Context Management in Mobile Environments

Ricky Robinson
Supervisor: Dr Andry Rakotonirainy

October 13, 2000

Abstract

The proliferation of mobile devices has created the need for applications which can operate in dynamic conditions. This implies the requirement for a reactive platform that can make informed decisions about how to respond to changes to device capability, user preferences, enterprise policy and many other environmental factors. These capabilities, preferences, policies and environmental attributes can be collectively referred to as ‘context’. Context can be used in many and varied ways. Context can be used as the basis by which an adaptation manager or algorithm decides to modify content or application behaviour. It could also conceivably be used directly by an application that pushes relevant information to a user based on the user’s present situation. A Context Manager provides the ability to collect, collate and process context information. A Context Manager should be flexible and extensible. This means it should be able to process context information from many different sources, and that new sources of context information can be added at will. Such sources of context information include a location awareness module, modules that detect modifications and additions to hardware and software, and even a component that provides data about enterprise policy. This project investigates several areas of context management. First of all it seeks to identify the strengths and weaknesses of Composite Capabilities/Preference Profiles (CC/PP), a proposed W3C standard for describing device capabilities and user preferences, in describing context information. In particular, the ability of CC/PP to describe context information beyond device capabilities and user preferences is investigated. A second goal of this project is to design and implement a mechanism by which context information can be updated and distributed via Elvin, an event notification system. This paper also shows how event correlation could be used to implement several aspects of a context manager in a simple and elegant fashion.

Acknowledgments

I would like to thank my supervisor, Dr Andry Rakotonirainy, for his guidance and the support he has given throughout the year.

Also, I would like to acknowledge the entire m^3 group for making me feel like part of the team and for showing such enthusiasm towards their work.

Last of all, I want to thank my Mum, Dad, and brother, Nigel, for the encouragement they have given me throughout this degree, and especially over the last year. I couldn't have done it without them.

Contents

Table of Figures

1 Introduction

Being able to perform some level of adaptation in a mobile environment is a key factor in providing users with applications that will work in situations where device capability and other attributes are constantly changing. Traditional desktop applications are not suited to the fluidity of mobile environments and simply will not function in a manner that is acceptable to the user. If some kind of behavioural or content adaptation could be performed, then applications would continue to provide useful information to a mobile user. The central idea is to take into consideration the current state of the device, network, user preferences and environment and provide content that is optimised for this situation. For example, due to a low bandwidth network it may not be viable to send full colour images to a mobile user, but it may be viable if the user migrates to a higher bandwidth network sometime in the future. In fact, this information could be further harnessed to push useful information to the user, or to customise information to the user's present situation. It must be emphasised that context information can be used for purposes other than transcoding (translating content from one form to another).

These requirements imply the need for a management component whose responsibility it is to collate all available context attributes, and provide these to the adaptability manager (or indeed any other component that is interested in context). This project describes a management component that is capable of managing context information from many different sources, such as a location awareness module, a hardware awareness module and policy manager. In particular, this document proposes a mechanism by which context information can be described and updated in a mobile environment.

2 Project Description

This project has several aims.

There are many different definitions of context throughout the many fields of computer science. The first aim of this project is to define what context means in the area of enterprise mobile computing environments.

Another goal is to identify the strengths and weaknesses of CC/PP [10, 11, 12] to describe context information in a mobile environment.

Device capabilities and user preferences are only two sources of context information. The next aim of this project is to discover whether CC/PP can be used to describe context from many different sources. For example, the projects and people within an organisation constitute another useful source of context information.

The possible relationships between attributes that make up context are examined. It is predicted that there will be circumstances in which certain attributes should not exist together within a profile. It is also foreseen that there will be instances where one attribute can exist only if a certain other attribute is present. Cardinality constraints will also need to be specified for certain attributes. This project shows some of the possible ways to describe these relationships and constraints using RDF. An RDF schema specifying these constraints is presented, along with a validator capable of validating these constraints.

CC/PP imposes no constraints on the protocol used to exchange CC/PP profiles. However, the CC/PP community has proposed only one protocol at this point in time. This protocol is called CC/PP exchange protocol [3] and is based on the HTTP extension framework [4]. Another aim of this project is to propose a protocol based on the DSTC's Elvin notification service [5]. Two separate implementations are provided. One version is based on an early CC/PP implementation developed by the author of this paper, and another is based upon the more recent CC/PP implementation developed by the m^3 team. It is expected that the latest version would be robust enough to be used within the m^3 project.

Another goal is to show how context classes can be implemented using Elvin correlations. A context class is made up of one or more attributes, each of which may take on a range of acceptable values. If any of the values falls out of the specified range, then the context class changes. Correlations allow varying levels of granularity for context change notifications. This will be explained thoroughly in the section about correlations.

Once these aims have been achieved, the design for a context manager is presented.

3 Context Management

3.1 What is Context?

Context information is used in many and varied computer science projects. The AI community, for instance, is especially interested in context because they are concerned with being able to deduce the intended meaning of words or other data in different situations. There are a number of slightly different definitions of context in the computer science fraternity. However, the differences are largely cosmetic, resulting from the unique applications of context. While each definition may focus on the particular application of context, most definitions seem to indicate that context information is that which can be gathered from the surrounding environment, and although this information may not relate directly to the task at hand, it may help to lift ambiguity [13].

In this report, the word ‘context’ will mean any information at all that can help determine the situation of an object, where an object can be a user, a device or any other entity being tracked by the system. The situation of an object is ultimately what a context manager is required to determine. Indeed, Dey et al in [21] define context as ‘any information that can be used to characterise the situation of an entity, where an entity can be a person, place, or physical or computational object.’

In a mobile environment, context data is stored mainly to so that decisions can be made about how to adapt content or behaviour in some way. Context can be used as provided by context sources, or it can be manipulated into a form more useful for certain purposes. In all cases, regardless of whether manipulation takes place, context is used to clarify the situation of a user, device or other object.

3.2 A Mobile Enterprise Architecture

There are a number of factors that make mobile computing different from desktop computing. Mobility means that attributes associated with devices and users are constantly changing. These changes mean that content and behaviour of applications should be adapted to suit the current situation. Sometimes these changes are made out of necessity. In these situations, applications would cease to function at all. An example is if the bandwidth should suddenly drop, a graphics intensive web application should start sending content that consumes less bandwidth. On other occasions content and behaviour adaptation takes place in order to provide a service that is tailored to the user’s present situation. For example, given location and time context and a user’s culinary preferences, the platform could point out to the user that it is almost time for dinner and that there is a good Italian restaurant around the corner [1].

There are several ways in which content and behaviour can be adapted. One way is to build each application with context awareness, and allow them to adapt content and behaviour accordingly. Another way is to provide an underlying platform that attempts to perform most of the adaptation transparently to the client and the application server.

The Mobile Enterprise Architecture proposed by the m^3 team within the DSTC is representative of such a platform. It is described here to provide a concrete example of a mobile enterprise platform, and to show the role of a Context Manager in such an architecture.

3.3 m^3

The goal of the m^3 architecture is to ‘maximise the enterprise productivity while minimising constraints brought by mobility’ [2]. In other words, an enterprise user

should not be prevented from working just because they are away from the desktop or because mobile devices have different capabilities to desktop computers.

To accomplish this goal, the m^3 project defines an architecture comprised of several management components, each of which performs a particular task.

3.3.1 Context Manager

First of all, there is a Context Manager, aspects of which is the focus of this paper. Its role is to maintain information pertaining to mobile devices, users, the environment around each mobile device and any other context information deemed relevant. All this information is provided by a set of 'awareness modules'. An example of an awareness module is a location manager that tracks the location of users. Another example is a mobile device itself, which can provide information about its present capabilities. All this context information is collated and made available to the other management components. These requirements lead us to three aspects of context management:

1. Context sensing. The way in which context data is obtained.

2. Context representation. The way in which context information is stored and transported.

3. Context interpretation. The way in which meaning is obtained from the representation.

Each of these are dealt with by this paper.

3.3.2 Policy Manager

The Policy Manager is responsible for controlling mobile enterprise policies. The introduction of mobility to an enterprise brings with it a number of issues which are not seen in traditional computing environments. Organisations generally have policies regarding the disclosure of certain pieces of information. For example, some hard copy documents may be viewed at specially designated locations only, such as the office of the CEO. These kinds of policies must be transferable to a computing environment, and in particular, a mobile computing environment. The m^3 Policy Manager assigns users to roles, and each role has permissions, prohibitions and obligations associated with it. In general, a policy in combination with current context information will determine what actions a user is allowed to perform, or what actions a user is obligated to perform.

3.3.3 Security Manager

The Security Manager provides secure session connection management between the front and back ends of the mobile platform. Depending on the security policies of an organisation, some content may not be viewable if the security requirements cannot be met.

3.3.4 Adaptability Manager

The Adaptability Manager is responsible for adapting content, behaviour and other aspects according to context and policy. The Adaptability Manager may take any number of actions depending on the information passed to it by the Context Manager. This information may or may not be in the form of RDF. The most obvious action to perform is to transcode content so that it may be viewed on a particular device. Other actions might include appending location specific information to documents.

3.3.5 MEADL

The Mobile Enterprise Architecture Description Language provides a programming interface that acts as a communications bus by which the above components may communicate control information. Not all inter-component communication needs to take place via MEADL. For example, the Adaptability Manager may obtain context information directly from the Context Manager.

4 Literature Survey

This literature survey is comprised of reviews of papers and projects that have some relevance to context management in mobile environments. While these papers do not describe existing context managers, they each lend something to the field of context management.

4.1 Composite Capabilities/Preference Profile

4.1.1 Description

Composite Capabilities/Preference Profiles (CC/PP) [10] is a proposed W3C standard for describing device capabilities and user preferences. Particular attention is being paid to wireless devices such as mobile phones and PDAs.

In practice, this model is based on RDF and can be serialised using XML. RDF is can be seen as a way of representing knowledge. The following diagram shows an example RDF tree which models a CC/PP instance.

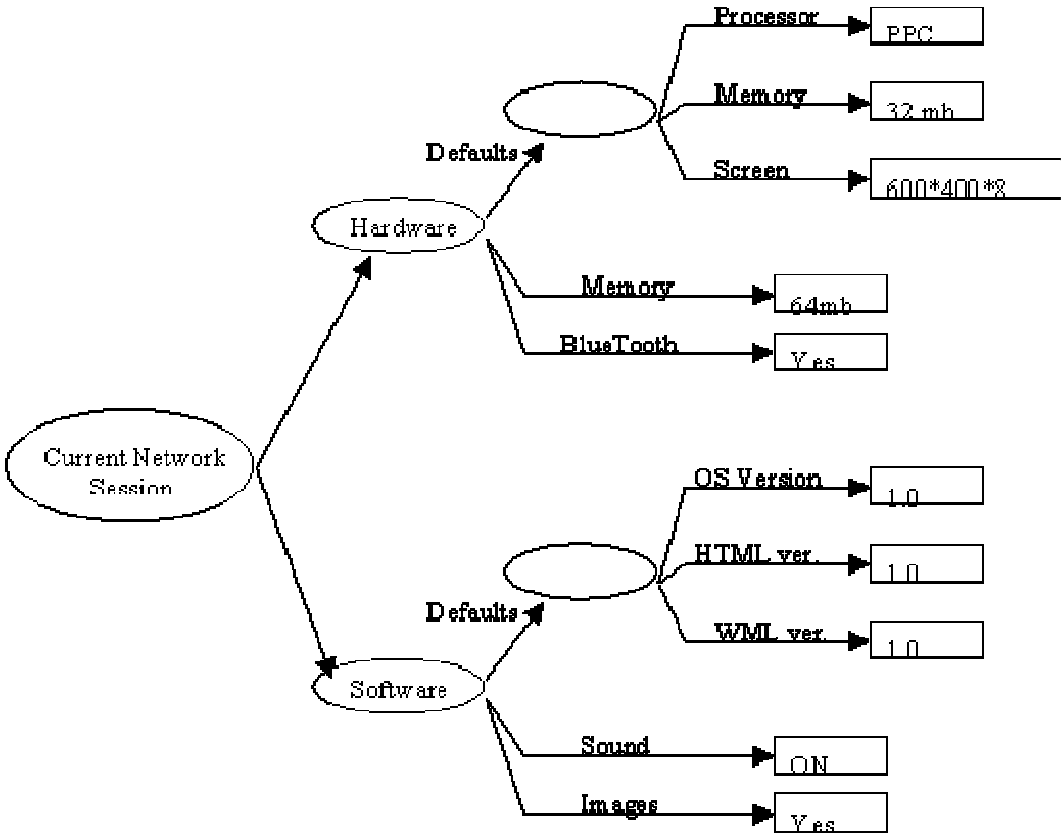

Figure 1 Example RDF graph of a CC/PP

The root of the tree is the current session associated with this instance. CC/PP only expresses information related to the current session. All information that is to be used to adapt content must be present somewhere in the tree.

```
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:ccpp="http://www.w3.org/2000/07/04-ccpp#"
     xmlns:uaprof="http://www.wapforum.org/UAPROF/ccppschema-
19991014#">

  <Description about="http://www.example.com/MyProfile">
    <ccpp:component>
      <Description about="http://www.example.com/TerminalHardware">
        <type
resource="http://www.example.com/Schema#HardwarePlatform" />
        <ccpp:Defaults>
          <Description>
            <type
resource="http://www.example.com/Schema#HardwarePlatform" />
            <uaprof:CPU>PPC</uaprof:CPU>
            <uaprof:ScreenSize>320x200</uaprof:ScreenSize>
          </Description>
        </ccpp:Defaults>
        <uaprof:ScreenSize>640x400</uaprof:ScreenSize>
      </Description>
    </ccpp:component>

    <ccpp:component>
```

```
      <Description about="http://www.example.com/TerminalSoftware">
        <type
resource="http://www.example.com/Schema#SoftwarePlatform" />
        <ccpp:Defaults>
          <Description>
            <type
resource="http://www.example.com/Schema#SoftwarePlatform" />
            <uaprof:OSName>EPOC</uaprof:OSName>
            <uaprof:OSVersion>2.0</uaprof:OSVersion>
            <uaprof:OSVendor>Symbian</uaprof:OSVendor>
          </Description>
        </ccpp:Defaults>
      </Description>
    </ccpp:component>

    <ccpp:component>
      <Description about="http://www.example.com/Browser">
        <type resource="http://www.example.com/Schema#BrowserUA" />
        <ccpp:Defaults>
          <Description>
            <type resource="http://www.example.com/Schema#BrowserUA"
/>
            <uaprof:BrowserName>Mozilla</uaprof:BrowserName>
            <uaprof:BrowserVersion>5.0</uaprof:BrowserVersion>
            <uaprof:CcppAccept>
              <Bag>
                <li>text/plain</li>
                <li>text/vnd.wap.wml</li>
              </Bag>
            </uaprof:CcppAccept>
          </Description>
        </ccpp:Defaults>
        <uaprof:CcppAccept>
          <Bag>
            <li>text/plain</li>
            <li>text/vnd.wap.wml</li>
            <li>text/html</li>
          </Bag>
        </uaprof:CcppAccept>
      </Description>
    </ccpp:component>
  </Description>
</RDF>
```

Listing 1 A serialised profile

The listing above shows how CC/PP can be serialised. It is a simple profile that describes the attributes of a mobile device. In this example, all components and attributes are described inline. That is, the attributes and their values are contained within this profile itself. CC/PP also allows profiles to contain indirect references to other profiles. These profiles will normally contain default values for attributes. The following listing is the same as that above with the exception that the listing below utilises indirection..

```
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:ccpp="http://www.w3.org/2000/07/04-ccpp#"
```

```
     xmlns:uaprof="http://www.wapforum.org/UAPROF/ccppschema-
19991014#">

  <Description about="http://www.example.com/MyProfile">
    <ccpp:component>
      <Description about="http://www.example.com/TerminalHardware">
        <type
resource="http://www.example.com/Schema#HardwarePlatform" />
        <ccpp:Defaults
rdf:resource="http://www.nokia.com/profiles/2000k" />
        <uaprof:ScreenSize>640x400</uaprof:ScreenSize>
      </Description>
    </ccpp:component>

    <ccpp:component>
      <Description about="http://www.example.com/TerminalSoftware">
        <type
resource="http://www.example.com/Schema#SoftwarePlatform" />
        <ccpp:Defaults
rdf:resource="http://www.Symbian.com/profiles/EPOC" />
      </Description>
    </ccpp:component>

    <ccpp:component>
      <Description about="http://www.example.com/Browser">
        <type resource="http://www.example.com/Schema#BrowserUA" />
        <ccpp:Defaults
rdf:resource="http://www.nokia.com/profiles/Mozilla" />
        <uaprof:CcppAccept>
          <Bag>
            <li>text/plain</li>
            <li>text/vnd.wap.wml</li>
            <li>text/html</li>
          </Bag>
        </uaprof:CcppAccept>
      </Description>
    </ccpp:component>
  </Description>
</RDF>
```

Listing 2 A serialised profile using indirection for default values

Profiles can be stored in a CC/PP repository. By using a repository, the client need not send its entire profile with each request for content. Instead, it sends only the differences since the last request. This implies the use of some kind of session (stateful) protocol.

4.1.2 Relevance and Evaluation

Since the evaluation of CC/PP is a major component of this thesis, it is given thorough treatment in later sections of this document. The relevance of CC/PP to context management is provided here.

CC/PP is intended for modelling context related to device specifications and user preferences. The use of XML/RDF as a description syntax means that profiles can be

transported to a wide variety of machines. It also means that existing vocabularies can be extended and new vocabularies added with the use of XML namespaces.

To be of real use in an extensible context manager, CC/PP would need to be capable of describing many different kinds of context information. It would be required to store location context, enterprise context (projects, roles and other information associated with an enterprise) and any other information that might help to determine an object's situation. There would also need to be a mechanism for storing context histories.

This paper addresses some of these issues, and shows that CC/PP can describe arbitrary contexts.

4.2 Odyssey

4.2.1 Description

Odyssey [7] is a platform that supports application-aware adaptation. In the application-aware model, the underlying platform (in this case Odyssey) is responsible for monitoring Quality of Service levels, but the applications are responsible for performing the required adaptation when notified by the platform.

Odyssey is realised in two parts. The major part consists of the Wardens and the Viceroy and is implemented in user space. The second part consists of an interceptor, which is placed inside the NetBSD kernel. The interceptor intercepts all system calls and redirects operations on Odyssey components to the Viceroy. All other calls are forwarded to the kernel.

The Viceroy is responsible for monitoring resources. These resources are shown in the table below.

Resource Type	**Unit of measurement**
Network Bandwidth	bytes/second
Network Latency	microseconds
Disk Cache Space	kilobytes
CPU	SPECint95
Battery Power	minutes
Money	cents

Table 1 Resources monitored by Odyssey

Applications can specify a ‘window of tolerance’ for each of these resource types. If the value of a resource falls outside the window of tolerance, then the application is notified of this. The application can then specify an adjusted window of tolerance for the resource. An adjusted window of tolerance usually means that the application is willing to accept a different level of content fidelity. Fidelity is a measurement of the difference between content as is presented by the mobile device, and the original content quality as stored on the server. A small difference corresponds to high fidelity. A drop in available resources means the applications must either accept lower fidelity or (presumably) inform the user that content can no longer be delivered.

Wardens perform all communication with the back end server. Wardens are content type specific. That is, a separate warden exists for each content type. New wardens can be added to the platform as they are developed. So there might exist a video warden, a streaming audio warden, a web warden or a database warden. Wardens exist because each content type has different allowable fidelity levels, and if there is a change in resource availability, different content types must be handled differently.

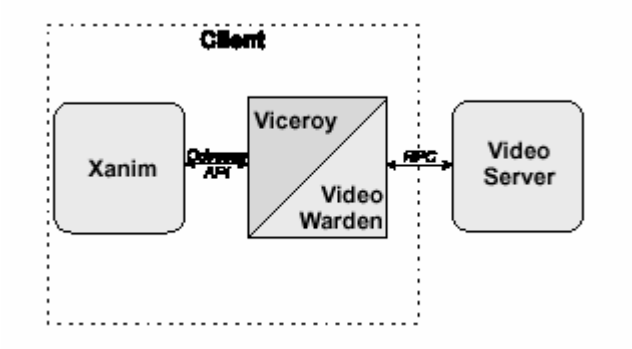

Figure 2 An example application using Odyssey

Wardens define the allowable fidelity levels, and communicate these to the Viceroy. If applications register an unknown fidelity level, the Viceroy informs the application of this fact.

4.2.2 Relevance and Evaluation

One can view the Viceroy component of Odyssey as a minimal context manager. The Viceroy is responsible for monitoring a set of Quality of Service resources. These resources can be seen as a subset of the context information monitored by a context-aware platform.

Just as applications can register with Odyssey to be notified when resource values fall outside a given window of tolerance, an adaptation manager can register with a context manager to be informed when certain context attributes change.

Odyssey provides us with an example of a minimal context-sensing platform. While sensing is limited to the set of six attributes shown in Table 1, it is a good example of the way in which context can be used to help adapt content in an optimal fashion.

4.3 PRAYER

4.3.1 Description

The PRAYER system [6] was an attempt to provide a low-level adaptable architecture for mobile devices. Various contextual attributes were monitored so that informed decisions could be made about what kind of adaptation should take place at any point in time. However, the set of attributes maintained by PRAYER was static. This restriction means that any adaptation that the platform performs is bounded by this finite source of context information.

There are two broad categories of context information in the PRAYER system: those attributes related to the capabilities of the hardware platform and those attributes related to network quality of service. Examples include connection cost, access protocol, latency, computing power, battery power and Operating System.

An application may specify a range of acceptable values for each of these attributes. Applications provide two callback functions for the PRAYER platform. The first function is invoked when a QoS parameter falls within the acceptable range. The second function is invoked when a QoS parameter falls outside the acceptable range. In this way, the application can decide what action to take when QoS falls below a specified value.

Applications that use the PRAYER platform are logically divided into regions. Each region within the application may need a different QoS in order to perform a task. QoS may be renegotiated with the platform in each transition from one QoS region to another.

Context management in PRAYER is limited to finding QoS classes from each of the above mentioned attributes. At any given time, the platform is within a single QoS class. Since new QoS parameters are never added, and existing ones never removed, QoS management is simple in the PRAYER system.

PRAYER is similar to Odyssey in a number of ways. First, they both opt to notify applications of changes in resource availability rather than attempting to provide a transparent platform. Second, they both monitor an inextensible set of QoS resources.

4.3.2 Relevance and Evaluation

The PRAYER system provides a concrete example of an adaptive architecture for mobile environments. While the context managed by PRAYER is limited to Quality

of Service characteristics and hardware capability, it does show that it is possible to group together several, albeit predefined, attributes and ranges for the attributes, and to categorise the current situation into one of these groups. This report will also make use of these groups. They will be referred to as context classes. An attempt is made to show how arbitrary context classes can be defined.

Although PRAYER does not contain a context manager, there are several ideas from PRAYER that can be extended to a context management system. One such idea is the notion of Quality of Service classes. Instead of dealing with each QoS resource separately, several classes are formed by assigning ranges of acceptable values for each resource. The application can request a particular class and be notified when the QoS class changes.

A context manager could use a similar idea to group several context attributes together along with ranges for those attributes into logical categories called ‘context classes’. An adaptation manager could then be notified when context falls into or out of a context class. This removes the need for the adaptation manager to concern itself with a large number of attributes if it doesn’t need to know details.

4.4 Savoir

4.4.1 Description

Savoir [1] is one version of the Cyberguide application developed at Georgia Tech. Cyberguide can be thought of as a replacement for tourist brochures. Ordinarily, tourist brochures provide information about places to see, restaurants to eat at and events to watch. However, the tourist brochure is not intelligent in that it does not know where or when it is being perused by a tourist. Cyberguide, on the other hand, makes use of location and time context so that it can make intelligent choices about what to show the user. For example, if it is late in the day, then the guide might show a list of places to eat dinner. If a tourist is close to a museum, then the guide might suggest that the tourist visit the museum.

Savoir is a voice only interface to Cyberguide.

Savoir was interesting because it specified how context information could be managed so as to make better decisions in the future. It also uses positional information to guess what kinds of commands a user is likely to issue. Both long term and short term context histories play a roll here. Savoir distinguishes two kinds of context: physical context (location, time etc) and informational context (based on the fact that users are likely to make requests related to information they have heard recently).

Because the system is voiced based, it needs to select a relevant grammar such that continuous speech may be interpreted correctly. Obviously, the grammar chosen should contain words relevant to the current time, location and recent topics.

Another interesting choice made by the Georgia Tech team is to maintain a history of commands that have never been issued before. Each command can be linked to context information such as the current location so that next time this location is visited, the command can be loaded as part of the relevant grammar. Also, by remembering some types of context for short periods, it is possible to resolve 'tricky' statements. For example, if the user says 'Get Dave's e-mail address' and then says 'Send this information to him', by remembering the word 'Dave' from the initial command, it is possible to resolve the pronoun 'him' in the second command. Without this context history, there would be no way to establish that the user means 'Send this information to Dave'.

4.4.2 Relevance and Evaluation

Savoir does not show *how* to interpret context, but it does show the *results* of context interpretation. By interpreting context correctly, the situation of an object can be determined more accurately and this allows applications to deliver content that has is more useful in the current situation. It also shows the benefit of keeping both long-term and short-term context histories.

Although the context manager discussed in this report is not especially concerned with voice context, there are some parallels in the way context information is managed in these two projects. For example, in both Savoir and the context management system discussed here, multiple sources of context information need to be collated in order to make decisions. In order to make good decisions, Savoir takes into account time, location and informational context. A context manager makes use of a greater variety of context sources but the principle remains the same.

Savoir and the Cyberguide project in general shows how context can be used to customise the user's experience. Based on context, Savoir can make the decision to push certain kinds of information to the user. That is, relevant information can be sent to the user without the user having to request it. Savoir shows that transcoding is not the only use for context information.

4.5 Active Badge

4.5.1 Description

Location information can play an important role in an adaptive mobile architecture such as m^3. For example, if the user's location is known, then content specific to that location can be sent to the user without the user having to request it. Location

information can also be used to coordinate the activities of several users. Because of this, location represents one of the most powerful pieces of context information.

The Active Badge System [8] was developed in an effort to show how personnel can be tracked in an office environment. Each person is given a small badge which emits infra-red signals. These signals are captured by sensors that have been strategically placed around the office space. This system is then able to track people as they move around inside the building.

ORL has found a number of useful applications for Active Badge technology. For instance, by placing badges on workstations as well as users, the workstation can be automatically personalised for a user as the user approaches it. Another novel use for Active Badges is to place badges on computer components such as wireless keyboards. When a user approaches a workstation while holding a wireless keyboard, the workstation configures itself to accept input from the wireless keyboard.

The location server itself has four distinct layers.

The first layer is the Network Control. It gathers raw location data by polling all the sensors in the building. This layer removes any erroneous data. It is possible to adopt a biased polling strategy whereby sensors that are known to have seen recent activity are polled more often than those sensors that have been quieter.

The raw data from the Network Control layer is given to the Representation Layer. It is at this layer that the raw data is timestamped. The location information, the timestamp and the ID of the badge that has been sighted form a triple. The data is now in a form that can be used for higher level processing.

The Data Processing layer makes decisions about which triples to keep and which to discard. A badge's location may not have changed, and in this case the triple could be discarded. An enormous amount of data can be generated by the Active Badge System, and this is why the Data Processing layer must make decisions about which triples actually contain useful information. This layer can produce location histories, or a location trace.

The final layer is simply a presentation layer. The Display Interfaces can display data in graphical or textual form depending upon the application. With access to other kinds of information, such as the floor plan of a building, it would be possible to build a system that predicts the movements of people.

The Active Badge Location System shows that it is possible to build applications that are aware of their surroundings. By using a user ID in conjunction with location information, the Active Badge system can help applications to adapt in ways that are useful to a roaming user. Workstations can adjust themselves for particular users

automatically and users can be followed by multiple cameras situated around an office.

4.5.2 Relevance and Evaluation

Active Badge is interesting on a number of fronts. First of all, it gives a solid example of context sensing. It also shows how context data can be processed and then interpreted to perform a useful function. Finally, an Active Badge type system could provide a context manager with a source of location context.

If such a level of customisation and adaptation can be brought about by a location-aware system alone, then it should be possible to create an even ‘smarter’ architecture by using context information from a number of different sources.

The Active Badge Project also makes it abundantly clear that some form of context filtering is required, since large amounts of data may be produced by the context sources.

Whilst Active Badge outlined the way in which raw location data is transformed into a useful representation, it does not represent data in a form that is transportable and useable by a large number of applications. This shortcoming may have been rectified in more recent versions.

Having said this, the Active Badge Project does give an excellent overview of the way in which raw data is transformed into something more meaningful. This kind of description was neglected by many of the other papers and projects reviewed here. A context manager will need to possess a similar data processing function. The use of a standard description format for this context information will make processing simpler. It should also allow complex relationships to be formed between attributes. This makes it possible to reason about context so that higher level context can be inferred from lower level (raw) context. Admittedly, the Active Badge system only maintains one kind of context information – location context – and it is therefore not necessary to describe relationships between attributes.

4.6 SNMP

4.6.1 Description

The Simple Network Management Protocol [19] is very rarely thought of as an example of a context aware application. However, in essence, SNMP is just that. The purpose of SNMP is to provide information about nodes on a network. The information provided can describe the node’s hardware, its operating system and software, and network statistics among other things.

SNMP defines a hierarchy of attributes that can be managed. This hierarchy is known as the Management Information Base (MIB). All SNMP compliant nodes run an SNMP agent that maintains a MIB particular to that device. The agents communicate with SNMP management stations. The management stations can query the agents running on the nodes in order to get up to date information about each node. In addition, agents can be configured to automatically notify management stations about some simple events. In SNMP parlance, these notifications are called traps.

Data is sent between agents and management stations using ASN.1 encoding to enable multi-vendor communication. If SNMP can be likened to CC/PP exchange protocol, then the MIB can be likened to CC/PP and ASN.1 can be likened to XML/RDF.

4.6.2 Relevance and Evaluation

SNMP demonstrates several the three aspects of context management: sensing, representation and interpretation. SNMP agents can be viewed as context sources. They make information available to management stations. As mentioned above, the data is modelled using a MIB and transported using ASN.1 notation. Although there is not a great deal of interpretation on the part of a management station, sense does have to be made of traps and information received. Network administrators perform the real interpretation when presented with several facts.

SNMP can be classified as a context management system because it maintains current information about nodes on a network. A system administrator uses this information to identify and (hopefully) prevent problems on the network. In some cases, software running on the management stations can take certain actions when specific conditions arise.

In much the same way that a mobile enterprise platform hopes to adapt applications to various changes in context, system administrators make alterations to the network or nodes on the network when SNMP notifies them of changes in the condition of the network.

SNMP is of interest not only for the purpose of drawing comparisons, but also for the purpose of being a source of context information. SNMP agents run on a wide variety of devices, including some mobile devices. SNMP could therefore be used to gather some kinds of context information for the context manager in m^3. For example, some MIBs contain information about the available network interfaces on a device. Information about the operating system running on the device is also available.

4.7 Formalizing Context

4.7.1 Description

No paper on context would be complete without a brief discussion of the formal roots of context theory. The AI community lends much to this area of study. McCarthy and Buvac [9] have described a way in which context can be formally described by logical formulae. While they do not provide a definition of context, they do provide a number of examples of the way in which context is applied to lift ambiguity. They also show how it is possible to infer truth in one context from truth in another context. This is known as lifting axioms.

McCarthy and Buvac introduce the relations:

ist(c,p)

and

value(c,e)

The first relation asserts that the proposition p is true in the context c. For example:

ist(context-of(‘Sherlock Holmes novels’), ‘Holmes is a detective’)

asserts that it is true in the context of Sherlock Holmes novels that Holmes is a detective. However, in the same context, the proposition ‘Holmes’ mother’s maiden name’ does not have a value. These examples are taken directly from the paper. The second relation is used to assign a value to a term depending on the context. Here’s an example, also taken from the paper.

value(context-of(‘Sherlock Holmes novels’), ‘Number of Holmes’ wives’) = 0

but

value(context-of(‘U.S. legal history’), ‘Number of Holmes’ wives’) = 1

These two relations are essentially a means for recording context information. They give the truthfulness and value of statements. One can write these relations using any syntax. Consider the following (pseudo) RDF statements which I claim are equivalent to the above value relation.

```
<?xml version=”1.0”?>
<rdf:RDF>
 <Description about=”USLegalHistory”>
  <NumberOfHolmesWives> 1 </NumberOfHolmesWives>
 </Description>
</rdf:RDF>
```

In many cases, a predicate in an *ist* relation or a term in a value relation can simply be treated as an attribute of the context. In this light, the RDF statements above make more sense.

These simple relations can be used in lifting axioms to infer truth in one context from truth in another context. This is the most interesting aspect of context theory in terms of context in a mobile environment. In some systems, document ownership relates only to a specific user. In others, ownership can relate to a user as well as a group. The following shows how the notion of groups can be integrated into an existing relation that describes the ownership of a document by a single user.

Imagine we have the general relation

$$policy: owner(user, document) \phi permission(user, document)$$

in a context *policy* which states that if the user *user* is the owner of document *document* then *user* has permission to access *document*. This can be translated into the context relations where *c0* is said to be an outer context (the big picture).

$$c0: \quad ist(policy, (Aud)(owner(u, d) \phi permission(u, d)))$$

Many operating systems also have the notion of group permissions. To model this using relations, it would be possible to simply augment the existing relations with a third argument, *group*. However, we need to express the same relationship using *ist*. The following shows how this is done. We introduce a new context called *group-policy*.

$$group\text{-}policy: \ (Augd)(owner(u, g, d) \uparrow ist(groups(g), owner(u, d)))$$

$$group\text{-}policy: \ (Augd)(permission(u, g, d) \uparrow ist(groups(g), permission(u, d)))$$

where *groups(g)* introduces a new kind of context pertaining to groups of users. Now all that remains to do is re-establish the relationship the *owner* and *permission* relations. This is achieved by *importing* the original *policy* context into the *group-policy* context.

$$c0: \quad (Ap)ist(policy, p) \phi ist(group\text{-}policy, (Ag)(ist(groups(g), p)))$$

which states that any fact from the *policy* context holds in the *group-policy* context. In short, it asserts the fact that just as user ownership of a document implies permission to read or write the document, group ownership also implies the right to read or write the document. This was done, using the same two argument relation *ist*.

4.7.2 Relevance and Evaluation

Formal theory of context describes useful methods for interpreting context. It not only shows how to gain meaning from simple contexts, such as the Sherlock Homes examples, but it provides a recipe for discovering truth in one context from truth in another context. Interpretation of context is a major goal of context management as stated in Section 3.3.1.

Although this project does not attempt to apply these context relations in the prototypes developed, it is suggested that this method of inferring facts in one context from facts in another context will be most useful. There are several reasons for this. The context data stored by a context manager cannot possibly be exhaustive. It is likely that certain facts will be missing from particular contexts. This may be due to disconnection, or it may be due to the fact that there are limited sources of context information in a particular situation. Whatever the reasons, it may still be possible to gain insight to the current situation using information from other contexts and then applying the lifting axioms. Lifting axioms are perhaps most useful in combining contexts. Given location context, it may be possible to combine this with policy context by applying lifting axioms to infer facts in the combined context. For example, if we have location context X and policy context Y, then it might be possible to infer Z (which could say something about the obligations a user has when he or she is at location X).

Since RDF is of the form 'subject-predicate-object', it should be fairly easy to convert RDF statements to *ist* relations, and then apply lifting axioms.

4.8 Conclusions

A good introduction to context awareness in mobile environments is given in [22]. In fact, that paper can be viewed as the basis for this project.

The paper introduces the concepts of abstract context and concrete context. The difference between these two types of context can perhaps best be explained with the use of an analogy. Abstract context can be likened to a relational database schema. A schema provides a data framework. It describes what kind of information may be stored, but it doesn't store the values themselves. Concrete context, on the other hand, can be likened to a populated database, where each field contains a value. The abstract context constrains the concrete context to the set of attributes that it defines. That is, the concrete context cannot have a value for a non-existent attribute in the same way that a database cannot have a value for a non-existent field. As will be seen, RDF, and thus CC/PP, by their very nature provide for these two context types.

A framework for context-awareness in distributed mobile systems is also outlined. It stresses the need for the separation of the various stages of context management

(context definition, context measurement, context interpretation, context management and context dissemination).

Because of the frequent possibility of disconnection in mobile environments, and the inherent latency of the network, the paper suggests the use of an asynchronous communication mechanism. By using asynchronous methods of communication, mobile devices can disseminate (push) their current context to the context manager whenever they have the opportunity to do so.

Another important issue is that context is described using a standard notation. This will allow platform independence. XML/RDF is such a notation. The CC/PP model should also be standardised shortly. Several other requirements outlined by the paper are supported by CC/PP. Attributes can be aggregated which allows the specification of a containment hierarchy. So, for example, the fact that rooms are contained within buildings can be described. The paper specifies the need to retrieve historical context data. Although CC/PP has the notion of a repository, the repository only acts as a store of current profiles. Therefore a context manager based upon CC/PP would need to introduce this facility.

The final requirement outlined by the paper is support for context filtering. Elvin and Elvin Event Correlations can be used to support this.

Each of the literature reviews above explains, describes or provides an example of one of the aspects of Context Management as outlined in [22]. The table below lists each review and shows which of the three main aspects of context management were covered.

	Sensing	**Representation**	**Interpretation**
CC/PP		RDF model and XML serialisation	
Odyssey	QoS resources only		
PRAYER	QoS resources only	Context Classes	
Savoir			Useful applications
Active Badge	Location only	Timestamped triples	
SNMP	SNMP Agents	MIB and ASN.1	
Context theory		*ist* and *value* relations	gaining meaning from context and lifting axioms

Table 2 Overview of literature survey

5 The Uses of Context in a Mobile Environment

In the following section, a number of scenarios demonstrate the way it is envisaged context will be used in an enterprise mobile architecture. In particular, it should highlight the relationship between the Context Manager and other management components such as the Adaptability Manager. This section also serves to present the various modes of mobility.

5.1 Adaptation scenarios

Several commercial products (IBM's WebSphere Transcoding Publisher for example) are available which can perform transcoding based on device specifications. This is only one of the functions a true platform for mobile environments must possess. The need for adaptation arises from much more than device capability. To see why, the various forms of mobility must be covered.

The word 'mobility' is not restricted to devices. Device mobility is simply the most obvious type of mobility and potentially the easiest to deal with. The mobility of users, software and other kinds of objects must all be considered. A user can be mobile without being attached to a mobile device. It's possible that a user may start a session at one workstation and complete the session at a different workstation. The workstations may have different specifications. The time elapsed between the start of the session and the completion of the session means that several environmental attributes of relevance to the application may have changed. Thus, adaptation needs to be performed for one reason or another. Of course, there are also a whole host of lower level network challenges that must be overcome during a session transfer, but these are beyond the scope of this thesis. However, a good source of information on session transfers is the work done at ORL on teleportation [26].

One can imagine that someday, items in a shopping centre will be tagged with transmitters and cheap CPUs. These objects can be mobile and in some situations are just as important as devices and users.

The following sections discuss aspects of user mobility, and the interactions required between an adaptation manager and a context manager to provide continued services to a user in the face of context changes. The differences between user mobility and device mobility are highlighted.

5.1.1 Mobile User With a Mobile Device

In this case, the user is mobile in the sense of using a single mobile device. Although the user and the device are mobile, the user remains bound to this single device. There are several issues to consider in this scenario. What happens when the device migrates from one network to another? The important case is when a big change in bandwidth

is associated with this network migration. As a concrete example, consider the case when a laptop computer migrates from a 9600bps GSM connection to a 100Mbps Fast Ethernet connection. This represents an enormous increase in bandwidth. The implications of this are that the laptop can now send and receive much larger amounts of data in a shorter amount of time. Instead of retrieving small black and white images, the application might now be able to retrieve and display larger full colour images.

Another issue to consider is the policy of the enterprise. Consider an organisation such as the United States Department of Defense (they call themselves America's largest business [14]). It is imaginable that a kind of wireless ad hoc network may be formed by devices for the purposes of collaboration amongst the staff of the DoD. The policy of the DoD might be that this kind of network may only be formed within the vicinity of the main Pentagon building as there is a high level of physical security and it is unlikely that the radio signals could be intercepted. Outside the grounds of the Pentagon, however, there is more chance that spies could intercept the radio signals.

5.1.2 Mobile User At Several Fixed Workstations

This scenario gives an example where users are mobile but devices are stationary. It highlights the need to think in terms of mobile users rather than in terms of mobile devices.

Usually, there will be a reason a user wishes to migrate from one machine to another. It might be that a particular machine has some specific capability that the current machine lacks. For example, a user might be having an audio conference with a colleague in a distant location. Before the conference started, it wasn't thought that video capability would be needed. However, as the meeting progresses, it becomes clear that the goals of the meeting cannot be accomplished without the aid of pictures. Both users migrate the session to a workstation that has video conferencing capability. The conference is continued with video.

In the DSTC, most members of staff are associated with one or more projects. Equipment is also managed according to project. Since resources are tight, project members guard their equipment closely. They don't want their workstation to be used by members of other projects[1]. So when a user attempts to migrate a game of 'Go' from one workstation to another, the mobile platform first checks the policy to see whether this user is allowed to migrate the session to this machine. If not, then the user is politely informed of this and presented with a list of workstations where the game of 'Go' can be continued. Whilst no adaptation takes place in this example (adaptation will take place when the user finds a machine where the game can continue), several other components of the mobile enterprise architecture do take part.

[1] This scenario is entirely fictitious and in no way reflects the state of affairs at the DSTC.

First of all, the context manager might need to provide the context of the situation (the user, the machine in question, the day of the week or any other information required) to the policy or security manager, so that they can make their decision (the user is not allowed to migrate to this machine).

5.1.3 Mobile User With Several Mobile Devices

This case is a composite of the above two cases. Both the devices and the users are mobile, and sessions can be migrated from one device to another. This case also covers the migration of a session from a stationary workstation to a mobile device and vice-versa.

In contrast to the previous case where the capabilities of the machines differ in only one or two aspects, the differences between mobile devices are much larger. Mobile devices range from laptops to mobile phones (one can even consider smart cards to be mobile devices). For example, the screen size of mobile devices can differ enormously. Consider the difference between a laptop screen and the screen on a smart phone. Whereas most stationary devices use a similar network (usually Ethernet or fast Ethernet), there are many different wireless network technologies available, each offering different bandwidths and each of which has a different cost-structure. There are relatively fast wireless networks such as 802.11 and relatively slow networks such as GSM. There are also more networks on the way (for example Bluetooth). Some mobile devices may be equipped with, for instance, GPS, and others may not be. Therefore the types of context available will also differ from device to device. This shows the vast differences between one mobile device and the next.

In the first case above, each user is bound to one mobile device, but the underlying platform still supports a number of users each with a different kind of mobile device. In this scenario, the platform must support multiple users who can swap between various mobile devices. An overview of some of the challenges faced are given here.

In the near future, the speed of the CPU within the PDA may cease to be a limiting factor on the capability of the PDA as a whole. That is, the CPU within a PDA will compare favourably with the CPU of a laptop computer. However, by its very nature, the screen size of a PDA will always be much smaller than that of a laptop (the whole idea of a PDA is to have a small computing device). This means that although the PDA is quite capable of processing large images, it is incapable of displaying them in a way that is pleasing to the user unless some sort of adaptation takes place. This raises the question: ‘What happens if a user migrates from a laptop to a PDA?’. It might be necessary to migrate a session from a laptop to a PDA if the laptop is running low on batteries, for example. Using information provided by the context manager (screen size, colour depth), the adaptation manager can perform some sort of filtering which shrinks the images to a size that fits nicely on the screen of the PDA.

There are a multitude of other situations where users might have to switch from one mobile device to another in mid-task. It is left to the reader's imagination and prior experience to find some other examples.

6 Requirements of a Context Manager

At this point the requirements of a context manager should be fairly obvious. They are clearly stated in [22].

1. A context manager must be able to present context information to consumers in a suitable form.

2. The context model must allow aggregation. In fact it should allow the expression of arbitrary relationships between attributes.

3. Dynamic updates should be possible. While static context goes some way towards achieving adaptability, dynamic context will allow for a much richer user experience, and it will allow better decisions to be made on the part of an adaptation manager and other components.

4. Context history should be available. As seen with the Savoir project, this is useful for making predictions about user behaviour. It can also help to infer more accurate context information when only limited context is available.

5. Context should be filtered. If it isn't then context consumers will be overwhelmed by the sheer volume of context data. The Context Manager takes the burden of filtering. That is one of its major tasks.

In short, the Context Manager is responsible for receiving raw context data, collating the data into a useful form and disseminating it to context consumers

6.1 The Long-Term Vision for Context Management

In its simplest form, context management can be thought of as storing application properties. For example, Netscape Communicator requires many user-supplied properties such as the user's name, the name of a mail server and the name of a proxy server. It is even possible for Netscape to automatically configure some of this information via scripts. Regardless of how this information is obtained, Netscape needs it in order to work correctly.

In a mobile environment, managing context becomes another issue altogether. Context information is maintained in order to provide timely information to an adaptation algorithm or manager so that content and the behaviour of applications can be altered to suit current conditions. This adaptation may be transparent or visible to

applications that run on top of this adaptation layer (the goal is to provide a completely transparent mobile architecture).

In a mobile environment, the attributes that constitute context information are always changing. For instance, a user might move from a high bandwidth network to a low bandwidth network; more memory might be slotted into the device; any number of attributes that make up the policy of an organisation could change.

In order to manage context information, there needs to be a way we can describe and store it, and there needs to be a way to update this information. This document presents a way in which to store and update context information, and justifies the choices made by referring to the significant constraints imposed by a mobile environment. A further aim of context management is to enable inference of contexts from other contexts. This feature is really what sets apart the context manager described here from other projects developed to this point. It is assumed that such inference will occur using methods similar to those outlined in the Buvac and McCarthy paper [9].

When describing concepts from computer science, it is often useful to find a real world analogy, especially if the concept must be described to the less technically inclined. The analogy here is to think of the Context Manager as an intelligence agency such as the CIA.

The task of an intelligence agency is to gather intelligence so that the government may make informed decisions about world affairs. In this sense, the government is analogous to an adaptation manager (actually the government can be thought of as a Policy Manager, and a Security Manager as well). The government takes the information gathered by the intelligence agency and applies it in some way. In the extreme, this information could be the trigger to start a war. In other less extreme cases, the intelligence provided could be used to determine a diplomat's approach when visiting a foreign nation. In the same way, the Adaptation Manager of a mobile architecture uses the information provided by the Context Manager to make decisions about how to modify content or change some aspect of an application's behaviour.

An intelligence agency can gather information from a number of different sources. Information can come from spies, military personnel, aircraft, and satellites. More recently, some information can be gathered from computer networks. The intelligence agency then collates this information to obtain an overall picture of a situation that can be provided to the government so it can take the required action. In a mobile architecture, such as m^3, there are many sources of information that must be gathered and presented in a usable form to the other components. Information can come from hardware, software, users, location managers, the organisation (the enterprise or institution), or even from the other components of the mobile architecture. For

example, some policy information could be used as context information. The Context Manager's task is to collate all this information and transform it into something that can be used by other components.

If the intelligence agency keeps a database of all its intelligence information, it should be possible to build an application that can answer questions such as 'What does "John Howard" mean in the context of Australia?' Hopefully, the application would give an answer along the lines of 'Prime Minister'. An advanced Context Manager should be able to provide a similar level of functionality. Instead of providing the Adaptability Manager only with information that pertains to the present situation, it should be able to provide information about an object or subject within a given context. The given context could be anything from a location to a time frame to a username. In fact, the given context is no different from the objects or subjects being asked about. Take the above example. It is also valid to ask 'What does "John Howard" mean in the context of Prime Minister?' This time the answer will be 'Australia'. Of course there maybe other valid answers depending on the context information stored by the Context Manager. In a sense, the Context Manager is like an expert system. Not only does the Context Manager provide notifications of context changes, but also it should be able to take a subject and a given context and provide an acceptable answer. This kind of system should be the ultimate goal of context management. Rather unfortunately, it is beyond the means of this project to build such a system, though it is reasonable to suggest such a system could be implemented in the near future. The reader is encouraged to compare the context manager proposed here with the MYCIN expert system [26].

7 .The Strengths and Weaknesses of CC/PP

As its name suggests, CC/PP was developed in order to describe device capabilities and user preferences. The intention of those who are designing CC/PP is that device capabilities and user preferences can be matched with application or document profiles so that an optimum representation can be returned to the user. However, as should be clear by this stage, context information extends beyond device capabilities and user profiles. Since CC/PP is likely to be adopted by device manufacturers, it is an obvious choice for describing context information in general, as long as it can be extended to include any context information that might be used by an adaptation or policy manager. It will be shown how CC/PP can be extended for this purpose. It is an advantage to be able to describe all context information using just one data model. It means that a single algorithm can collate all context information.

7.1 Suitability for Describing Context

This section evaluates the suitability of CC/PP to describe context information. It does this by referring to the requirements of a context manager (Section 6).

7.1.1 The Model

Recall from the Literature Survey (Section 4.1) that the CC/PP model is based on a tree structure whose root is the current network session. This section discusses the suitability of this model to describe general context information.

CC/PP is designed such that an origin server or proxy can perform some sort of device-document matching. In this sense, CC/PP is designed for use directly by an adaptation algorithm. The sequence of steps in the general case (device sends entire profile, document profiles are modelled in CC/PP or translated to CC/PP, disregard proxies and there are no errors) would look something like the following:

1. Device sends serialised profile model with request for content
2. Origin server receives serialised RDF and converts to an in memory model
3. The profile for the requested document is retrieved and an in memory model created
4. The device profile model is matched against the document profile model
5. A suitable representation of the document is chosen. At this stage the document to be returned can be chosen from a number of different versions of the same document or it can be dynamically generated
6. Document is returned to device and presented

This scenario is one presented by the W3C and is captured in Figure 3 below.

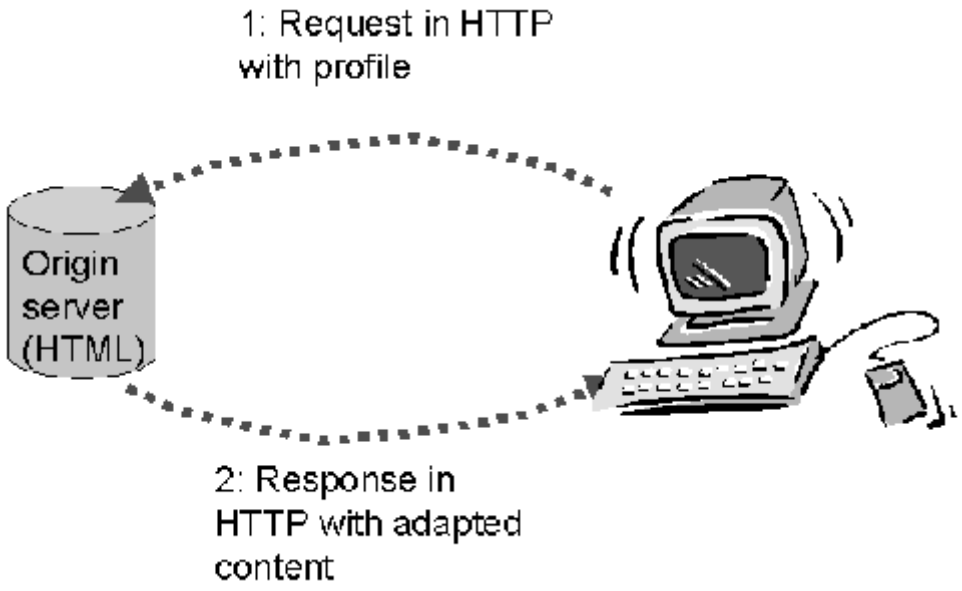

Figure 3 The simplest use case of CC/PP

The approach taken by a mobile enterprise architecture such as m^3 must be different. There can be innumerable context sources, and the adaptation manager would be overrun if it were required to perform all matching. Filtering context data is one of the five requirements of a context manager. CC/PP, as it stands, has no facility to support filtering. A proxy or origin server must process whatever information it is sent by a device, and whatever information it retrieves from a CC/PP repository.

The approach taken by a mobile architecture should be flexible enough to provide some adaptation mechanisms even where documents and applications can't just be matched with device capabilities. In order to provide this, a context manager should allow an adaptation manager to be as general or as specific about context as it desires. For example, if a document or application is specific about how it should be displayed, or if there are several versions of the document or application for different devices, then the Adaptation Manager can ask the Context Manager for detailed context information. This fine-grained approach allows a high level of adaptation to take place. In cases where the document does not provide profile information, or the profile is limited in description, the Adaptation Manager can obtain a general context class from the Context Manager, and perform some limited adaptation. For example, some adaptation can still take place where the location of the user is important. The Policy Manager can specify some rules about how adaptation should take place when a user is at a certain location, regardless of the information provided in an application or document profile. In a context manager, the CC/PP model will sometimes be used as envisaged by the W3C (that is, for describing device capabilities and supporting document profile matching), but is more likely to be used simply as a context description model.

Since CC/PP is to be used as a general context description model, the question needs to be asked:

'Does it still make sense for a network session to be the root of a CC/PP model?'

The implementation provided by this project retains the network session as the root of the model. However, it is suggested that in a context manager such as that proposed by m^3, objects will still be monitored by the Context Manager and its associated awareness modules even if that object is not participating in a session of some kind. For example, consider the situation where a context consumer would like to know the whereabouts of a particular person, but that person is not participating in any network sessions at the time (this might be the case when a context consumer is attempting to determine when another person is co-located with this person). In the CC/PP model as it exists, this information will not be determined since the person in question is not participating in a session and therefore will not have an associated CC/PP model at the server side.

A context manager should have the capability to provide context outside of the existence of network sessions. It should make as few assumptions as possible with regard to the kinds of context information required by context consumers.

Also, in the current model, if a user participates in more than one session at a time, then large amounts of data will be duplicated. The roots of the CC/PP trees will be different (unique session IDs), but much of the rest of the trees will be identical.

Therefore, it is proposed that in future, context models, described using a slightly modified CC/PP, should be associated with an *object ID* instead of a session ID. This makes sense for a number of reasons:

1. As described above, the context information associated with objects should be available outside the scope of a session. That is, an object exists whether or not it participates in a session

2. An object can participate in many sessions, but data will not be duplicated since the model is object centric instead of session centric

3. It is possible that some context information will not apply to sessions at all. There is some context information that provides descriptions of the environment in general, and does not make sense if applied to sessions. For example, how does one assign the information that 'room 619 is air-conditioned' to a session?

There may be other reasons for modifying the CC/PP model to become object centric. These are just the obvious reasons.

A final criticism is directed more towards the RDF model than to CC/PP itself. RDF is often cumbersome and complex. Statements made in RDF can often be modelled in other knowledge representation syntaxes (simple predicate/relational calculus comes to mind) in a more efficient and comprehensible way. But RDF's ability to be serialised in XML is a major positive factor and may aid in the widespread adoption of CC/PP.

7.1.2 The Vocabulary

As mentioned earlier, new vocabularies must be introduced if CC/PP is to be useful as a means for expressing any kind of context. It is presently designed to express device capabilities and user preferences. A mobile architecture should be flexible and extensible. This means that the architecture itself should be able to adapt to new situations, and it should support the addition of new context sources. At first, the architecture may deal with device capabilities and user preferences. As it evolves, new sources of context may be added. For example, a context source that provides information as to the whereabouts of a device and its user could be added to the system. A component that describes information about the objects or processes within an organisation would also be a rich source of context information. An example of context information provided by a component such as this would be the current set of projects undertaken by the organisation, and the members of each project. The Policy Manager might act as such a component. Thankfully, because it is based on RDF, new vocabularies can be extended to express these extra kinds of context.

CC/PP in its present form is only useful for describing the current situation. That is, it will store information such as 'the current bandwidth is 9600bps' but does not store information such as 'the bandwidth available in GP South is 10Mbps'. It's true that CC/PP is not intended to provide this kind of information, but once again, it is precisely this kind of information that will make the mobile platform useful, and it relates to the discussion of the CC/PP model above. Imagine the scenario where a user is prevented from downloading a multimedia object because it would take too long over the current low bandwidth connection. Using knowledge of the user's current location, its proximity to GP South and the bandwidth available in GP South, an action taken by the adaptation manager might be to suggest that the user walk to GP South and use the facilities there.

This information is describable in CC/PP, but it is certainly not what its originators intended. In order for this information to be described, one needs to introduce new vocabularies relating to projects. This includes but is not limited to a Project type, Personnel such as scientists and project leaders, and new constraints that can be applied to the new attributes.

The RDF model allows new vocabularies and the semantics associated with those vocabularies to be created using a layering approach. First of all, a base schema can be created. Using the example above, one would create a schema defining new types such as a Project type. The schema will also define some attributes (properties) such as scientists and engineers. Other schemas can be layered on top of this to provide new types and attributes. These attributes can inherit from attributes defined in the base schema or other schemata. More schemata may be layered on top of this if need be. Finally, a document describing instances of classes is created. The document also assigns values to the properties of the class. As an example, we create ‘Project X’ which is of type ‘Project’. Then we may assign values to the properties of Project X. The ‘scientists’ property may take a number of values, so we create a ‘Bag’ and add the values ‘Jill’, ‘John’ and ‘Susan’. Jill is also the leader of the project. A model of the resulting RDF graph is shown below.

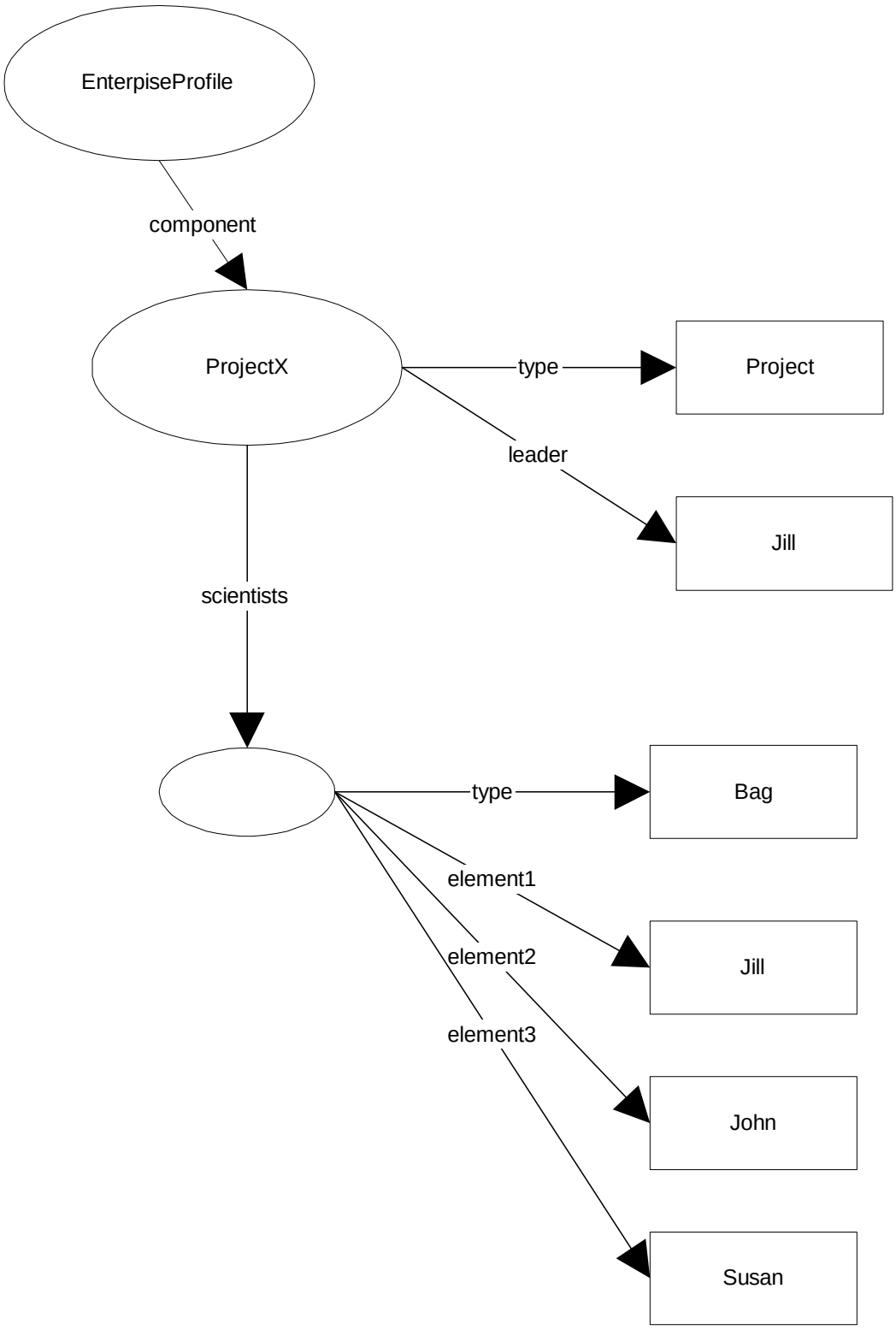

Figure 4 An RDF model of Project X

One important factor in choosing CC/PP is that it is a proposed W3C standard. This means that it is likely to be supported by a wide range of device manufacturers, each of whom may introduce their own vocabularies.

7.1.3 The Serialisation

Another consideration is that RDF can be serialised using XML. The platform independent nature of XML means that CC/PP can be transferred to a large number of systems. XML is becoming the standard way to encode information for transmission across a network. The number of RDF tools available for processing RDF documents is growing. The Java based RDF API was used extensively in this project [16].

Context sources provide raw context data. Sometimes this data is useful in its present form. At other times, users of context information such as an adaptation manager will require this information to be manipulated in some way by the Context Manager. The following example is based on context data as will be sent by the Location Manager described in [23].

```
<? xml version="1.0" ?>
<rdf:RDF
xmlns:loc="http://www.m3.org/location-schema#">
<rdf:Description about="BobsPhysicalLocation">
<rdf:type rdf:resource="http://www.m3.com/location-
schema#PhysicalLocation"/>
<loc:country>Australia</loc:country>
<loc:state>Queensland</loc:state>
<loc:city>St Lucia</loc:city>
</rdf:Description>
</rdf:RDF>
```

Listing 3 Location context encoded using XML/RDF

Using this information as sent by the Location Manager, the Context Manager can manipulate this data into a form that stores this information in terms of the location instead of in terms of a session or user. Components may prefer the information to be stored in this way. Here is an example of how the data can be manipulated.

```
<? xml version="1.0" ?>
<rdf:RDF
xmlns:user="http://www.m3.org/user-schema#"
xmlns:loc="http://www.m3.org/location-schema#">
<rdf:Description about="BrisbaneHQ">
<rdf:type rdf:resource="http://www.m3.com/location-
schema#PhysicalLocation"/>
<loc:country>Australia</loc:country>
<loc:state>Queensland</loc:state>
<loc:city>St Lucia</loc:city>
<user:personnel>
 <rdfs:Bag>
  <rdf:li>Bob</rdf:li>
  <rdf:li>Jane</rdf:li>
  <rdf:li>Greg</rdf:li>
 </rdfs:Bag>
</user:personnel>
</rdf:Description>
</rdf:RDF>
```

Listing 4 Location context shown in terms of a location

Now the data is stored terms of the location instead of the user. This view makes it easy to see who is at a particular location, in this case, the Brisbane headquarters of some company. The question remains, is the Context Manager responsible for this kind of data manipulation, or are the consumers of context data responsible for transforming data into a view suitable for their needs?

The Context Manager has no way of knowing what form consumers would like the data to be in. For this reason, it is suggested that consumers of context that prefer information to be in particular forms also act as context sources. In the above example, consumers would receive context data in a form similar to the first RDF listing. If the consumer knows that it will frequently require information about who is at the Brisbane headquarters, then it can create a new context in the form of the second RDF listing. This will save on processing time since the Context Manager will not have to process requests that require searching through all user contexts looking for people who are at the Brisbane headquarters. Instead, the Context Manager can simply return the single location component that records the personnel currently at that location.

7.1.4 Conclusions

Given this level of complexity, the task of collating context information should be allocated to a specialised manager: the Context Manager. By doing this, the component that performs adaptation can be provided with different views of context depending upon the situation. In this way, the Adaptation Manager is not restricted to rendering content based on device and document profiles.

This has the following ramification: in the mobile architecture proposed by the DSTC, CC/PP is not always used in the way envisaged by it creators. Adaptation will not always take place through matching CC/PP profiles with document profiles. The Adaptation Manager may make use of CC/PP, but is more likely to request a more abstract view of context such as a context class. In short CC/PP provides the basis for expressing context information and can be manipulated by the Context Manager into a form more useful to an Adaptation Manager.

7.2 Suitability for a Mobile Environment

The CC/PP working group has considered several issues that are important to mobile devices.

7.2.1 Limited Bandwidth

First of all, the bandwidth available to a mobile device is usually limited to that compared with fixed workstations. For this reason it is important to think about ways to lower the bandwidth consumed by CC/PP. There are two main ways that bandwidth

can be preserved. CC/PP allows device characteristics and user preferences to be described with an indirect reference. That is, a profile may simply one or more URIs that point to a location where the actual attributes and values are stored. In this way, a mobile device saves bandwidth by sending a reference instead of a complete profile. Note this is distinct from the references sent using the CC/PP exchange protocol.

Second, profiles may be cached by proxies and origin servers, and stored in a CC/PP repository. This means that mobile devices need only send profile differences instead of sending the entire profile on each request for content. If some aspect of the user preferences or the device capability changes, then only that change need be sent. The origin server or proxy merges the change with the cached profile to create a full, up-to-date description of the device and user preferences.

7.2.2 Disconnectedness

Disconnectedness is also a characteristic of mobile environments. While it is beyond the scope of CC/PP to address this issue (since CC/PP is intended for request-response protocols such as HTTP, if the device is disconnected, it cannot make a request in the first place), a context manager is concerned with disconnection. The context surrounding disconnection can be used in the future. For example, if it can be determined that disconnection occurred due to a gap in wireless coverage, then the user can be warned of this next time he or she approaches the gap. Context may be used to re-establish the session upon reconnection. Note that this does not entail creating a *new* session, but actually using the *same* session as before the disconnection occurred. This is similar to session migration, and has but occurs for a different reason. Context consumers may also need to know the context of a disconnected entity. Disconnection should not prohibit the dissemination of context to context consumers.

7.3 CC/PP exchange protocol

Although the CC/PP exchange protocol is separate from CC/PP, it is discussed here as an example protocol for the exchange of CC/PP.

The CC/PP exchange protocol (CC/PPex) provides one method of sending profile information to proxies and origin servers [3]. This protocol is based on the HTTP extension framework (HTTPext) [4], which means that it can interact with existing web servers and proxies. There are several different methods by which CC/PP information can be transferred.

First of all, the HTTP extension framework describes two declaration strengths: mandatory and optional. Mandatory should be used when the user agent requires an error response when the origin server or proxy does not comply with the extension

protocol (in this case CC/PPex). Optional should be used when the user agent wants to receive content whether it has been tailored or not.

Second, HTTPext offers two extension declaration scopes: hop-by-hop or end-to-end. Which scope is used depends on the first-hop proxy. If the user agent knows that the first-hop proxy supports CC/PPex, then hop-by-hop should be used. If, on the other hand, the user agent does not use a proxy or knows that the first-hop proxy doesn't support CC/PPex, it should use end-to-end scope.

This means that there are four different combinations of declaration strength and scope.

For examples of each of the four combinations, refer to the W3C document 'CC/PP exchange protocol based on HTTP Extension Framework' [3]. This document also describes the Profile-warning header and lists the warn codes that should be returned by servers and proxies when content is returned (or if no content is returned).

The CC/PP exchange protocol provides its own method for indirection. The header of a request may contain one or more URIs that point to CC/PP profiles. A request may contain either an entire profile or just the differences since the last request. The above two features were included for the purpose of saving bandwidth.

A brief example of how the CC/PP exchange protocol operates is included here. This example is a combination of mandatory strength and end-to-end scope.

First of all, the user agent issues a mandatory request, using end-to-end scope. This is achieved by using the M-GET method and by including a 'Man' field in the header. This request includes a profile difference as well as two indirect references. The profile difference is identified by the profile-diff-digest which aids in the profile lookup process within servers and proxies. A profile-diff-digest is created by running the MD5 digest algorithm and then the Base64 algorithm over the XML that constitutes the profile difference itself. A profile-diff-digest appears on line 4 of the following listing.

```
M-GET /a-resource HTTP/1.1
Host: www.w3.org
Man: "http://www.w3.org/1999/06/24-CCPPexchange"; ns=99
99-Profile: "http://www.aaa.com/hw","http://www.bbb.com/sw","1-uKhJE/AEeeMzFSejsYshHg=="
99-Profile-Diff-1: <?xml version="1.0"?>
                <RDF xmlns=http://www.w3.org/TR/1999/PR-rdf-syntax-19990105#
                    xmlns:PRF="http://www.w3.org/TR/WD-profile-vocabulary#">
                 <Description ID="SoftwarePlatform" PRF:Sound="On" />
                </RDF>
```

Listing 5 An example request using CC/PP exchange protocol

The origin server can then either reply with an error if it doesn't support CC/PPex, or it can fetch the profiles pointed at by the indirect references from the CC/PP

repositories. Once it has retrieved the profiles from the repositories, the origin server generates the tailored content and returns it to the user.

The CC/PP exchange protocol is the only proposed protocol for exchanging profiles at this point in time. A protocol based on Elvin is proposed later in this document. This protocol is especially suited to context updating in situations where a large proportion of content is pushed to the client. It is also useful for context sources (awareness modules) that are separate from the mobile device.

8 Contributions Made by this Project

In order to be useful in an enterprise mobile environment, CC/PP must be able to do much more than just express the capabilities of a device and the preferences of a user. By augmenting the CC/PP vocabulary to enable other kinds of context to be stored, a single data model can be used to describe all contexts within the architecture. That is, the operation of the context manager can be simplified a little because it only needs to speak CC/PP and it doesn't need to translate between CC/PP and other models for context storage. This section outlines the major components of work completed during the course of this project.

8.1 Extending the vocabulary

In an enterprise mobile environment, it would be useful to store information related to the objects of an organisation. For example, by storing context information about the projects being undertaken by an organisation, a context manager could combine this knowledge with location information or some other context so that the architecture could provide fine tuned access control to documents. Using cross-location information, the architecture could automatically send content that should be discussed while two or more relevant people share the same location.

This is just one small example of the kind of information that a context manager is required to store, but that CC/PP was not intended for. Figure 4 in Section 7.1.2 shows an example profile that describes information about a project. This information is in no way related to device capabilities or user preferences, but it does represent an interesting context that a context manager might be required to store. It is also true that the information stored within a profile about projects is not associated with a session in the way that device capabilities and user preferences are. As discussed in Section 7.1.1, it might make more sense to use object IDs rather than session IDs for this very reason. For now, the entity that creates and updates the project information can be thought to be associated with a session, in the same way that a device and user profile is associated with a session.

Several schemata and profiles are given in Appendix C: Examples of extended CC/PP vocabulary. These examples continue with the theme of project contexts. To make sure the examples were legitimate RDF, they have been parsed with the RDF API and converted into a CC/PP model using the prototype context manager created by the m[3] team. These examples are used in the prototype demonstration.

The task of extending CC/PP with a new vocabulary was relatively trivial as compared with the task of creating new constraints.

8.2 Introducing new RDF constraints on attributes

Also of concern are the constraints and relationships between attributes that make up mobile context. RDF specifies some simple constraints such as the domain and range of an attribute. There are other kinds of constraints that are useful in describing context information that CC/PP does not provide, but may be described using RDF. It may sometimes be necessary to constrain the software installed on a device. This may be due to conflicts caused by installing two particular applications together, or it may be some other reason. Imagine that installing Microsoft .net and the Java platform on the same device causes conflicts. In this case, there needs to be a constraint placed between the .net attribute and Java. This constraint can be called XOR[2]. If an XOR relationship exists between two attributes, then a profile may not contain both of these attributes at the same time. Note that it would be up to the operating system or installation software to impose this constraint. CC/PP can describe the constraint and a modified RDF validator can validate the constraint. That is, on parsing a profile, the validator would detect if the profile has violated the constraint.

In a similar way an AND relationship between two attributes would mean that one attribute cannot exist without the other. The validator would detect violations of this constraint as well.

Cardinality constraints allow control over the number of elements in a collection. Some attributes may take on more than one value, or the value of an attribute may be a collection of elements. For example, an organisation can place cardinality constraints on the number of researches working on a particular project. The constraint might be tight, requiring an exact number of researches, or it might be loose, requiring the number to be in a certain range.

These constraints were described in RDF using a layering approach. In an RDF schema, the constraints themselves are defined. Because these constraints apply to attributes, they inherit from the RDF *propertyConstraint* type. As will be shown, this is not the most natural or efficient way to describe these constraints, but it does

[2] This XOR is not an XOR in the traditional sense. An equivalent logical statement would be NOT (X AND Y). That is, X may or may not exist, Y may or may not exist, but they may not exist together.

conform to RDF conventions. Another RDF schema is used to define components and attributes. It is here that the constraints are placed on attributes. Finally, a profile contains the actual values for the attributes. These profiles should conform to the constraints placed on attributes. The validator will detect if these constraints are violated.

A more natural way of specifying constraints such as XOR and AND is to define these relations as properties of a new class that called 'Relation' (for example) and to form a predicate tree. A brief example is shown here. Using the .net and Java example above, a logically correct constraint could be applied as follows.

```
<not>
 <and>
  <rdfs:Seq>
   <rdf:li>.net</rdf:li>
   <rdf:li>Java</rdf:li>
  </rdfs:Seq>
 </and>
</not>
```

Listing 6 Alternative constraint representation

In this way, it is possible to use the unary connective *not*. Unary connectives cannot be used in the original version. It is also possible to build arbitrarily complex relations using this method. However, specifying constraints in this way means that the *propertyConstraint* property of attributes cannot be utilised. From this point of view, the original version remains true to RDF conventions.

One point that was almost overlooked is the fact that properties can inherit from other properties in much the same way that classes can extend other classes. A ramification of this is that a validator needs to gather all m^3 constraints from parent properties. For example, there might exist a property *leader* and a child property *projectLeader*. Any m^3 constraints that apply to *leader* also apply to *projectLeader*. Therefore when the validator comes across a *projectLeader* attribute, it must not only check the m^3 constraints on *projectLeader* but also on the parent property *leader*. If *leader* is a subproperty of some other property, then the validator would recursively check its parent property too.

8.3 An Asynchronous Exchange Protocol

Elvin can be used by the sources of context information (the 'awareness modules') to notify the context manager of changes to context. Elvin is also used for the communication between the context manager and those components that wish to be notified of certain changes.

The proposed protocol is most effective where a device is not the source of the context information. For example, a location manager will probably not run on each device. Instead it is a separate entity which could monitor the location of a large

number of users and provide updates to the Context Manager in an asynchronous fashion.

However, this is not to say an Elvin based exchange protocol is not useful on devices themselves. The CC/PP fraternity has concentrated on web based content. In this environment, content is delivered in a synchronous fashion. A user agent makes a request using CC/PPex and an up to date device profile and it receives tailored content in reply. In the future, content may be delivered in all sorts of different ways. It is possible that some devices will not have access to the web, and will therefore not be able to use CC/PPex. In situations where content is delivered asynchronously (pushed), via a publish-subscribe protocol like Elvin, context updates need to be sent immediately to the Context Manager. The reason for this is that requests for data are only made once, in the form of a subscription. From then on, whenever new content matches the subscription it is automatically delivered to the client device. It is possible, even likely, that changes will occur in device or user context. Some of these changes will affect the way content should be adapted. The only way the Context Manager can get these updates is if the client sends a notification as soon as the change occurs.

If the client device itself has some intelligence, it can send updates only when a critical change occurs. That is, if the client knows which changes will require content to be modified in a different way, it can delay sending context updates until one of these changes occurs. This is unlikely to be the case, however, since it would require client devices to possess the same knowledge of adaptation as a fully-fledged adaptation manager.

Overall, the protocol described here bears little resemblance to CC/PPex. It does not consider proxies, (though there is work being done on an Elvin proxy at the DSTC [28]). In short, the Elvin exchange protocol was developed with a context manager in mind, rather than as a general exchange protocol for CC/PP.

For the purpose explanation, in the sections below, the Elvin server will be treated as a part of the Context Manager. Therefore the Elvin server does not appear in the diagrams.

8.3.1 Awareness Modules to Context Manager

There can be potentially hundreds of sources of context information. The can be referred to as ‘awareness modules’. The task of these awareness modules is to detect changes to any context attributes that might be relevant to the other components of a mobile architecture. One such awareness module could be a Location Manager that provides the Context Manager with information about the location of a user or other objects. Awareness modules can provide information not only about users and mobile devices, but also about other ‘objects’ within an organisation. For example, by

knowing the collection of projects underway within a research centre and the scientists involved with each project, the adaptability manager, in conjunction with the policy manager, could conceivably utilise this context information to restrict the dissemination of project documents to situations where only those people involved with the project are present.

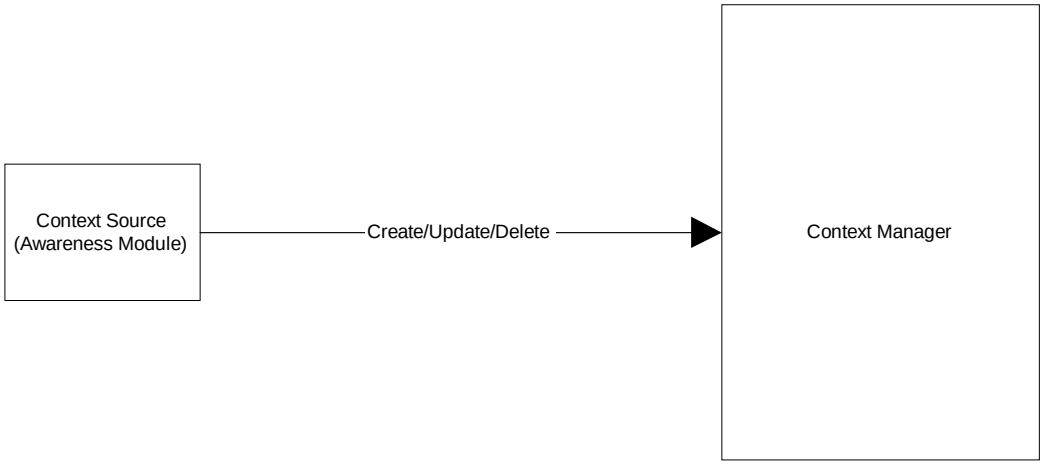

Figure 5 Source to Context Manager

8.3.2 Context Manager to Context Consumers

Here, a 'context consumer' refers to those components that register with the Context Manager to receive notification about context changes. The Adaptability Manager would be one of these. The Policy Manager would also be interested in some changes in context.

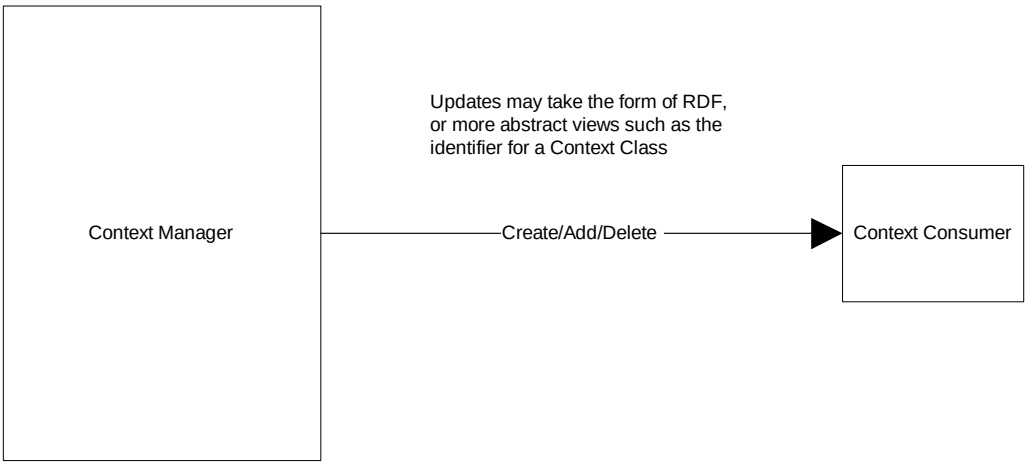

Figure 6 Context Manager ot Consumer

It is important to note that context consumers can also be sources of context information. For example, the Policy Manager has context information associated with it.

8.3.3 Elvin Event Correlations

Event correlation is explained in [18]. That paper also gives an event correlation model. As outlined in the paper, an event correlation could serve one of many purposes. Correlations can be used to compress, filter, suppress, generalise and specialise events and sequences of events. For example, compression is used when a sequence of two or more events can be adequately described with one event. The correlation engine listens for the component events which constitute the correlation, and emits a single event after all the component events have been received. The Elvin Event Correlation Engine is capable of performing these functions. The following section explains how and why Elvin Event Correlations can be used in a context manager.

It is sometimes the case that context consumers only need to be notified when a combination of several context changes occurs. For example, in some situations, an Adaptation Manager could decide that if bandwidth drops by an order of magnitude and the user removes some memory from the device, then it should be notified. If either of these events occur in isolation, it will not affect the way content is adapted.

Another use of correlations is to provide the implementation of context classes. A context class is a group of attributes along with a range of values for each of those attributes. The context changes if any of the values falls outside of the given range. This is similar to the notion of QoS classes in the PRAYER system [6], as well as many other network technologies. Figure 7 shows a conceptual view of context classes. The classes here consist of QoS type context information. There are two separate classes defined in the diagram. The first class encapsulates poor bandwidth, high probability of disconnection and low memory, slow CPU and poor graphics resolution. The second class defined represents medium bandwidth, high probability of disconnection and medium to high memory, CPU and graphics capability. This conceptual model can be extended to have as many dimensions as needed. Note though, that it is only possible to model context types that are quantifiable.

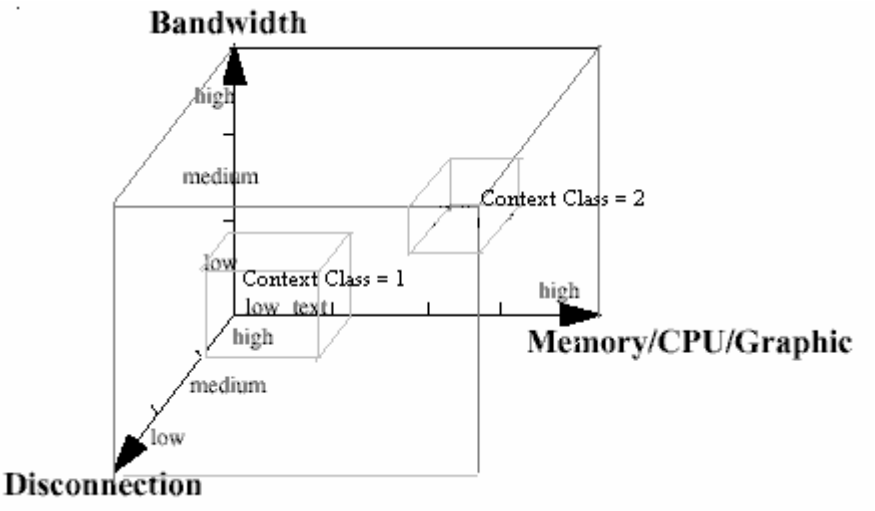

Figure 7 Conceptual view of Context Classes

Context classes can be described dynamically by the client (Adaptation or Policy Manager), or the Context Manager can predefine several context classes so that the clients ask to be notified when the context class changes. In the first scheme, the client will receive a list, or tree, of attributes and their values. In the second scheme, the client will only receive the name or identifier of the new context class. In this way, the client can control the level of detail with which it wishes to be notified.

Predefined context classes are managed by the Context Manager. In order to provide predefined context classes, the Context Manager behaves like a client by subscribing with correlation expressions that define each class. When the Context Manager receives a notification in response to these subscriptions, it in turn emits a notification that alerts clients who have asked to be notified using predefined context classes. Figure 8 shows how the Context Manager interacts with the correlation engine. Again, the Elvin server is taken to be part of the Context Manager.

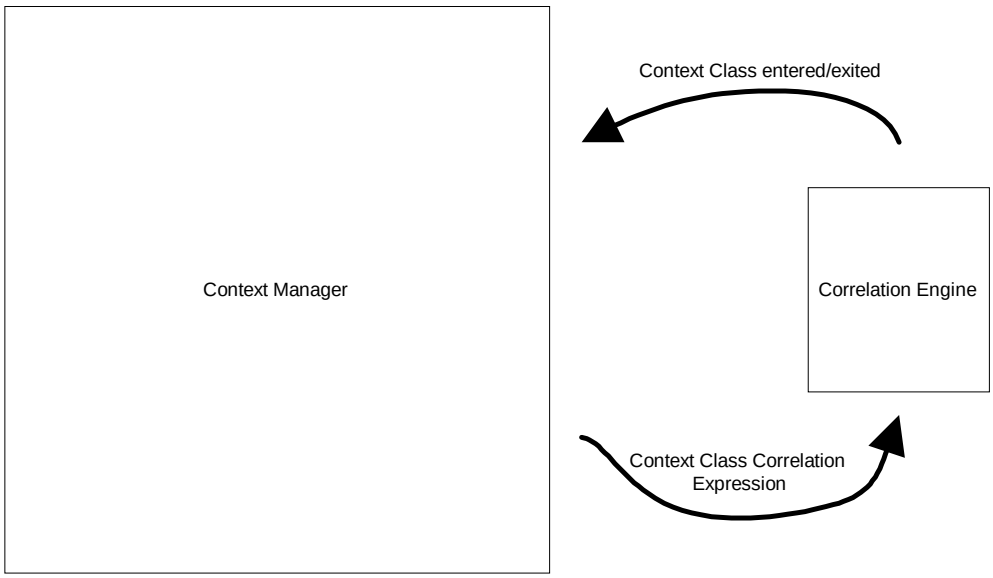

Figure 8 The Context Manager using correlations to implement context classes

The reason clients are given two different methods is for the sake flexibility. Although context classes provide a level of abstraction, in some circumstances they are much too abstract. Potentially useful information is hidden from the client. It is also the case that some attributes are hard to categorise into classes. For example, how does one integrate location information into predefined context classes?

9 Design of a Context Manager

In this section, the design of a Context Manager is presented. The design given here is based on the design of the m^3 Context Manager [2]. As described earlier, the purpose of the Context Manager is to gather and collate information from several context sources and make this information available to other components of the platform. Although the context sources, or awareness modules, are part of the Context Manager itself, the Context Manager can best be described by separating the awareness modules from the part that performs the collation and storage of context information. In the diagram below, the awareness modules are segregated from the Context Manager.

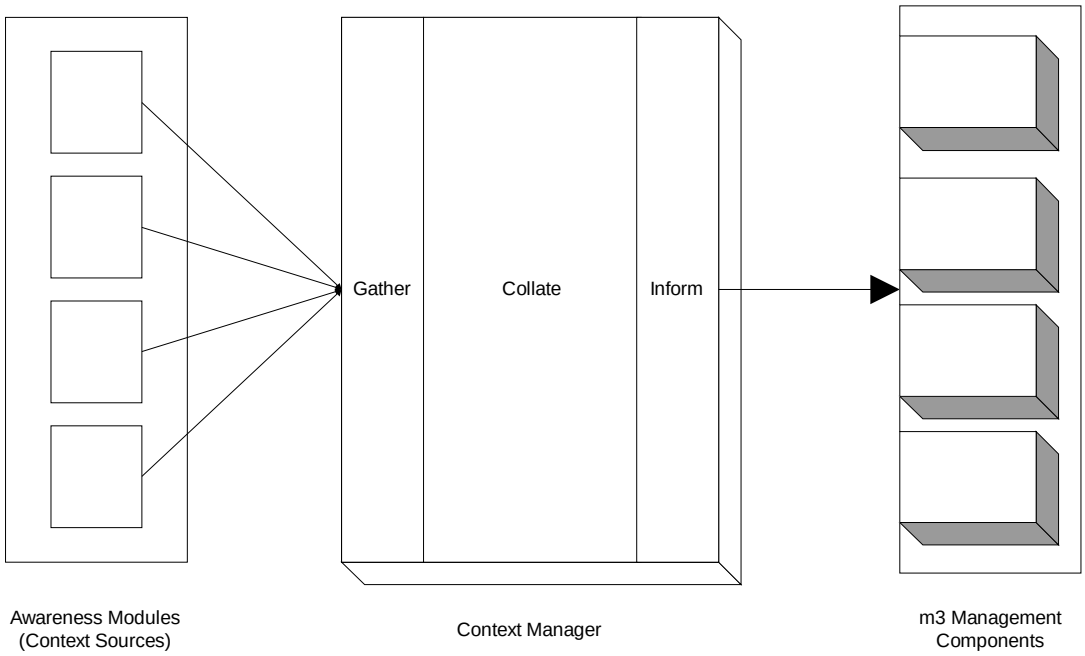

Figure 9 Top level view of a Context Manager

This diagram shows that there are three main tasks to be performed by a context manager.

Firstly, it needs to gather raw context information from the various context sources. Examples of context sources include a location module, an enterprise information systems and mobile device which can transmit device capabilities and user preferences. Context information is passed from the context sources to the Context Manager in the form of CC/PP serialised using RDF/XML

Next the data is stored in such a way that higher level facts can be inferred from the individual pieces of raw context data. Collation also encompasses the storage of historical context data that may be of use in the future.

The last task of the Context Manager is to make all this context information available to those components that are interested. Components should be able to request only a subset of the context information that represents those elements of context they are interested in. Another feature of the Context Manager is it should allow components to request varying levels of context granularity. Some components may require very detailed context information. In this instance, components can ask to be notified of changes to particular context attributes such as bandwidth, memory and location. This method provides the finest grained view of context. Other components may only be interested if more than one attribute changes. In this case, they can define their own context classes, and ask to be notified when the context class changes. Finally, the context manager specifies some pre-defined context classes which are identified by globally known names and whose properties are known by the components.

Components who ask to be notified of changes on this level receive notification whenever context changes from one pre-defined class to another. This method provides a coarse grained view of context.

The Context Manager also provides a method for synchronously accessing the context within a given scope. That is, components may acquire a view of the context as it exists at that instant.

9.1 Interface of the Context Manager

Accordingly there are several operations that may be invoked on the Context Manager. They are briefly outlined here. The methods can be roughly categorised according to the granularity of context as described above. In practice, these methods reside in a stub on client side, where the client is taken to be one of the components that require context information from the Context Manager. The interface defined here is different to that defined by m^3. For clarity and purposes of demonstration, the various kinds of context notifications are separated into several operations.

9.1.1 Fine granularity

There are several operations which fit into this category. Ordinarily, these operations would be used in combination.

9.1.1.1 registerForContextCreation( )

Register with the Context Manager to be notified when a new context is created.

9.1.1.2 registerForContextDeletion(String id)

Register with the Context Manager to be notified when the context with ID *id* is deleted.

9.1.1.3 registerForContextScope(String id, ContextScope scope)

Register for any context changes that occur to the context with ID *id* within the scope *scope*. A scope limits the context information that the user is interested in.

9.1.2 Medium granularity

This category contains only one method of registration, but allows components to define their own *context classes.* A context class is a set of context attributes and a range of acceptable values for each of those attributes. If one or more actual attribute values fall outside the given range, then the context is said to exit from one context to another. It is not always necessary to define all possible contexts. It is often sufficient for a component to be notified when the context exits from a class.

9.1.2.1 registerForContextCorrelation(String id, ContextClass cc)

Register with the Context Manager to be notified when the context with *id* enters or exits from the context class *cc*. A context class encapsulates an Elvin Correlation expression. This expression is sent to the correlation engine.

9.1.3 Coarse granularity

This category also contains only one method. It is similar to the previous operation except that the Context Manager predefines the context classes. Components that register in this way are simply notified when the context changes from one predefined class to another. The classes are given globally recognisable identifiers.

9.1.3.1 registerForContextClass(String id)

Register with the Context Manager to be notified when the context with *id* changes from one class to another.

9.1.4 Synchronous operations

There are several synchronous operations available too.

9.1.4.1 getContext(String id, ContextScope scope)

Retrieve the context for *id* within *scope*

9.1.4.2 getContextDescription(String id)

Get the schema for context with *id*. A schema defines the structure of a context.

For examples of use of these methods, refer to Appendix D: Context Manager Demonstrations.

10 Implementation

An initial Context Manager prototype was implemented halfway through the project. This entailed the development of a simple CC/PP model since no implementation existed at that point. This model conformed to the original CC/PP Note [24].

Since then, the m^3 team has developed a CC/PP model that more closely conforms to the recent CC/PP specifications [10, 11, 12]. A simple context manager was also developed. A slightly enhanced version of this context manager provides the basis for the prototypes implemented in the later stages of this project.

10.1 Original implementation

The original implementation made use of a Java based RDF parser (SiRPAC) which is freely available on the web. SiRPAC was used to parse serialised RDF into triples. Triples are the output of RDF parsers and are of the form triple(subject, predicate, object). Using these triples, it was possible to build a CC/PP model. The model was updated as new context data was sent from context sources.

This initial implementation was used primarily for demonstration purposes.

10.2 Current implementation

The m^3 team built the current implementation of the CC/PP model and simple Context Manager. This model is still in its early stages, but allows for a far richer vocabulary than the original implementation. It uses the RDF API [16] for processing RDF. The Context Manager was slightly enhanced for use within the prototypes developed by the author.

First of all, it was modified to allow the reception of context updates in the form of XML. It previously could receive only indirect URLs to CC/PP profiles.

Secondly a method was added to allow the resending of context information. This was required so that information already within the context manager could be disseminated to a context consumer when the consumer becomes interested. Previously, only new context data (context updates) would be sent by the Context Manager.

Third and foremost, a generic protocol adaptor was provided for use by context sources. The intention here is that the underlying protocol (Elvin, RPC, HTTP) is hidden from the context source. The source simply invokes the relevant method on the adaptor (create, add, delete) and the adaptor can make the decision as to which protocol to use. A concrete implementation using Elvin was also developed. The Elvin implementation marshals the serialised RDF into a notification and sends it to the peer at the Context Manager itself. Here the RDF is unmarshalled and passed to the CC/PP processor where it becomes integrated with the existing context model if one exists. The picture below shows one of the demonstration applications. This application emulates both a context source and a context consumer. Profile differences can be sent by typing RDF into the text area at the bottom of the screen. The tree on the left-hand side reflects a CC/PP tree. The main panel shows the value associated with the attribute selected in the tree. The application is called *Granular Context* because it registers to receive detailed context information.

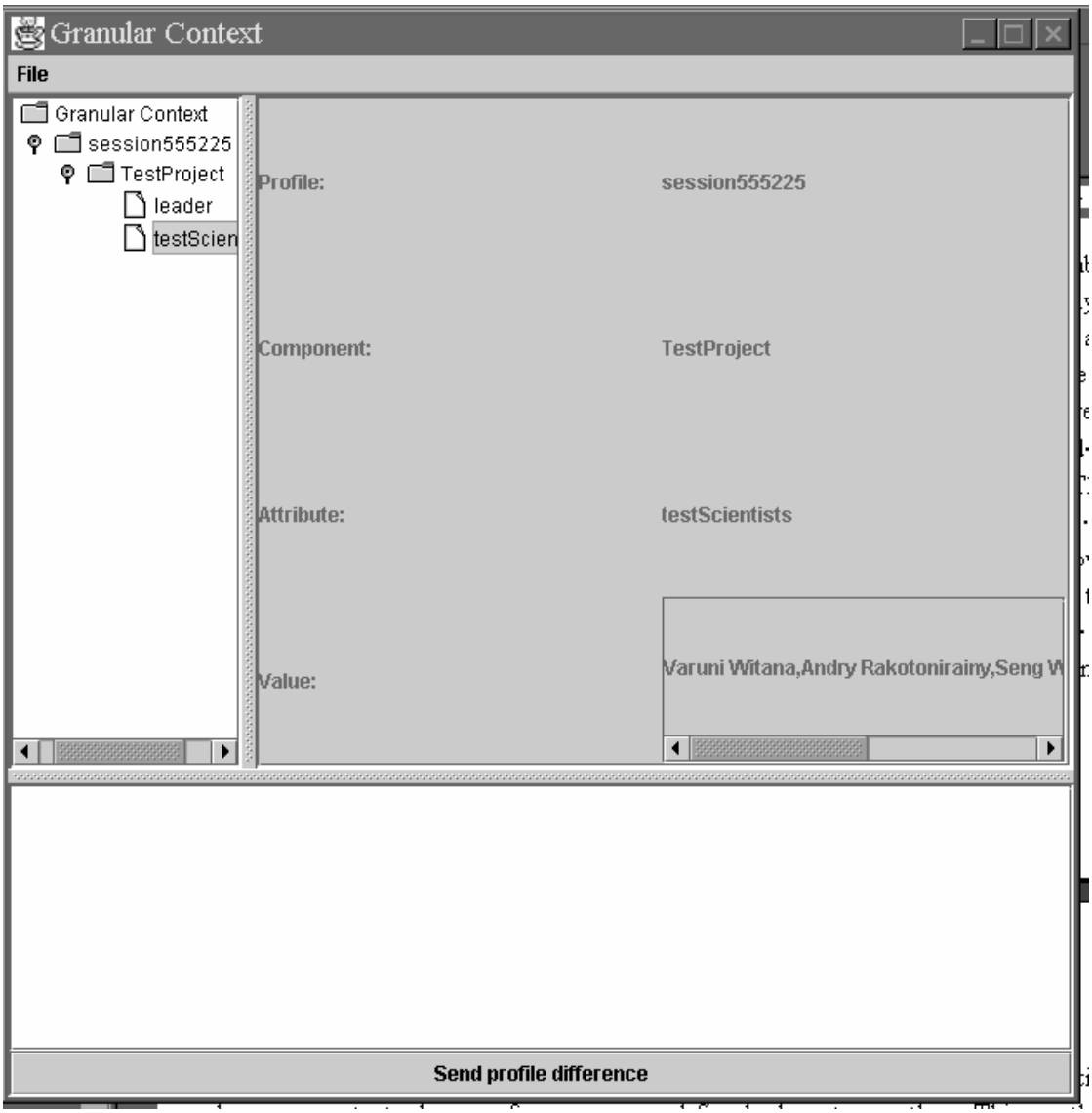

Figure 10 Screen shot of a GUI that emulates a context source and a context consumer

Context consumers use a similar stub mechanism. Again, this is to provide abstraction and makes changing the underlying protocol easy if this needs to occur. Once again, a concrete implementation using Elvin was developed. As shown in the design section, there are several ways in which a context consumer can register with the Context Manager. Each method provides a different view of the context. Once registered, the consumer receives notifications of context changes. In this prototype, the notifications sent are not in XML format, simply because the demonstration consumer presents the context information in a GUI. It does not have to do any complex processing or reasoning. Rather changes to each attribute are sent as individual notifications. It is likely that most context consumers will require XML notifications.

A separate validator was also developed for testing the validity of profiles against schemata that contain m^3 constraints as described in section 8.2. This validator is currently standalone. It has not been integrated with the Context Manager at this

stage. Although it would be a simple matter to integrate the validator, it is easier to demonstrate its purpose as a separate application.

10.3 Elvin Notification Formats

10.3.1 Context Source Side

The following documents the format of the Elvin notifications used between the context sources and the Context Manager. In the prototype, the context source emulator allocates the session ID. In a real platform for mobile computing, the context manager or a specialised session manager would do this.

10.3.1.1 Create

Key	Value (example)
m3.source.create	“1.0”
SESSIONID	“session123456”
data	“<?xml version=”1.0”?> etc

10.3.1.2 Update

Key	Value (example)
m3.source.update	“1.0”
SESSIONID	“session123456”
data	“<?xml version=”1.0”?> etc

10.3.1.3 Delete

Key	Value (example)
m3.source.delete	“1.0”
SESSIONID	“session123456”

The current implementation of the Context Manager does not cater for deletion of particular attributes within a session. Only the entire session can be deleted. The original version of the CC/PP model did allow for deletion within a given scope.

10.3.2 Context Consumer Side

10.3.2.1 Create

Key	Value (example)
m3.cm.create	“1.0”
SESSIONID	“session123456”

10.3.2.2 Add

Key	Value (example)
m3.cm.add	“1.0”
SESSIONID	“session123456”
COMPONENT	“PhysicalLocation”
ATTR	“City”
VAL	“Moskva”

10.3.2.3 Delete

Key	Value (example)
m3.cm.delete	“1.0”
SESSIONID	“session123456”

10.3.2.4 Class

Key	Value (example)
SESSIONID	“session123456”
CLASS	“A”, “B”, “C” etc
EVENT	“Entered” or “Exited”

Note that although the demonstration context consumer supports this notification, the Context Manager itself does not implement predefined context classes. The method for providing notifications of this sort is in place, but the specification of context classes themselves is not a trivial issue. An entire paper could be written on context classes alone.

10.3.2.5 Correlation

At present, correlations are implemented differently from regular notifications. This situation is expected to change as the Elvin correlation API becomes maturer. Instead of receiving a regular Elvin notification, a correlation object is received. Note also, that the expression language is undergoing redevelopment. Once the correlation API is stable, it will prove to be extremely useful in the Context Manager on several fronts.

10.4 Validator

The task of the validator is to validate profiles that contain attributes affected by m^3 constraints. The m^3 constraints can be found in Appendix B: m^3 Constraints Schema.

The validator is passed the profile to be validated along with the schemata against which the profile is to be validated. Both the profile and the schemata have been parsed and are in the form of RDF models. The algorithm used by the validator is outlined here:

```
For each attribute in the profile model
        if the attribute is an m3 attribute then
                gather all constraints from the schema models that apply to this attribute and the
                supertypes of this attribute
                check attribute against each constraint. report any errors
        end if
end for
```

The actual implementation defines a separate method for each constraint type. So the appropriate method is called when the constraint is retrieved from the constraint set.

10.5 Issues faced during implementation

10.5.1 CC/PP and RDF

The RDF API [16] is not very well designed. The Model class is the core class in the API, yet is not trivial to create a Model. A Model is really just a collection of RDF statements in the form of triples. To be precise, Model is an interface and ModelImpl is the concrete implementation. In this discussion, the distinction will not be made. Instead of being able to construct a Model in the obvious way (create the an empty Model and then call some sort of *read* method which takes the URL of an RDF document as argument), then programmer is forced to add statements one at a time to the Model.

Once the Model is populated with statements, the task is a little simpler. To search for particular statements, there is a *find* method that takes a subject, predicate and object as argument. The subject and predicate are both of type *Resource* and the object is of type *RDFNode*. *RDFNode*s and *Resource*s model the nodes of an RDF graph as seen in Figure 1 Example RDF graph of a CC/PP.

The RDF API does not have support for Bags and other RDF containers. This is a major shortcoming since containers are frequently used. Programmers are thus forced to implement Bag handling code for themselves.

It is **strongly** suggested that future development involving CC/PP or RDF utilises the Jena API [25]. Jena does have built-in support for all of RDF's container types (Bag, Alt and Seq). This makes it trivial to process containers and to add elements to the containers. It also has a more natural way of creating and populating an RDF Model. One simply needs to construct an empty model and then call the *read* method with a profile URL as argument. Jena also provides a more natural way of manipulating statements within a Model.

10.5.2 Elvin

Overall, Elvin is extremely easy to use. The publish-subscribe paradigm is very easy to program with and simple to learn. The complexity of the underlying transport protocols is hidden from the user, so there is no need to worry about creating sockets and such.

One drawback of using Elvin for the task of context updating is the fact that Elvin is non-persistent. This is an issue for context consumers. In general, consumers first subscribe to be notified of all creation events. Upon receipt of this creation event, the consumer must make a decision as to whether it wishes to receive add and delete notifications associated with this new context. If it does, then it must subscribe to these events as well. The problem is that between reception of the context creation event and the subscription to add and delete events, the Context Manager may have already sent one or more add or delete events. Since there is some amount of time where the consumer is not subscribed to these events, it is possible that these events will be missed.

One solution to this problem, and the solution chosen for the prototype implementation, is to have the Context Manager resend all context information pertaining to a particular context when a consumer registers its interest in add and delete events for that context. By doing this, the consumer can be guaranteed not to miss any events pertaining to a particular context. This solution requires that immediately after a consumer subscribes to add and delete events, it also emits a notification requesting that the Context Manager resends all context information for a particular context.

This is not an issue between the context sources and the Context Manager. This is because the Context Manager is interested in *all* events emitted by context sources. Filtering only occurs between the Context Manager and context consumers.

11 Future Work

There is still a plethora of work to accomplish in the field of context management in mobile environments. Two areas of work are outlined here.

11.1 Context Management and Expert Systems

As this project progressed, it became evident that there is a strong link between the field of Artificial Intelligence and context management for mobile environments. This relates not only to knowledge representations and formal context theory, but also to expert systems. The MYCIN project is well over ten years old, yet it exhibited some strong parallels with context management in mobile environments. These parallels are alluded to in Section 6.1. It could be profitable to further investigate the links between context management and expert systems. It is believed expert systems could lend a great deal of knowledge to this field.

11.2 Context Classes

It was mentioned in Section 10.3.2.5 that a whole paper could be written on ways to define context classes. Context classes are useful for a number of reasons. They provide an abstraction for consumers of context information. Instead of dealing with a number of separate attributes, context consumers need only deal with context class identifiers. Context classes can also be used by the Context Manager itself in certain situations. Although this paper proposes a way in which context classes may be implemented, it does not solve the problem of defining context classes in the first place. Which attributes should be grouped together? How does non-quantifiable data such as location names fit with the idea of a context class? How does one specify ranges of acceptable values for each context class? Does one set of context classes suffice for every conceivable situation? Work also needs to be done on allowing context consumers to specify their own context classes.

12 Conclusion

This paper has presented an in depth look at Context Management in Mobile Environments. In particular it evaluated CC/PP as a means of describing context. The following summarises the outcomes of the paper with respect to its stated goals.

1. To identify the strengths and weaknesses of CC/PP.

   The RDF model can often be over complex. However, the serialisation into XML is a positive factor. The CC/PP model itself makes it easy to match device profiles with document profiles.

2. To discover whether CC/PP is capable of describing context beyond device capability and user preference. It was subsequently found that CC/PP was indeed capable of describing richer context information.

   CC/PP is capable of describing more than it was originally intended for. However, there is some doubt as to whether it makes sense to associate session IDs with all context information.

3. The introduction of certain constraints between attributes. A constraint schema was developed in conjunction with a validator to process these constraints.

   Two methods of constraint specification were outlined and evaluated. A validator capable of processing profiles that have these constraints applied to them was developed and successfully tested.

4. The development of an exchange protocol using Elvin. This was done for both context sources and context consumers.

   An asynchronous exchange protocol using Elvin was designed and implemented. This protocol can be used in situations where content is pushed from the server, or where context sources are separate from the mobile devices.

5. To formulate an idea as to how context classes can be easily implemented using Elvin event correlations. This was performed in Section 9.

   An interface for specifying custom and static context classes was designed and implemented.

6. Finally, to scour the literature and come to a conclusion as to what context means in a mobile environment. Section 3.1 provided a definition based upon work done in the fields of AI and Human Computer Interaction.

   Context was decided to be ‘any information at all that can help determine the situation of an object, where an object can be a user, a device or any other entity being tracked by the system.’

It is believed CC/PP will be supported by a large number of device manufacturers. CC/PP can be used beyond its intentions to describe general context information. This paper has described CC/PP in detail and shown how it can be utilised within a context manager.

.

13 References

[1] Gregory D. Abowd et al. “Context Awareness in Wearable and Ubiquitous Computing”. Graphics, Visualisation and Useability Center, Georgia Institute of Technology, 1997

[2] CRC for Distributed Systems Technology. “Mobile Enterprise Architecture”. The m^3 Research Home Page. http://www.dstc.edu.au/m3

[3] The World Wide Web Consortium CC/PP Working Group. “CC/PP exchange protocol based on HTTP Extension Framework”. (Note). June 1999. http://www.w3.org/TR/NOTE-CCPPexchange

[4] The Internet Society, Network Working Group. “RFC 2774: An HTTP Extension Framework”. February 2000

[5] Segall, B. et al. “Content Based Routing with Elvin4”. AUUG2K Conference. Canberra. June 2000. http://elvin.dstc.edu.au/doc/papers/auug2k/auug2k.html

[6] Bharghavan, V. "Challenges and solutions to adaptive computing and seamless mobility over heterogeneous wireless networks", International Journal on Wireless Personal Communications, 1996

[7] Brian D. Noble, Morgan Price and M. Satyanarayanan. “A Programming Interface for Application-Aware Adaptation in Mobile Computing.” Proceedings of the Second USENIX Symposium on Mobile and Location-Independent Computing. Ann Arbor, MI, April 1995

[8] R. Want, A. Hopper, V. Falcao, and J. Gibbons, "The active badge location system," ACM Transactions on Information Systems, vol. 10, pp. 91--102, Jan. 1992

[9] J. McCarthy and S. Buvac. Formalizing context (expanded notes). Technical Note STAN-CS-TN-94-13, Computer Science Department, Stanford University, Stanford, CA, 1994

[10] The World Wide Web Consortium CC/PP Working Group. “Composite Capabilities/Preference Profiles: Requirements and Architecture”. (Draft). July 2000. http://www.w3.org/TR/2000/WD-CCPP-ra-20000721

[11] The World Wide Web Consortium CC/PP Working Group. “Composite Capabilities/Preference Profiles (CC/PP): Structure”. (Draft). July 2000. http://www.w3.org/TR/2000/WD-CCPP-struct-20000721/

[12] The World Wide Web Consortium CC/PP Working Group. "CC/PP Attribute Vocabularies". (Draft). July 2000. http://www.w3.org/TR/2000/WD-CCPP-vocab-20000721/

[13] Buvac, S. (1996), `Resolving Lexical Ambiguity using a Formal Theory of Context', in Van Deemter and Peters (Eds.) Semantic Ambiguity and Underspecification, CSLI Publications, Stanford 1996

[14] The United States Department of Defense. "An Introduction to the United States Department of Defense". http://www.defenselink.mil/pubs/dod101/largest.html

[15] The World Wide Web Consortium RDF Working Group. "Resource Description Framework (RDF) Model and Syntax Specification". (Recommendation). February 1999. http://www.w3.org/TR/REC-rdf-syntax/

[16] Sergey Melnik. "RDF API Draft". The RDF API homepage. http://www-db.stanford.edu/~melnik/rdf/api.html

[17] Michael Henderson. "A Framework for Event Correlation". October 1999. http://elvin.dstc.edu.au/correlation/doc/

[18] Jakobson, G. and Pathak, G. "Correlation DialTone – Building Internet-Based Distributed Event Correlation Services". 3rd IFIP/GI International Conference on Trends Towards a Universal Service Market. September 2000.

[19] The Internet Society, Network Working Group. "RFC 2570: Introduction to Version 3 of the Internet-standard Network Management Framework ". April 1999

[20] Fensel, D. et al. "OIL in a Nutshell". Knowledge Acquisition, Modeling, and Management, Proceedings of the European Knowledge Acquisition Conference (EKAW-2000). October 2000

[21] Dey, A. et al. "The Conference Assistant: Combining context-awareness with wearable computing". Proceedings of the 3rd International Symposium on Wearable Computers. 1999.

[22] Rakotonirainy, A. et al. "Context-Awareness for the Mobile Environment". January 2000.

[23] Harcourt, A. "Location Management in a Ubiquitous Computing Environment". Honours Thesis. University of Queensland. October 2000.

[24] The World Wide Web Consortium CC/PP Working Group. "Composite Capability/Preference Profiles (CC/PP): A user side framework for content negotiation" (Note). July 1999. http://www.w3.org/TR/NOTE-CCPP/

[25] McBride, B. "Jena – A Java API For RDF". http://www-uk.hpl.hp.com/people/bwm/rdf/jena/index.htm

[26] Wood, K. et al. "Global Teleporting with Java: Towards Ubiquitous Personalised Computing ". Nomadics '96, San Jose, USA. March 1996

[27] van Melle, W. "The Structure of the MYCIN System". *Rule-Based Expert Systems: The MYCIN experiments of the Stanford Heuristic Programming Project*. ed Buchanan, B and Shortliffe, E. Addison-Wesley Publishing, 1984.

[28] Arkins, R. "Persistent Elvin For Mobile Devices". Honours Thesis. University of Queensland. October 2000.

14 Appendix A: Terminology

Adaptability

Adaptation is the response of an application or underlying platform (such as m^3) to changes in the context. Adaptation can be thought of as the behaviour of an entity in response to changes in external factors.

CC/PP

Composite Capabilities/Preference Profiles. A framework developed by W3C for content negotiation.

CC/PP exchange

A protocol based upon the HTTP extension standard used for transferring CC/PP in a web enabled environment.

Context

1. Any attribute or factor that may influence the interpretation of information. For example, the statement 'It's cold' has a different meaning depending on whether or not I'm holding a frozen fish in my hand. If I am holding a frozen fish in my hand, then the statement is interpreted as 'The fish is cold'. If I'm not holding a frozen fish in my hand (or indeed any other cold object), then the statement should probably be interpreted as 'It's a cold day' or something similar.

2. A collection of attributes and their values that describe context as defined above.

In a mobile environment, context may be static or dynamic. In static context, values of attributes are set at startup time and they do not alter. Therefore dynamic adaptation cannot be supported. In dynamic context, attribute values can change during execution. This allows dynamic adaptation to be performed.

Elvin

Elvin is an event notification service developed by the DSTC.

HTTP

Hypertext Transfer Protocol. The protocol by which clients and servers on the world wide web communicate.

m^3

The Context Manager is just one component of the m^3 architecture. m^3 proposes an adaptable architecture for mobile devices. In discussing Context Management, m^3 will sometimes be used to give a concrete example.

Notification

Notification occurs when a component is alerted of changed conditions at the mobile device side.

Profile

A document that provides metadata about a device, a user, a document or even an application. Profiles can be matched so that an optimum representation of content can be provided.

QoS

An abbreviation for Quality of Service. The actual or required fidelity with which content is expected to arrive over a network connection.

RDF

Resource Description Framework. An XML based format for describing documents or other objects. It was developed be W3C.

Subscription

In Elvin terminology, a client can ask to be notified of particular events. This constitutes a subscription.

Transcode

The process by which content is translated from one form to another. This can be as simple as translating HTML to WML or as complex as converting an audio stream to a text based representation.

URI

Universal Resource Indicator. Somewhat analogous to a Universal Resource Locator, a URI references a some kind of object. The object need not reside on the world wide web.

W3C

The World Wide Web Consortium. The originator of many web based standards and recommendations such as RDF.

15 Appendix B: m^3 Constraints Schema

```
<?xml version="1.0"?>

<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:ccpp="http://www.w3.org/2000/07/04-ccpp#"
 xmlns:m3constraints="http://staff.dstc.edu.au/rickyr/m3-constraint-
schema#">
 <!-- The last namespace is this schema itself -->


 <!-- ************************ -->
 <!-- m3 relational constraints -->
 <!-- ************************ -->

 <m3constraints:RelationalConstraint rdf:ID='xor'>
  <rdfs:label>xor</rdfs:label>
  <rdfs:domain rdf:resource='http://www.w3.org/2000/07/04-
ccpp#Attribute'/>
  <rdfs:range rdf:resource='http://www.w3.org/2000/07/04-
ccpp#Attribute'/>
  <rdfs:comment>
   The "xor" relation
  </rdfs:comment>
 </m3constraints:RelationalConstraint>

 <m3constraints:RelationalConstraint rdf:ID='or'>
  <rdfs:label>or</rdfs:label>
  <rdfs:domain rdf:resource='http://www.w3.org/2000/07/04-
ccpp#Attribute'/>
  <rdfs:range rdf:resource='http://www.w3.org/2000/07/04-
ccpp#Attribute'/>
  <rdfs:comment>
   The "or" relation
  </rdfs:comment>
 </m3constraints:RelationalConstraint>

 <m3constraints:RelationalConstraint rdf:ID='and'>
  <rdfs:label>and</rdfs:label>
  <rdfs:domain rdf:resource='http://www.w3.org/2000/07/04-
ccpp#Attribute'/>
  <rdfs:range rdf:resource='http://www.w3.org/2000/07/04-
ccpp#Attribute'/>
  <rdfs:comment>
   The "and" relation
  </rdfs:comment>
 </m3constraints:RelationalConstraint>

<!-- ********************** -->
<!-- Cardinality Constraints -->
<!-- ********************** -->

 <m3constraints:CardinalityConstraint rdf:ID='lessThan'>
  <rdfs:label>Cardinality: less than</rdfs:label>
  <rdfs:domain rdf:resource='http://www.w3.org/2000/07/04-
ccpp#Attribute'/>
```

```
  <rdfs:range rdf:resource='http://www.w3.org/2000/07/04-
ccpp#Integer'/>
  <rdfs:comment>
   The "less than" numerical relationship
  </rdfs:comment>
 </m3constraints:CardinalityConstraint>

 <m3constraints:CardinalityConstraint rdf:ID='greaterThan'>
  <rdfs:label>Cardinality: greater than</rdfs:label>
  <rdfs:domain rdf:resource='http://www.w3.org/2000/07/04-
ccpp#Attribute'/>
  <rdfs:range rdf:resource='http://www.w3.org/2000/07/04-
ccpp#Integer'/>
  <rdfs:comment>
   The "greater than" numerical relationship
  </rdfs:comment>
 </m3constraints:CardinalityConstraint>

 <m3constraints:CardinalityConstraint rdf:ID='lessThanOrEqualTo'>
  <rdfs:label>Cardinality: less than or equal to</rdfs:label>
  <rdfs:domain rdf:resource='http://www.w3.org/2000/07/04-
ccpp#Attribute'/>
  <rdfs:range rdf:resource='http://www.w3.org/2000/07/04-
ccpp#Integer'/>
  <rdfs:comment>
   The "less than or equal to" numerical relationship
  </rdfs:comment>
 </m3constraints:CardinalityConstraint>

 <m3constraints:CardinalityConstraint rdf:ID='greaterThanOrEqualTo'>
  <rdfs:label>Cardinality: greater than or equal to</rdfs:label>
  <rdfs:domain rdf:resource='http://www.w3.org/2000/07/04-
ccpp#Attribute'/>
  <rdfs:range rdf:resource='http://www.w3.org/2000/07/04-
ccpp#Integer'/>
  <rdfs:comment>
   The "greater than or equal to" numerical relationship
  </rdfs:comment>
 </m3constraints:CardinalityConstraint>

 <m3constraints:CardinalityConstraint rdf:ID='equalTo'>
  <rdfs:label>Cardinality: equals</rdfs:label>
  <rdfs:domain rdf:resource='http://www.w3.org/2000/07/04-
ccpp#Attribute'/>
  <rdfs:range rdf:resource='http://www.w3.org/2000/07/04-
ccpp#Integer'/>
  <rdfs:comment>
   The "equal to" numerical relationship
  </rdfs:comment>
 </m3constraints:CardinalityConstraint>

 <m3constraints:CardinalityConstraint rdf:ID='notEqualTo'>
  <rdfs:label>Cardinality: notEqualTo</rdfs:label>
  <rdfs:domain rdf:resource='http://www.w3.org/2000/07/04-
ccpp#Attribute'/>
  <rdfs:range rdf:resource='http://www.w3.org/2000/07/04-
ccpp#Integer'/>
  <rdfs:comment>
   The "not equal to" numerical relationship
  </rdfs:comment>
 </m3constraints:CardinalityConstraint>
```

```
 <!-- ***************** -->
 <!-- Update constraint -->
 <!-- ***************** -->

 <rdfs:ConstraintProperty ID='updateMethod'>
  <rdfs:label>ConstraintProperty</rdfs:label>
  <rdfs:domain rdf:resource='http://www.w3.org/2000/07/04-
ccpp#Attribute'/>
  <rdfs:range rdf:resource='http://staff.dstc.edu.au/rickyr/m3-
constraint-schema#UpdateMethod'/>
  <rdfs:comment>
   The method by which an attribute can be updated
  </rdfs:comment>
 </rdfs:ConstraintProperty>

 <!-- ***************************** -->
 <!-- Classes needed for constraints -->
 <!-- ***************************** -->

 <!-- Attribute updating semantics -->

 <rdfs:Class rdf:ID='UpdateMethod'/>

 <m3constraints:UpdateMethod rdf:ID='locked'>
  <rdfs:comment>
   The attribute takes the first assigned value
   and is not overridden from thereon in.
  </rdfs:comment>
 </m3constraints:UpdateMethod>

 <m3constraints:UpdateMethod rdf:ID='override'>
  <rdfs:comment>
   The attribute takes the latest value assigned to it.
  </rdfs:comment>
 </m3constraints:UpdateMethod>

 <m3constraints:UpdateMethod rdf:ID='append'>
  <rdfs:comment>
   The attribute takes on a list of values.
   This update method is closely related to the cardinality
constraints.
  </rdfs:comment>
 </m3constraints:UpdateMethod>

 <!-- Cardinality class -->

 <rdfs:Class rdf:ID='CardinalityConstraint'>
  <rdfs:label>CardinalityConstraint</rdfs:label>
  <rdfs:subClassOf rdf:resource='http://www.w3.org/2000/01/rdf-
schema#ConstraintProperty'/>
  <rdfs:comment>
   Constrains the number of values an attribute may have.
  </rdfs:comment>
 </rdfs:Class>

 <!-- relational constraints -->

 <rdfs:Class rdf:ID='RelationalConstraint'>
  <rdfs:label>Realtional Constraint</rdfs:label>
```

```
  <rdfs:subClassOf rdf:resource='http://www.w3.org/2000/01/rdf-
schema#ConstraintProperty'/>
  <rdfs:comment>
   relations specify constraints like "this AND that must exist"
  </rdfs:comment>
 </rdfs:Class>
</rdf:RDF>
```

16 Appendix C: Examples of extended CC/PP vocabulary

16.1 m³-project-schema

```
<?xml version="1.0"?>

<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:ccpp="http://www.w3.org/2000/07/04-ccpp#"
 xmlns:m3constraints="http://staff.dstc.edu.au/rickyr/m3-constraint-schema#"
 xmlns:m3projects="http://staff.dstc.edu.au/rickyr/m3-project-schema#">
 <!-- The last namespace is this schema itself -->

 <!-- ********************************************************** -->
 <!-- This schema gives a demo of how m3 constraints can be used. -->
 <!-- It also provides a demo schema for m3 projects for policy.  -->
 <!-- ********************************************************** -->

 <!-- Project Class -->

 <rdfs:Class rdf:ID="Project">
  <rdfs:label>Project</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/07/04-ccpp#Component"/>
  <rdfs:comment>
   A Project is a component that provides context information about projects
   within an organisation. The source of this context information will most
   likely be the Policy Manager.
  </rdfs:comment>
 </rdfs:Class>
```

***NOTE**: The following lines of XML were unintentionally omitted from the submitted version of the thesis. These lines complete the m3-project-schema:*

```
<!-- Projects have scientists -->

 <ccpp:Attribute rdf:ID="scientists">
  <rdfs:domain rdf:resource="http://staff.dstc.edu.au/rickyr/m3-project-schema#Project"/>
  <m3constraints:and rdf:resource="http://staff.dstc.edu.au/rickyr/m3-project-schema#leader"/>
 </ccpp:Attribute>

 <!-- Projects have a leader -->

 <ccpp:Attribute rdf:ID="leader">
  <rdfs:label>Project: leader</rdfs:label>
  <rdfs:domain rdf:resource="http://staff.dstc.edu.au/rickyr/m3-project-schema#Project"/>
 </ccpp:Attribute>

</rdf:RDF>
```

16.2 m³-test-schema

```
<?xml version="1.0"?>

<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:ccpp="http://www.w3.org/2000/07/04-ccpp#"
 xmlns:m3constraints="http://staff.dstc.edu.au/rickyr/m3-constraint-schema#">

 <!-- Define a subProperty of the project schema's "scientists" attribute -->

 <ccpp:Attribute rdf:ID="testScientists">
  <rdfs:subPropertyOf rdf:resource="http://staff.dstc.edu.au/rickyr/m3-project-schema#scientists"/>
  <m3constraints:updateMethod rdf:resource="http://staff.dstc.edu.au/rickyr/m3-constraint-
schema#append"/>
  <m3constraints:greaterThanOrEqualTo>6</m3constraints:greaterThanOrEqualTo>
 </ccpp:Attribute>
</rdf:RDF>
```

16.3 m³-test-profile

```
<?xml version="1.0"?>

<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:ccpp="http://www.w3.org/2000/07/04-ccpp#"
 xmlns:m3constraints="http://staff.dstc.edu.au/rickyr/m3-constraint-schema#"
 xmlns:m3projects="http://staff.dstc.edu.au/rickyr/m3-project-schema#"
 xmlns:m3test="http://staff.dstc.edu.au/rickyr/m3-test-schema#">
 <!-- The last namespace is this schema itself -->

 <rdf:Description about="TestProject">
  <rdf:type rdf:resource="http://staff.dstc.edu.au/rickyr/m3-project-schema#Project"/>
  <m3test:testScientists>
   <rdfs:Bag>
    <rdf:li>Ricky Robinson</rdf:li>
    <rdf:li>Karsten Schulz</rdf:li>
    <rdf:li>Andry Rakotonirainy</rdf:li>
    <rdf:li>Varuni Witana</rdf:li>
    <rdf:li>Seng Wai Loke</rdf:li>
   </rdfs:Bag>
  </m3test:testScientists>
  <m3projects:leader>Karsten Schulz</m3projects:leader>
 </rdf:Description>
</rdf:RDF>
```

17 Appendix D: Context Manager Demonstrations

This section includes the code for the M3Validator and a test class for the Validator. Also included is one demonstration that shows how to use the Context Manager interface. The example included emulates both a context source and a context consumer.

17.1 M3Validator

```
package au.edu.dstc.rdf.m3;

import java.util.*;

import org.w3c.rdf.model.*;
import org.w3c.rdf.implementation.model.*;
import org.w3c.rdf.vocabulary.rdf_syntax_19990222.*;
import org.w3c.rdf.vocabulary.rdf_schema_19990303.RDFS;

/**
 * M3Validator.java
 *
 * Takes a Model of a profile and checks it against
 * one or m3 schema models.
 *
 * @author <a href="mailto:rickyr@dstc.edu.au">Ricky Robinson</a>
 * @version 1.0
 */

public class M3Validator {

    public static final String M3CONSTRAINTS =
      "http://staff.dstc.edu.au/rickyr/m3-constraint-schema";

    public static final String M3HOME =
"http://staff.dstc.edu.au/rickyr/";

    /**
     * The constraints
     */

    public static final String SUBPROPERTY =
      "http://www.w3.org/2000/01/rdf-schema#subPropertyOf";

    public static final String XOR = M3CONSTRAINTS+"#xor";

    public static final String OR =  M3CONSTRAINTS+"#or";

    public static final String AND =  M3CONSTRAINTS+"#and";

    public static final String LT =  M3CONSTRAINTS+"#lessThan";

    public static final String GT =  M3CONSTRAINTS+"#greaterThan";

    public static final String LTOET =
M3CONSTRAINTS+"#lessThanOrEqualTo";
```

```
    public static final String GTOET =
M3CONSTRAINTS+"#greaterThanOrEqualTo";

    public static final String ET =  M3CONSTRAINTS+"#equalTo";

    public static final String NET =  M3CONSTRAINTS+"#notEqualTo";

    public static final String UPDATE =
M3CONSTRAINTS+"#updateMethod";

    private Model profile;

    private Map schemas;

    private List errors = new ArrayList();

    /**
     * Construct a validator that will validate profile
     * against the Map of schemas.
     */
    public M3Validator(Model profile, Map schemas) {
      this.profile = profile;
      this.schemas = schemas;
    }

    /**
     * Validate <code>profile</code> against the set of schemas in
     * the Map <code>schemas</code>.
     *
     * @returns the List of errors
     */
    public List validate() {

      if(schemas.size() == 0) {
          errors.add("There are no schemas to test against");
          return errors;
      }

      try {

          /*
           * For each attribute in each component
           * check to see if the attribute satisfies
           * the constraints associated with it.
           */
          Enumeration e = profile.elements();
          while(e.hasMoreElements()) {
            Statement s = (Statement)e.nextElement();
            // Make sure the attribute is an m3 attribute
            Resource r = s.predicate();
            String uri = r.getURI();
            if(uri.startsWith(M3HOME)) {

                // Get the set of constraints that apply to this
attribute
                List constraints = gatherConstraints(s);
                Iterator i = constraints.iterator();
                while(i.hasNext()) {
                  Statement constraint = (Statement)i.next();
                  String pred = constraint.predicate().getURI();
                  if(pred.equals(XOR)) {
```

```
                    processXOR(s, constraint);
                }else if(pred.equals(OR)) {
                    processOR(s, constraint);
                }else if(pred.equals(AND)) {
                    processAND(s, constraint);
                }else if(pred.equals(LT)) {
                    processCardinality(s, constraint);
                }else if(pred.equals(GT)) {
                    processCardinality(s, constraint);
                }else if(pred.equals(LTOET)) {
                    processCardinality(s, constraint);
                }else if(pred.equals(GTOET)) {
                    processCardinality(s, constraint);
                }else if(pred.equals(ET)) {
                    processCardinality(s, constraint);
                }else if(pred.equals(NET)) {
                    processCardinality(s, constraint);
                }else if(pred.equals(UPDATE)) {
                    processUPDATE(s, constraint);
                }
              }
            }
          } catch (ModelException ex) {
              errors.add(ex);
          } finally {
              printErrors();
              return errors;
          }
        }

        /**
         * Find all the constraints that apply to the attribute
<i>uri</i>
         * and store them in a list. Return the List.
         *
         * This also supports m3 constraints that might be inherited from
         * super properties.
         */
        private List gatherConstraints(Statement s) throws ModelException
{

          List l = new ArrayList();
          String uri = s.predicate().getURI();
          String base = uri.substring(0, uri.indexOf('#'));

          Model mod = (Model)schemas.get(base);

          if(mod == null) {
              errors.add("No such schema in schema list: " + base);
              return l; // return empty list
          }

          // The predicate of the statement is the subject
          // of a constraint statement.
          Resource res = new
ResourceImpl(uri.substring(uri.indexOf('#')+1));
          Model result = mod.find(res, null, null);

          if(result.isEmpty()) {
              errors.add("There is no such attribute " + uri);
```

```
  }

  Enumeration e = result.elements();
  while(e.hasMoreElements()) {
      Statement t = (Statement)e.nextElement();
      Resource con = t.predicate();
      String g = con.getURI();

      if(g.equals(XOR)   ||
         g.equals(OR)    ||
         g.equals(AND)   ||
         g.equals(LT)    ||
         g.equals(GT)    ||
         g.equals(LTOET) ||
         g.equals(GTOET) ||
         g.equals(ET)    ||
         g.equals(NET)   ||
         g.equals(UPDATE)) {
        l.add(t);

      } else if(g.equals(SUBPROPERTY)) {
        Resource superProperty =
            new ResourceImpl(t.object().getLabel());
        Statement stmt = new StatementImpl(new ResourceImpl(""),
                                   superProperty,
                                   new ResourceImpl(""));
        l.addAll(gatherConstraints(stmt));
      }
  }
  return l;
}

/**
 * One or the other but not both, or neither.
 */
private boolean processXOR(Statement s, Statement constraint)
throws ModelException {
  boolean result = true;
  String obj = constraint.object().getLabel();
  Enumeration e = profile.elements();
  while(e.hasMoreElements()) {
      Statement t = (Statement)e.nextElement();
      String pred = t.predicate().getURI();
      if(obj.equals(pred)) {
        relationError("XOR",
                    s.predicate().getURI(),
                    constraint.object().getLabel());
        result = false;
      }
  }
  return result;
}

/**
 * Or is always satisfied
 * Note that just like the other relationships
 * this OR is not exactly a boolean OR. This OR is
 * one or the other or both or neither.
 */
private boolean processOR(Statement s, Statement constraint) {
  return true;
```

```
    }

    private boolean processAND(Statement s, Statement constraint)
    throws ModelException {
      boolean result = true;
      RDFNode obj = constraint.object();
      Resource convert = new ResourceImpl(obj.getLabel());
      Model m = profile.find(null, convert, null);
      if(m.isEmpty()) {
          relationError("AND",
s.predicate().getURI(),obj.getLabel());
          result = false;
      }
      return result;
    }

    /**
     * This stuff would be much easier using Jena because it has
     * built in support for containers such as Bags.
     */

    private boolean processCardinality(Statement s, Statement
constraint)
    throws ModelException {
      int count = 0;
      /*
       * Find the number of elements the attribute actually has
       */
      Model m =
          profile.find(new ResourceImpl(s.object().getLabel()), null,
null);
      Enumeration e = m.elements();
      while(e.hasMoreElements()) {
          Statement t = (Statement)e.nextElement();
          if(!t.predicate().equals(RDF.type)) {
            count++;
          }
      }

      /*
       * Compare count to the constraint
       */
      int cons = Integer.parseInt(constraint.object().getLabel());
      boolean passed = true;
      String pred = constraint.predicate().getURI();

      if(pred.equals(LT)) {
          passed = count < cons;
          if(!passed) {
            cardinalityError("less than", s.predicate().getURI(),
                        count, cons);
          }
      }else if(pred.equals(GT)) {
          passed = count > cons;
          if(!passed) {
            cardinalityError("greater than", s.predicate().getURI(),
                        count, cons);
          }
      }else if(pred.equals(LTOET)) {
          passed = count <= cons;
          if(!passed) {
```

```
            cardinalityError("less than or equal to",
                        s.predicate().getURI(),
                        count, cons);
            }
        }else if(pred.equals(GTOET)) {
            passed = count >= cons;
            if(!passed) {
              cardinalityError("greater than or equal to",
                        s.predicate().getURI(),
                        count, cons);
            }
        }else if(pred.equals(ET)) {
            passed = count == cons;
            if(!passed) {
              cardinalityError("equal to", s.predicate().getURI(),
                        count, cons);
            }
        }else if(pred.equals(NET)) {
            passed = count != cons;
            if(!passed) {
              cardinalityError("not equal to", s.predicate().getURI(),
                        count, cons);
            }
        }

        return passed;
    }

    private boolean processUPDATE(Statement s, Statement constraint)
{
        // Not implemented
        return true;
    }

    private void printErrors() {
        Iterator i = errors.iterator();
        while(i.hasNext()) {
            String error = (String)i.next();
            System.err.println();
            System.err.println(error);
        }
    }

    private void relationError(String constraintType,
                        String attr1,
                        String attr2) {
        errors.add("The " + constraintType + " constraint between " +
                   attr1 + " and " +
                   attr2 + " has been violated.");
    }

    private void cardinalityError(String constraintType,
                          String attr,
                          int actual,
                          int constraint) {
        errors.add("The " + constraintType + " constraint on " +
                   attr + " has been violated. The actual count is
" +
                   actual + " and the constraint is " + constraint
+
                   ".");
```

```
    }
}
```

```
package au.edu.dstc.rdf.m3;

import java.net.*;
import java.util.*;

import org.w3c.rdf.model.*;

/**
 * ValidatorTest.java
 *
 * Test the m3 constraints validator.
 *
 * @author <a href="mailto:rickyr@dstc.edu.au">Ricky Robinson</a>
 * @version 1.0
 */

public class ValidatorTest {

    public static void main(String[] args) {
      Map schemas = new HashMap();
      CCPPProcessor processor = new CCPPProcessor();
      Model m;
      URL profile = null;
      M3Validator validator;

      if(args.length < 1) {
          System.err.println("Usage: java
au.edu.dstc.rdf.m3.ValidatorTest" +
                         " profileURL [schemaURL]*");
          System.exit(1);
      }
      try {
          try {
            profile = new URL(args[0]);
          } catch (Exception e) {
            profile = new URL("file", null, args[0]);
          }

          m = processor.createModel(profile);
          if(args.length > 1) {
            for(int i = 1; i < args.length; i++) {
                URL surl;
                try {
                  surl = new URL(args[i]);
                } catch (Exception ex) {
                  surl = new URL("file", null, args[i]);
                }
                Model schema = processor.createModel(surl);
                schema.setSourceURI(surl.toString());
                schemas.put(surl.toString(), schema);
            }
          }

          validator = new M3Validator(m, schemas);
```

```
          List errors = validator.validate();
          if(errors.isEmpty()) {
            System.out.println("The profile satisfies all m3
constraints");
          } else {
            int errs = errors.size();
            System.out.println();
            System.out.println("There is " + errs +
                          (errs == 1 ? " error " : " errors ") +
                          "altogether.");
          }
          System.exit(0);
      } catch (Exception exc) {
          System.err.println(exc);
          System.exit(1);
      }
    }
}
```

17.2 Demonstration GUI

```
package au.edu.dstc.contextgui;

import java.util.*;
import javax.swing.*;
import javax.swing.tree.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.net.URL;
import java.io.StringReader;

import org.w3c.rdf.model.*;
import org.w3c.rdf.util.*;
import org.w3c.rdf.implementation.model.*;

import au.edu.dstc.mobility.cm.*;

import org.elvin.correlation.*;

/**
 * SourceGUI.java
 *
 * A GUI that emulates a context source.
 * This GUI is also a ClientGUI. It receives update notifications as
well.
 *
 * @author <a href="mailto:rickyr@dstc.edu.au">Ricky Robinson</a>
 * @version 1.0 October 2000
 */

public class SourceGUI extends JFrame
    implements ContextListener, ActionListener ,
TreeSelectionListener {

    private CCPPAdaptor adaptor;

    /**
     * For processing profiles
     */
```

```
    private CCPPProcessor ccpp = new CCPPProcessor();

    /**
     * We can emulate more than one context source.
     * So we need a map of sessions and their profiles
     */
    private Map models = new HashMap();

    private ClientStub stub;

    private ContextListener mediator;

    private JSplitPane leftRight;

    private JSplitPane upDown;

    private JTree tree;

    private DefaultTreeModel treeModel;

    private DefaultMutableTreeNode root;

    private InfoPanel infoPanel;

    private JTextArea rdfBox;

    private JButton rdfButton;

    private JMenuItem profileItem;

    private static final int WIDTH = 640;

    private static final int HEIGHT = 480;

    public SourceGUI(String title, CCPPAdaptor adaptor, ClientStub
stub) {
      super(title);
      addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                System.exit(0);
            }
          });

      // For sending updates
      this.adaptor = adaptor;

      // For receiving updates
      this.stub = stub;
      stub.setContextListener(this);
      stub.registerForCreation();

      root = new DefaultMutableTreeNode(title);
      treeModel = new DefaultTreeModel(root);
      tree = new JTree(treeModel);
      tree.getSelectionModel().setSelectionMode
          (TreeSelectionModel.SINGLE_TREE_SELECTION);
      tree.setShowsRootHandles(true);
      tree.addTreeSelectionListener(this);


      JScrollPane treeScroller = new JScrollPane(tree);
```

```
infoPanel = new InfoPanel();

JScrollPane infoScroller = new JScrollPane(infoPanel);
infoScroller.setMinimumSize(infoPanel.getMinimumSize());

leftRight = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                     treeScroller,
                     infoScroller);

rdfButton = new JButton("Send profile difference");
rdfButton.addActionListener(this);

rdfBox = new JTextArea();
rdfBox.setEditable(true);
JScrollPane rdfScroller = new JScrollPane(rdfBox);

rdfScroller.setPreferredSize(new Dimension(WIDTH,
                              HEIGHT/3-rdfButton.getHeight()));

JPanel rdfPanel = new JPanel();
rdfPanel.setLayout(new BorderLayout());

rdfPanel.add(rdfButton, BorderLayout.SOUTH);

rdfPanel.add(rdfScroller, BorderLayout.NORTH);

upDown = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
                  leftRight,
                  rdfPanel);

getContentPane().setLayout(new BorderLayout());
getContentPane().add(upDown, BorderLayout.CENTER);

JMenu menu = new JMenu("File");

profileItem = new JMenuItem("New Session");

profileItem.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
          addSession();
      }
    });

menu.add(profileItem);

JMenuItem exitItem = new JMenuItem("Exit");

exitItem.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
          System.exit(0);
      }
    });

menu.add(exitItem);

JMenuBar menuBar = new JMenuBar();
menuBar.add(menu);

setJMenuBar(menuBar);
pack();
```

```
    }

    // ActionListener for update button

    public void actionPerformed(ActionEvent e) {
      send(rdfBox.getText());
      //rdfBox.setText(null);
    }

    private void addSession() {
      // Seed a random number generator with a random number;
      Random r = new Random((int)(Math.random() * 1000000));
      String session = "session" + r.nextInt(1000000);
      URLDialog dialog = new URLDialog(this, "Create session " +
session);
      dialog.setVisible(true);
      String answer = dialog.getURL();
      if (answer != null) {
          URL url;
          try {
            try {
                url = new URL(answer);
            } catch (Exception f) {
                url = new URL("file", null, answer);
            }
            Model m = ccpp.createModel(url);
            models.put(session, m);
            adaptor.createContext(session, m);
          } catch(Exception x) {
            System.err.println(x);
          }
      } else {
          JOptionPane.showMessageDialog(this,
                                "The session will be based " +
                                "on an empty profile.");
          Model m = new ModelImpl();
          models.put(session, m);
          adaptor.createContext(session, m);
      }

    }

    // The ContextGUI methods

    public void send(String xml) {
      DefaultMutableTreeNode parent;
      String session;
      Model m;
      Statement s;

      if(xml == null || xml.equals(""))
          return;

      DefaultMutableTreeNode n = (DefaultMutableTreeNode)
          tree.getLastSelectedPathComponent();

      if(n == null) {
          JOptionPane.showMessageDialog(this,
                                        "You must select a
component from " +
```

```
                                "the tree.");
            return;
        }

        if(n == root) {
            JOptionPane.showMessageDialog(this,
                                          "To add a new session, use
the " +
                                "File->New Session command.");
            return;
        }

        TreePath path = tree.getSelectionPath();
        DefaultMutableTreeNode theNode =
            (DefaultMutableTreeNode)path.getPathComponent(1);

        ContextObject co = (ContextObject)theNode.getUserObject();
        session = co.getProfile();

        Model update = ccpp.createModel(new StringReader(xml));

        adaptor.updateContext(session, update);

    }

    // The ContextGUI methods

    public void add(ContextObject obj) {
        DefaultMutableTreeNode parent;

        parent = findParent(obj, root);
        if(parent == null) {
            System.err.println("Parent is null");
            return;
        }

        if(parent == root) { // New session

            stub.registerForId(obj.getProfile());
        }

        System.err.println(parent.getUserObject());

        Enumeration e = parent.children();
        DefaultMutableTreeNode childNode = null;
        boolean exists = false;
        while(e.hasMoreElements() && !exists) {
            childNode = (DefaultMutableTreeNode)e.nextElement();
            ContextObject co =
(ContextObject)childNode.getUserObject();
            if(co.getName().equals(obj.getName())) {
              exists = true;
            }
        }

        if(exists) {
            childNode.setUserObject(obj);
            DefaultMutableTreeNode n = (DefaultMutableTreeNode)
              tree.getLastSelectedPathComponent();

            if(n != null) {
```

```
            if(n == childNode)
                infoPanel.displayInfo(n);
          }

          tree.scrollPathToVisible(new
TreePath(childNode.getPath()));
      } else {
          DefaultMutableTreeNode newChild = new
DefaultMutableTreeNode(obj);
          treeModel.insertNodeInto(newChild, parent,
parent.getChildCount());
          tree.scrollPathToVisible(new TreePath(newChild.getPath()));
      }
    }

    public void remove(ContextObject obj) {
    }

    public void print(String message) {
    }


    private DefaultMutableTreeNode findParent(ContextObject obj,
                                    DefaultMutableTreeNode start) {

      // every context object belongs to a profile
      if(obj.getProfile() == null) // ERROR
          return null;

      Enumeration e = start.children();
      while(e.hasMoreElements()) {
          DefaultMutableTreeNode child =
            (DefaultMutableTreeNode)e.nextElement();
          ContextObject co = (ContextObject)child.getUserObject();
          if(co.getProfile().equals(obj.getProfile())) {
            return findComponent(obj, child);
          }
      }

      // New profile/session
      return root;
    }

    private DefaultMutableTreeNode findComponent(ContextObject obj,
                                      DefaultMutableTreeNode start){

      if(obj.getComponent() == null) // ERROR
          return null;

      Enumeration e = start.children();
      while(e.hasMoreElements()) {
          DefaultMutableTreeNode child =
            (DefaultMutableTreeNode)e.nextElement();
          ContextObject co = (ContextObject)child.getUserObject();
          if(co.getComponent().equals(obj.getComponent())) {
            return child;
          }
      }

      // New component
      ContextObject ctxt = new ContextObject(obj.getProfile(),
```

```
                                        obj.getComponent());
      DefaultMutableTreeNode component = new
DefaultMutableTreeNode(ctxt);
      treeModel.insertNodeInto(component, start,
start.getChildCount());
      return component;
    }

    /**
     * We implement ContextListener
     */
    public void contextChanged(ContextEvent event) {
      System.out.println("Received event: " + event);
      add((ContextObject)event.getData());
    }

    public void contextCreated(ContextEvent event) {
      System.out.println("Received event: " + event);
      add(new ContextObject(event.getSessionId()));
    }

    public void contextDeleted(ContextEvent event) {
      System.out.println("Received event: " + event);
      remove(new ContextObject(event.getSessionId()));
    }

    public void contextClassChanged(ContextEvent event) {
      System.out.println("Received event: " + event);
      print(event.getSessionId() + ": class changed to: " +
              (String)event.getData());
    }

    public void contextCorrelationOccurred(ContextEvent event) {
      System.out.println("Received event: " + event);
      Correlation c = (Correlation)event.getData();
      print(event.getSessionId() + ": correlation occured: " +
              c.getPattern().getId());
    }

    /**
     * TreeSelectionListener methods
     */

    public void valueChanged(TreeSelectionEvent e) {
      DefaultMutableTreeNode n = (DefaultMutableTreeNode)
          tree.getLastSelectedPathComponent();

      if(n == null) {
          return;
      }

      infoPanel.displayInfo(n);
    }

    public Dimension getMinimumSize() {
      return new Dimension(WIDTH, HEIGHT);
    }
}
```