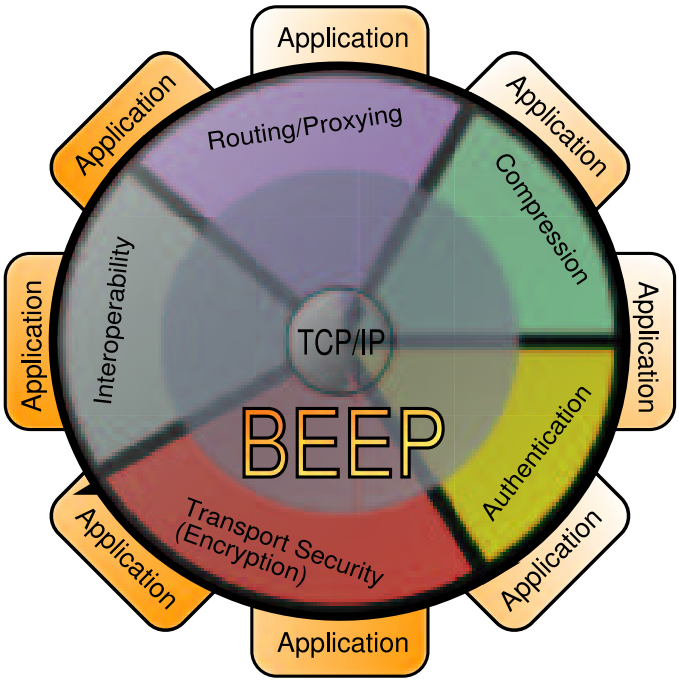

Figure 3: Typical design layers for BEEP based application protocols. Profiles (modules) implementing functionality are widely reused. Reducing risk in the design phase allows increased interoperability, security, efficiency and extensibility.

2.4 APEX – Application Exchange

The *Application exchange core* (APEX), is an Internet standard for exchange of application-level messages, useful for applications with the following needs:

- ⋆ Sending *and* receiving messages (push and pull)
- ⋆ Asynchronous communication
- ⋆ Stateless communication

For example, instant messaging programs have the above needs, while the world wide web only has *pull*, and Internet mail delivery only has *push* as method for message delivery.

APEX is described in 3 RFCs, namely:

- ⋆ **RFC 3340: The Application Exchange core [20]**
  Main document for the basic structure. Abstract algorithms and the XML Document Type Definition (DTD) for the basic APEX messages.
- ⋆ **RFC 3341: APEX Access Service [19]**
  Conventions for access restrictions and control.
- ⋆ **RFC 3342: APEX Option Party Pack, Part Deluxe! [11]**
  Document presenting more options for the APEX message passing.

The APEX core provides a message passing service for end user applications. There are two types of nodes in the core, *relay nodes* and *endpoints*. At the endpoints, users run applications. The relay nodes act more like servers. One relay node handle one *administrative domain*, (e.g. codefactory.se).

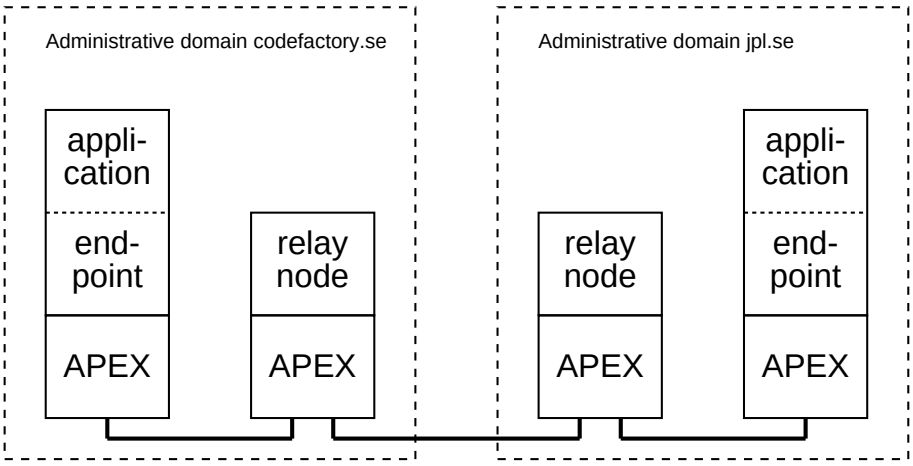

Figure 4: The structure of an APEX system.

Messages are addressed on the form *user@domain*, (e.g. jon@codefactory.se). If the user field has a solidus character ("/"), like jon/appl=wb@codefactory.se, the

string after / is a sub address. Sub addresses starting with appl= are reserved for use by APEX endpoint applications registered with the *Internet Associated Numbers Authority* (IANA) [8].

At every administrative domain, there is a set of *well-known endpoints* (WKEs). APEX applications communicate with the APEX services by addressing messages to a well-known endpoint. The endpoints are addressed on the form apex=wke, where wke is the name of the well-known endpoints.

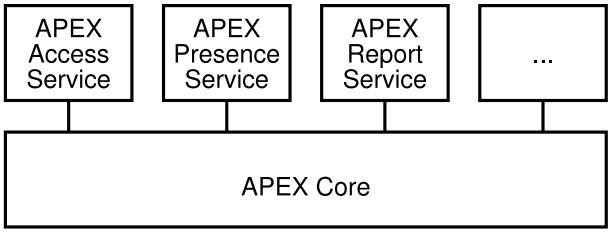

Figure 5: Well-known endpoints on top of the APEX core.

2.5 Earlier work

In this section, we present a summary of what has been done earlier on the subject, witch is, not very much since APEX is a quite new standard.

2.5.1 The APEX Working Group

The APEX working group is an IETF organized group, lead by Pete Resnick. The goal for the group is to develop the different parts of APEX, to provide a more or less complete instant messaging framework.

So far, three documents have reached RFC status ([20], [19] and [11]), and at least one document is under construction ([14]) and will probably gain RFC status in the near future. More information about the group can be found at:

http://www.ietf.org/html.charters/apex-charter.html

In addition, the working group tries to specify CPIM-compliant application services for text-based instant messaging and for online presence, based on the APEX service. CPIM is Common Presence and Instant Messaging, an IETF attempt to standardize instant messaging and presence messages. CPIM today consists of six draft documents. These are available online from:

http://www.ietf.org/ids.by.wg/impp.html

2.5.2 RoadRunner

RoadRunner [2] is a powerful full-featured implementation of BEEP, written in object oriented C by Jonas Borgström and Daniel Lundin at Codefactory AB. It is an application toolkit library for use in network application development. RoadRunner also consists of a set of profiles like APEX, usable in applications.

RoadRunner, as well as our APEX profile, is written using the GNU utility library (GLib) to ensure portability between platforms on areas like multi-threading, memory handling, data types and network communication.

RoadRunner can be used by C programmers on every GLib compliant platform. The library can also be used with C++ and Python. The latter through special bindings.

2.5.3 Netlab research

Michael J Riggio, Temple University Netlab, have been working on an APEX implementation of IMPP, RFC 2779 [13]. The ambition was to write an APEX implementation using the BEEPCore Java API.

Since it would be good for us to have at least one other implementation to test our implementation against, we were really hopeful. But when we found Mr. Riggio's code incomplete and non-standard compatible, our hope faded away. More informations about the Netlab APEX project can be found at http://netlab.cis.temple.edu.

3 Implementation

We have implemented APEX as a profile for RoadRunner. The result is a "plug in" library for RoadRunner called *RRApex*. Since we have used object oriented C, our design has an object oriented approach. The C data structures are used (and can be seen) as objects/classes and the functions take a structure-pointer as first parameter (the *instance* for which the *method* is invoked).

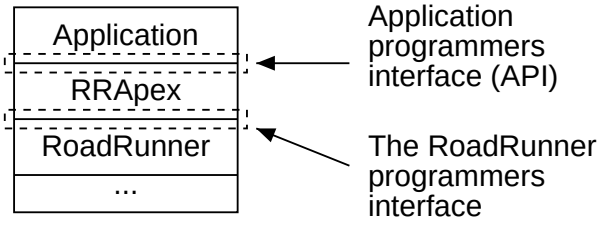

Figure 6: RRApex and the surrounding interfaces.

The first and maybe the most important design issue, was the design of the application programmers interface (API). After some design advices from one of the LICQ (Linux ICQ clone) authors, we decided to keep the interface simple but powerful. Asynchrony is possible through call-back functions. The complete API documentation can be found in Appendix C.

Figure 6 illustrates the surrounding interfaces for RRApex. The RoadRunner programmers interface was, when we started the implementation, undocumented. Loads of communication with the RoadRunner authors later, we got an idea about how RoadRunner could be used. The RoadRunner interface is now documented, and can be found online at:

http://rr.codefactory.se/doc/

The principle for the RoadRunner interface is simple: special options and functions are registered when RoadRunner is initialized and channels are created. As soon as the channels are created, incoming messages are passed up to the RRApex level. The messages can be of different types. Below, the most common types are listed.

RR_TYPE_MSG This is the type for regular messages sent from one peer to another. The initiating message has this type.

RR_TYPE_RPY This is the type for reply messages. When a peer is responding to another (sending an acknowledgement), which has previously sent a RR_TYPE_MSG, this type is used.

RR_TYPE_ERR This type is for sending error replies (acknowledgements) when something unexpected occurs.

With help from these message types, the APEX library get a system for safe transmission of data. The message number for every RR_TYPE_MSG that is sent, is pushed onto a queue waiting for reply. As soon as this message has been

responded to (either with a RR_TYPE_RPY or a RR_TYPE_ERR), it is popped from the queue, indicating the original message has reached its destination. This also ensures that every message is received in the correct order. This method can be compared to the acknowledgement system in TCP.

3.1 Environment

Since RoadRunner[2] is implemented using GLib, we are not restricted to a specific operating system or platform. We have chosen to write the code, compile and run tests primary on GNU/Linux (Debian) systems. Also the APEX profile is implemented using GLib, so it is easily portable to all GLib platforms.

The code was written in object oriented C like RoadRunner, and the compiler used was the C front-end to the *GNU Compiler Collection*, GCC [7]. For XML message parsing, Libxml is used. The test application is implemented using the Gimp Toolkit 2 (GTK2) and Glade/libglade. All tools and libraries are described in Appendix A.

3.2 Implementation details

3.2.1 Overview

Since exchange of APEX messages must be asynchronous, we have used the method of *call-back functions* to bring information from the library up to the application. E.g., whenever a DATA message arrives to an endpoint, the endpoint APEX-instance simply call the function (if any) registered by the application and pass the message payload to the application. Further processing of incoming messages is left to the application.

3.2.2 Threads

Incoming DATA messages to a relay node are sent immediately to a pool of *light-weight-processes* also known as threads. Each message are then received, processed and distributed by one of these threads.

3.2.3 Data structures

The main structure (class) is named RRApex and represents an APEX channel. It inherits the RoadRunner class RRChannel, representing a BEEP channel, which itself inherits the super class GObject from GLib. The most important data structures together with their member attributes are shown in Figure 7 on page 11.

An endpoint node has one APEX channel, hence one RRApex object. A relay node, on the other hand, need zero or *more* channels, to be able to serve many endpoints and keep connections to other relay nodes alive. Therefore, the endpoints can keep their configuration (RRApexEndpointConfig) in the RRApex

structure. The RRApexEndpointConfig structure contains information about the callback functions registered by the application.

Relays, on the other hand, are passing their configuration RRApexRelayConfig to APEX through the global_config, since the configuration shall be the same one among all APEX channels, independent of the current number of active channels. The relay configuration keep track of callback functions for registered services (e.g. presence) in the relay. The passing of configuration structures are shown as dashed arrows in Figure 7.

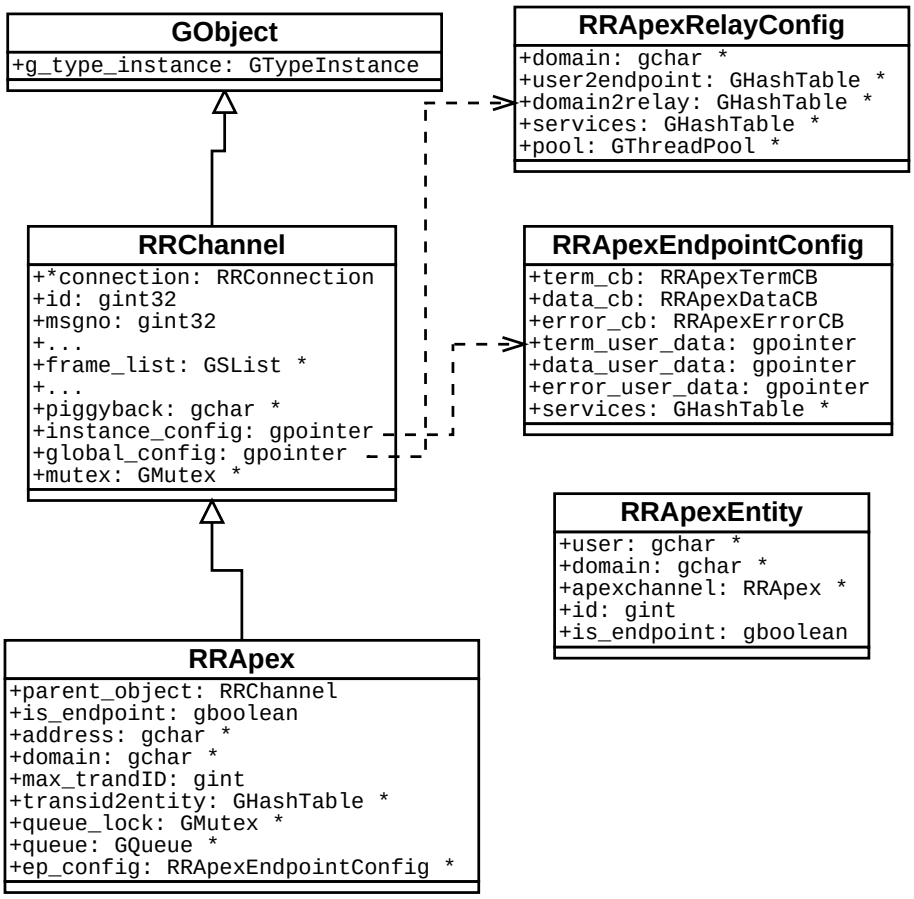

Figure 7: Class hierarchy for APEX implementation.

The relays (containing one or more APEX channels), keep track of the channels through hash tables. The first hash table, named user2endpoint, gives a username, (e.g. per), returns an RRApexEntity structure containing information about and a pointer to the APEX channel for the user per in *the same administrative domain.*

A relay also contains a hash table named domain2relay which given a domain name returns an RRApexEntity structure containing information about a relay.

Hence, if an endpoint (per@cs.umu.se) send information to another endpoint (jon@codefactory.se), the information is first passed to the relay for cs.umu.se.

The relay notice that the recipient is in another administrative domain and look that domain up in the domain2relay hash table. It forwards the information to that relay. The relay for codefactory.se recognize the domain as its own and simply look for the user jon in its user2endpoint hash table.

Every APEX channel (the RRApex structure) also contains a hash table named transid2entity. This is because every APEX channel contains a multiplexing mechanism called *transaction id*. One application can for example register multiple endpoints over one single APEX channel. Every endpoint is then associated with a number like 1 or 2. Given such a number, the hash table transid2entity returns an RRApexEntity structure containing information about that specific endpoint.

3.3 Algorithms

Below the interesting algorithms for our solution are presented. They might look a bit straight forward and on a high level of abstraction. That is because we wanted to keep the description short and possible to overview. The reader interested in all the details can read the source code, freely available at:

http://www.cs.umu.se/~jon/exjobb/src/

In the normal scenario, an endpoint sends a message to the APEX service at the relay for its administrative domain. The relay passes the message to the relay for the receiver's administrative domain. The receiver's relay passes the message to the receiving endpoint.

3.3.1 Algorithms for relay nodes

The relay node is listening for incoming connections. When a new connection appears, RoadRunner gives us a channel, and the following algorithm is used to handle incoming messages.

1. The channel is registered.
2. When a frame (message) is available on the channel, different actions are taken, depending on the message type. To start with it depends on whether the message is a new one, or a reply or an error to a previous message sent from this relay.
   (a) If the message type is RR_TYPE_RPY or RR_TYPE_ERR, the message is a reply or an error message. The message number is removed from the queue of messages waiting for response. If it is an error message, actions are taken to correct the error.
   (b) Else, if the message type is RR_TYPE_MSG, the message is a new incoming message and actions are taken depending on the contents. These algorithms are shown below.

ATTACH The XML message is parsed:

1. If the attach message contains an invalid identity, error code 558 is returned.
2. If the attach message contains an invalid transaction ID, error code 555 is returned.
3. If the endpoint does not belong to this relay (administrative domain), error code 553 is returned.
4. If another application is registered with this username, error code 554 is returned.
5. Else, new endpoint is inserted into hash tables and *OK* is returned.
6. The channel is marked as *channel to endpoint*.

BIND The XML message is parsed:

1. If the bind message contains an invalid transaction ID, or if it is already registered, error code 555 is returned.
2. If the binding relay wants to register as relay for the same administrative domain as this one, error code 537 is returned.
3. Else, the relay is accepted as serving its administrative domain. It is inserted into hash tables and *OK* is returned.
4. The channel is marked as *channel to relay*.

DATA The XML message is parsed:

1. If the message is not an XML nor a MIME message, error code 558 is returned.
2. Originator, recipient(s) and services (if any) are found out through parsing of the message.
3. If the message have an invalid or no content tag, error code 559 is returned.
4. If the message is addressed to a service, which is not registered, error code 556 is returned.
5. Else, the callback function for the service is called.
6. If the originator is trying to spoof his/hers identity, error code 558, 557 or 537 is returned, depending on what is not correct.
7. An *OK* message is returned to the sender.
8. Start a thread and give it the recipient(s) together with the message. The tread then performs the following:
   (a) For each *recipient:*
      i. If the recipient contains an invalid identity, set error code 558 for that recipient.
      ii. If the recipient is in the same domain and is connected, send the message with only one recipient to that endpoint.
      iii. If the recipient is not connected, print error message to error file handle and continue.

iv. Else if the recipient is in another domain and this relay already have a connection to that relay, send the message with only one recipient to that relay.

v. If this relay is *not* connected, send a bind request to that relay. If it turns out well, send the message with only one recipient to that relay.

vi. Free data.

TERMINATE The XML message is parsed:

1. If the transaction identifier given is invalid (e.g. NaN), error code 550 is returned.
2. If the value of the transaction identifier is zero, then all associations established by this application over this BEEP session, either as an endpoint attachment or a relay binding, are terminated, and *OK* is returned.
3. If the transaction identifier does refer to an already terminated operation on this BEEP channel, error code 550 is returned.
4. Else, the operation on this BEEP channel corresponding to the given transaction identifier is terminated and *OK* is returned.

3.3.2 Algorithms for endpoints

The endpoints are listening for incoming messages as well as outgoing messages. The algorithm for sending messages is very simple. All the endpoint has to do is compose a new message and send it to the relay, which takes care of the rest.

The algorithm for incoming messages is also quite simple, since the only event is incoming **DATA** and **TERMINATE** messages.

Incoming **DATA** messages are processed and passed up to the application layer through a call-back function registered by the application itself. **TERMINATE** messages are treated the same way. They are passed to the application via a call-back function. A difference though, is that the APEX session is closed after receiving this message.

If the endpoint receives a **DATA** message, the XML is parsed as follows:

1. If the message is not an XML nor a MIME message, error code 558 is returned.
2. Originator, recipient(s) and services (if any) are found out through parsing of the message.
3. If the message have an invalid or no content tag, error code 559 is returned.
4. If the message is addressed to a service, which is not registered, error code 556 is returned.
5. Else, the callback function for the service is called.

6. Reply with an OK to the relay.
7. Pass the message to the callback function for **DATA** messages, registered by the application.

The channel to the relay node are initialized by sending an **ATTACH** message. Outgoing messages are sent as **DATA** messages to the relay over the open APEX session. Whenever we wish to disconnect, a **TERMINATE** message is sent to the relay and the endpoint is detached from the APEX mesh.

3.4 Tests

The APEX profile have been developed with running tests at all stages of the code maturity. To make the tests more real, we have written an instant messaging application. We have also tested our implementation against the Netlab implementation mentioned earlier.

3.4.1 Apecis

To make testing more realistic, we have written a tool called Apecis. It is a primitive instant messaging application written in GTK 2, developed for a Linux environment but there shall be no problem to use it on other Glib/GTK compliant platforms like Windows, Solaris and BSD.

The GUI is formed as a ”normal” instant messaging GUI with a tree-like list of ”buddies”. Figure 8 shows a screen-shot of the GUI.

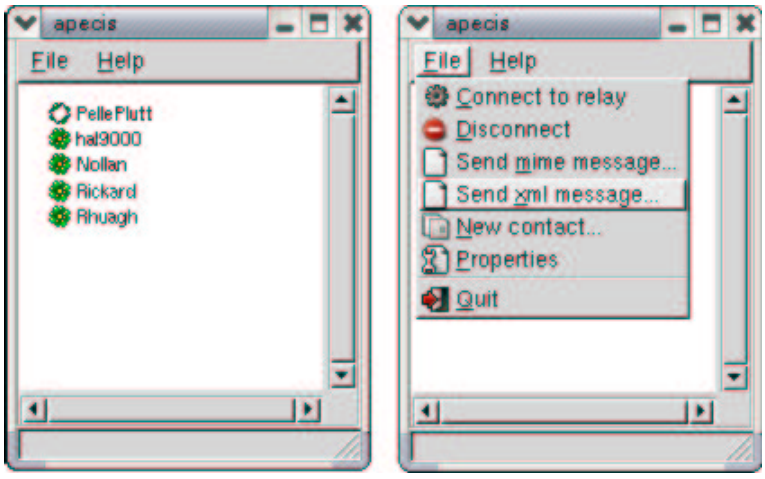

Figure 8: The Apecis GUI.

Libglade are used for the GUI. This means that the whole GUI are represented in an XML document. Changes to the GUI can be made without compiling.

The Apecis source code is included in the RRApex source code tar ball. It is split up in three parts:

apecis.{c,h}
Initializes the network core and GUI, connects GTK signals. Starts GTK main loop. Further execution is event driven.

networking.{c,h}
All code for network communication, which is, initializing the APEX communication, connecting, disconnecting, sending messages and callback functions for incoming messages.

events.c
Event handlers for the GUI signals, as well as incoming messages from the networking core.

3.4.2 Tests with Netlab implementation

Since the Netlab implementation is the only other APEX implementation we know about, it was all we had to test against. The Netlab implementation is not complete, so not many tests could be done at all. Table 1 shows the test cases and their result.

Test-case	Result	Comment
Simple XML messages in same adm. dom.	?	Unable to test.
Simple XML messages inter-adm. dom.	?	Unable to test.
Simple MIME messages in same adm. dom.	-	Not implemented.
Simple MIME messages inter-adm. dom.	-	Not implemented.
Presence: Publish	-	Not following RFC.
Presence: Notify	-	Not implemented.
Presence: Subscribe	-	Not implemented.

Table 1: Tests with Netlab implementation. *Not implemented* indicates a lack in *the Netlab* implementation, not in our.

3.5 Result

The result of our project is an APEX profile for RoadRunner, namely RRApex. As far as we know, it is the first ever, APEX implementation. Even if we mentioned a Netlab implementation earlier, that implementation proved to be incomplete and (therefore) unusable for us.

Many commercial software companies have shown interest for APEX. This fact means we might have more APEX implementations around the world, hidden behind the secrecy of the companies research laboratories.

The RRApex implementation is our contribution to the open source community. It is released under *The Blocks Public License*. RoadRunner has the same license. It is about the same as GPL but without the ”virus” effect. The license is available in Appendix B.

Since it is a profile for RoadRunner, it is used as any other RoadRunner profile. The C programmer simply includes <librr/rr.h> to use the RoadRunner core,

and <librrapex/rr-apex.h> for the APEX profile. The complete API documentation for the library is available online and can be found from:

http://www.cs.umu.se/~jon/exjobb/API

The API documentation is available in Appendix C as well.

Our implementation covers the most of RFC 3340. Some handling of options had to be left out, due to the limited time.

We have also implemented some parts of the IETF draft for APEX presence service. This can be seen as a *profile for the profile* and provides distribution of presence information. The presence code are documented in the same API description, mentioned above.

The implementations have been tested to work with itself. Due to the lack of other (complete) implementations, much of the compatibility tests have not been possible to do.

3.5.1 Limitations

Since time has been limited, we had to leave some functionality unimplemented. The interested reader might want to continue our work and implement the avoided parts mentioned in section 4.7, as well as the extensions proposed in the same section. The known limitations in our implementation are:

- We do not handle options in the way they are mentioned in the RFC. One option can for example be mustUnderstand='true'. This option should be taken care of in some sense, but is not in our implementation.
- We do not have support for multiple recipients to one message. Due to the RFC, the <recipient identity='..' /> can exist one or more times in a message header, but in out implementation it has to exists exactly one time. This means the message can only be sent to one recipient at a time.
- Our way of handling OK responses is incorrect. We send OK if the message travels over the *first* APEX channel without an error, not after it traveled *to the destination*, as the RFC specifies.

4 Discussion

4.1 19 years of application protocols

In RFC 3117[18], Marshall Rose take a look at the past 19 years of application protocol evolution. Rose talks about *Simple Mail Transfer Protocol* (SMTP), as the (almost) perfect protocol. It is simple enough to implement in few lines of code, but still powerful enough to be the core of the Internet e-mail-service.

When designing a new application/exchange protocol today, one have a few alternatives to choose among:

1. Look at existing protocols and choose the most suitable for the current needs. Reuse of HTTP is a popular example of protocol reuse.
2. Build your own exchange on top of the world wide web infrastructure. A good example here is Soap, using HTTP over port 80 to travel through firewalls, etc.
3. Use an existing protocol (like SMTP), and add the functionality you need. The *Simple Mail Transfer Protocol* (SMTP) is an example of a simple and extensible protocol, suitable as core in a new protocol.
4. Write a new protocol from scratch that has exactly the functionality you need for the current application.

Today many protocols and techniques are implemented on top of HTTP. Many organizations have firewalls, only letting TCP port 80 through. This fact makes HTTP very suitable, even if it is a dirty solution.

In RFC 3117, Rose continues the discussion by talking about taking the needs to a basic level. The core needs of a generic application protocol can be summarized as:

1. **A connection oriented communication.**
   When designing Connection-less protocols (like DNS), it might be a better idea to implement reliable communication on the application protocol level. Rose leave that case and focus on connection orientation instead.
2. **Request/response principle for message passing.**
   Most protocols today uses this principle; *a request* (question) is answered with *a response* (answer). The protocols of today are often limited to client/server behavior, which is, the client always send request and the server response. Since this is an unnecessary limitation, Rose prefers the *peer-to-peer* paradigm.
3. **Support for asynchronous message exchange.**
   Since Rose wants to leave out the limitations from the client/server model, he adds support for an asynchronous message passing pattern.

Issue	Description
Framing	What is the beginning and where is the end of a message?
Encoding	What common way do we have to represent data?
Reporting	What is the status? What went wrong?
Asynchrony	How do we deal with independent exchanges?
Authentication	Identification and verification of peers.
Privacy	Protection against sniffing and modification.

Table 2: The problems a generic application protocol have to deal with.

This gives a feeling of the nature of a generic protocol. Let us look at the problems a such protocol have to deal with. Table 2 show a summary of the issues.

There seem to be consensus about five properties of a well-designed application protocol. It shall be:

Scalable
The classic client/server model is not a good example, since too much of the functionality is expected from the servers. One server is supposed to handle many clients, an unfair model not tolerating the number of clients to grow. Moving on to peer-to-peer, by moving functionality to the peer nodes, help us keep the protocol more scalable. Depending on underlying transport protocol, it might be idea to use only one connection on transport level, to provide many logical channels on the application level. For example, TCP uses adaptive methods for tuning the speed of outgoing data. Finding a good transmit speed takes time. Reusing an existing TCP connection can save us this (overhead) time.

Efficient
How efficient a protocol is depends on the choice of data encoding and of cause, what we mean by *efficiency*. For example, HTTP/1.1 uses octet counting, efficient in the meaning we do not need to search for control data in the payload. MIME uses textual headers, efficient in the sense that we easily can add meta-data, and we do not need to know the size of the payload before sending starts.

Simple
Common and simple functionality shall be easy to implement. Complex functionality shall be possible to implement.

Extensible
Every protocol some time reach the point when we want to add functionality. In poorly designed protocols, this is not possible. A good protocol allow us to add arbitrary functionality whenever we need it.

Robust
When designing a protocol as efficient as possible, it might impact the robustness. Even if *defaults* in the protocols can save some bytes in the

messages, error often occur in the implementations. A well-designed protocol is efficient without affecting the robustness.

4.2 The nature of BEEP

Rose continues in RFC 3117 by choosing the best (in his opinion) possible technique for each ”problem” above. The result is called the BXXP (Blocks eXtensible eXchange Protocol, later BEEP). The foundations of BXXP are presented in table 3.

Mechanism	**Choice for BXXP**
Framing	Counting, with trailer
Encoding	MIME, defaulting to text/xml
Reporting	3-digit and localized textual diagnostic
Asynchrony	Channels
Authentication	Simple Authentication and Security Layer (SASL), RFC 2222
Privacy	Transport Layer Security Protocol (TLS), RFC 2246 or SASL

Table 3: The foundations of BXXP (BEEP).

4.3 BEEP as a common framework

As we have seen above, application protocol designers/programmers deal with about the same problems for every new protocol they design. Marshall Rose's discussion about a general solution to the problems is important and relevant. Even if he has to do some compromises concerning the components in BEEP, it is still the best possible. A compromise we are not relay confident with, is the support for MIME. We think XML is extensible enough to be the only choice for BEEP. XML can be used to transfer all kinds of data, even the data Rose intend to use MIME for.

Even thought BEEP is a new standard, we today have stable and efficient implementations such as RoadRunner, available for everyone to use in new implementations. RoadRunner has a number of interesting users, working with implementations like Intrusion Detection eXchange Protocol (IDXP) and XML-coded Remote Procedure Calls (XML-RPC).

4.4 Add APEX to BEEP, what do we get?

BEEP is a good complement to ordinary TCP/IP sockets. As we have seen, BEEP provides a lot of services to the application protocol programmer, making design of application protocols more efficient.

But for some applications, we can go one step further and add *end-to-end user routing*. This is simply what APEX *does*. Since this end-to-end user routing is not only used for instant messaging, but also a number of other applications,

(e.g. remote conference tools and organization message-boards), it is appropriate to implement this service on a library level, like APEX.

Thanks to the APEX ”plug-in” services (the so called well-known endpoints), APEX can be extended and developed to fit new application needs. This is a very important difference from many of the application protocols used today on the Internet. Hopefully, it is this extensibility that will keep APEX alive as the best alternative in the future.

4.5 The need of an instant messaging standard

Since the techniques for instant messaging used today are incompatible between each other, there is a large need for an open standard. Users of the ICQ program cannot communicate with users on the Microsoft® Instant Messenger network, and so on.

Even if large players like AOL Instant Messenger and ICQ today uses the same application protocol, it is still not an open standard, available for everyone to use.

Thought, it is important to note that APEX is not meant to be just another instant message protocol, but a powerful framework with loads of interesting possibilities, all independent of implementation, just like the world wide web looks today.

4.6 APEX as part of an instant messaging standard

As we have mentioned before, APEX itself is not a standard for instant messaging. Such a standard need to specify a message format for the passed messages. A good idea would be to specify an XML Document Type Definition (DTD), for the message passing standard.

Hopefully, the APEX Working Group will present a such DTD in the near future. When we have a such DTD, we are not far away from a complete system. Thanks to well-specified WKEs such as the IETF draft *The APEX Presence Service*, we have good support for many of the necessary parts in an instant messaging system. These well-defined parts can be implemented as library functions and therefore be application-independent.

In the future, we hope to see a such open standard for instant messaging. We cannot promise it is going to happen, but if it does, it will be a powerful system built on a flexible core of gold.

4.7 Future work

The APEX Working Group is working on draft documents for more APEX services. As mentioned earlier, three of the documents have reached RFC status. We have implemented one of them, The APEX Core. This also means we have two unimplemented services at RFC status:

- ⋆ RFC 3341 [19]: *The Application Exchange Access Service*
  Describes the well-known endpoint apex=access. The access service is used to control both the use of APEX *relaying mesh* and other APEX services.
- ⋆ RFC 3342 [11]: *The Application Exchange Option Party Pack, Part Deux!*
  Options are used to alter the semantics of the core service. This memo defines various options to change the default behavior of APEX's relaying mesh.

The RFCs mentioned above are interesting and necessary extensions to the APEX core. To get a complete APEX toolkit, both RFCs should be implemented.

RFC 2779 defines a number of requirements for an instant messaging system. An interesting task would be to design a system based on APEX, implementing all the requirements from RFC 2779. A part of the task can be to design a simple but complete XML Document Type Definition (DTD) for the system.

When other APEX implementations are finished, a lot more testing should be done, to guarantee the compatibility.

Further work could even include more work on the RRApex profile, support for access control service as part of the core profile and last but not least, further development of the RRApexPresence library.

RFC 3340 specifies the APEX addressing to use the SRV [12] algorithm, fetching special SRV records from the DNS system. This is not implemented in our code.

4.8 Conclusion

Throughout our work with the BEEP/APEX standards and the RRApex implementation, we have experienced the power of a rising new solution framework for an old problem. We believe the new approach is a well-designed system of components, each providing an important piece of abstraction. BEEP/APEX can be the *once for all* solution to the common application protocol designer problems discussed in section 4.1. Powerful implementations of BEEP are available today, and the APEX components are taking form. It is now up to application protocol designers and programmers around the world, to accept or reject the new standard. It is difficult to speculate whether they will accept or not, many of the experienced designers/programmers are conservative and not willing to accept new standards before they are convinced.

5 Acknowledgment

We would like to thank the following people.

- ⋆ Jonas Borgström
  Even thought Jonas is a student two years under us, he is still our BEEP, RoadRunner and open source (in general) god!
- ⋆ Pedher Johansson
  Our advisor! Thanks for all the feedback!
- ⋆ Daniel Lundin
  Daniel wrote RoadRunner together with Jonas (above). Daniel is the illutrator behind Figure 2 and 3.
- ⋆ Jerry Eriksson
  A busy man who helped us set up the task and search founding for the project.
- ⋆ Marcus Bergner
  The thanks to Mackan is due to all the general hacking help at all stages of our work.
- ⋆ All the open source programmers...
  ...out there, providing free powerful software and libraries. Many parts of our project are built using good open source libraries. Peace.

Älska varandra!

References

[1] America online® instant messenger™. Web Site, 29 Nov. 2002. http://www.aim.com/.

[2] Codefactory roadrunner. Web Site, 10 Oct. 2002. http://rr.codefactory.se/.

[3] Extensible markup language (xml) 1.0 (second edition). Web Site, 12 Nov. 2002. http://www.w3.org/TR/REC-xml.

[4] Jabber software foundation. Web Site, 20 Dec. 2002. http://www.jabber.org/.

[5] Microsoft msn messenger. Web Site, 18 Dec. 2002. http://messanger.msn.com/.

[6] Mirabilis icq. Web Site, 28 Nov. 2002. http://www.icq.com/.

[7] The gnu compiler collection. Web Site, 3 Jan. 2003. http://gcc.gnu.org/.

[8] Internet assigned numbers authority. Web Site, 3 Jan. 2003. http://www.iana.org/.

[9] The internet engineering task force. Web Site, 3 Jan. 2003. http://www.ietf.org/.

[10] Freed, N., and Borenstein, N. Multipurpose internet mail extensions (mime). RFC 2045, 30 Nov. 1996.

[11] G. Klyne et. al. The application exchange (apex) option party pack, part deux! RFC 3342, 30 July 2002.

[12] Gulbrandsen, A., Vixie, P., and Esibov, L. A dns rr for specifying the location of services (dns srv). RFC 2782, 30 Mar. 2000.

[13] M. Day et. al. Instant messaging / presence protocol requirements. RFC 2779, 01 Feb. 2000.

[14] M. Rose and G. Klyne and D. Crocker. The apex presence service. IETF Draft, 14 July 2002.

[15] Mattis, P., Kimball, S., and MacDonald, J. Gtk+ f.a.q. Web Site, 12 Nov. 2002. http://www.gtk.org/faq.

[16] R. Movva and W. Lai. Msn messenger service 1.0 protocol. IETF Draft, 1 Aug. 1999.

[17] Rose, M. T. The beep core. RFC 3080, 30 Mar. 2001.

[18] Rose, M. T. On the design of application protocols. RFC 3117, 1 Feb. 2001.

[19] Rose, M. T., Klyne, G., and Crocker, D. The application exchange (apex) access service. RFC 3341, 30 July 2002.

[20] Rose, M. T., Klyne, G., and Crocker, D. The application exchange core. RFC 3340, 30 July 2002.

A Tools and libraries

This appendix will describe all the wonderful libraries we have used for the implementation of the APEX library and the test application.

A.1 GLib

GLib is a library of useful functions and definitions available for use when creating GDK and GTK applications. It provides replacements for some standard libc functions, such as malloc, which are buggy on some systems.

It also provides routines for handling:

- Doubly Linked Lists
- Singly Linked Lists
- Timers
- Error Functions
- A Lexical Scanner
- String Handling
- Hash Tables
- Threads

The RRApex implementation is multi-threaded and uses the GLib thread-pool. This kind of functionality is really useful and have saved us a lot of time.

The GLib API documentation is available online at:
http://developer.gnome.org/doc/API/2.0/glib

A.2 GTK+

The Gnome Toolkit (GTK), is a powerful set of platform-independent widgets for use in graphical user interfaces. GTK together with GLib are the core components for Gnome, an attempt to build a complete open source desktop environment.

More information about GTK and Glib can be found in the the GTK+ F.A.Q. [15]. The GTK API documentation is available at:
http://developer.gnome.org/doc/API/2.0/gtk

A.3 Libxml

Libxml is the XML library of Gnome. It is used to parse and compile XML documents. Since all APEX messages are in XML format, the Libxml functions are used in all message processing.

Libxml first...

Libxml is freely available from http://www.xmlsoft.org. The API documentation is available at
http://www.xmlsoft.org/html/libxml-parser.html

A.4 Other tools and environment

All source code are written in vim, a powerful vi clone. Vim is freely available from http://www.vim.org.

The code is compiled using the GNU Compiler Collection, GCC, a free collection of compilers available from [7].

All development and testing have been done on GNU/Linux systems. GNU/Linux, also known as Debian, is a free Unix-like operating system, available from:
http://www.debian.org

RoadRunner Debian packages are available for quick and easy installation. To simplify compilation, we use Make, available from:
http://www.gnu.org/software/make/make.html

To generate Make-files, Automake is used:
http://www.gnu.org/directory/automake.html

The API documentation is generated using DocBook. More information about DocBook can be found at http://www.docbook.org. This report is typeset using LaTeX.

Sköt om varandra!

B License for the RRApex library

C API documentation for RRApex and RRApexPresence