Dynamically Managing Sensed Context Data

UNIVERSITY
of
GLASGOW

Martin A. Ritchie

Presented for the degree of
Master of Science by Research
Department of Computing Science
The University of Glasgow
September 2004

This thesis is my own composition except where indicated in the text.

30 September 2004

Abstract

This thesis reports on an investigation of the feasibility and usefulness of incorporating dynamic management facilities for managing sensed context data in a distributed context-aware mobile application. The investigation focuses on reducing the work required to integrate new sensed context streams in an existing context aware architecture.

Current architectures require integration work for new streams and new contexts that are encountered. This means of operation is acceptable for current fixed architectures. However, as systems become more mobile the number of discoverable streams increases. Without the ability to discover and use these new streams the functionality of any given device will be limited to the streams that it knows how to decode.

The integration of new streams requires that the sensed context data be understood by the current application. If the new source provides data of a type that an application currently requires then the new source should be connected to the application without any prior knowledge of the new source. If the type is similar and can be converted then this stream too should be appropriated by the application.

Such applications are based on portable devices (phones, PDAs) for semi-autonomous services that use data from sensors connected to the devices, plus data exchanged with other such devices and remote servers. Such applications must handle input from a variety of sensors, refining the data locally and managing its communication from the device in volatile and unpredictable network conditions. The choice to focus on locally connected sensory input allows for the introduction of privacy and access controls. This local control can determine how the information is communicated to others.

This investigation focuses on the evaluation of three approaches to sensor data management.

The first system is characterised by its static management based on the pre-pended metadata. This was the reference system. Developed for a mobile system, the data was processed based on the attached metadata. The code that performed the processing was static.

The second system was developed to move away from the static processing and introduce a greater freedom of handling for the data stream, this resulted in a heavy weight approach. The approach focused on pushing the processing of the data into a number of networked nodes rather than the monolithic design of the previous system. By creating a separate communication channel for the metadata it is possible to be more flexible with the amount and type of data transmitted.

The final system pulled the benefits of the other systems together. By providing a small management class that would load a separate handler based on the incoming data, Dynamism was maximised whilst maintaining ease of code understanding.

The three systems were then compared to highlight their ability to dynamically manage new sensed context. The evaluation took two approaches, the first is a quantitative analysis of the code to understand the complexity of the relative three systems. This was

done by evaluating what changes to the system were involved for the new context. The second approach takes a qualitative view of the work required by the software engineer to reconfigure the systems to provide support for a new data stream.

The evaluation highlights the various scenarios in which the three systems are most suited. There is always a trade-off in the development of a system. The three approaches highlight this fact. The creation of a statically bound system can be quick to develop but may need to be completely re-written if the requirements move too far. Alternatively a highly dynamic system may be able to cope with new requirements but the developer time to create such a system may be greater than the creation of several simpler systems.

Acknowledgements

There are a number of people that I would like to thank for all their help for all their time and patience.

- My supervisors Phil Gray and Peter Dickman for helping me to maintain my focus.
- My family for keeping me going through the hard times.
- The people of the Kelvin Institute for being so generous with my time.
- The masters of LaTeX for guiding me through the nuances: Huw Evans and Michael Dales.
- Ian McColl and the DCS Equator team for their assistance and support.
- My great friends Alasdair Agnew, Sharon Moore, Jonathan Paisley, Craig Robertson, Cristina Sacco and my sister Lois for their proof-reading, motivational nagging and beer buying.

Finally, a big thank you to all the rest of you that listened to me go on about this work, its finally finished.

to my wonderful parents

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

There is a class of systems now being developed that exploit the possibility of providing data through several different media. These systems aim to collect sensory data via mobile devices that are then shared in a distributed system. One example of such a system is the Co-Visiting system of Equator's City Project which gathers a mobile user's position and relays this to other co-visitors. This project will be used as the basis of the developmental work carried out within this dissertation.

There are a number of challenges that the development of such a system poses. These can be broken down into three categories, Collection, Management and Delivery. These are arbitrary categories to divide the data flow from sensors to client. This dissertation will focus on the examination of Management strategies. In particular it will investigate the various approaches to transporting the data from collection point to delivery mechanism.

1.2 Aims

The architectures that currently provide streams of context aware data do so without an understanding of the data. By adding context awareness to these architectures they can alter their behaviour based on the data transmitted. These new "context aware" architectures have the ability to provide an improved data management system as they have the ability to examine the data that is being transported.

This work is an investigation into the feasibility of integrating dynamic management facilities into a context aware architecture without requiring an end-client (person or software) to have explicit knowledge of this management. This will enable the overall system to be more resilient to changing components and more self-managing.

Resilience to change is important in this form of context aware architecture as the hardware components are mobile and are capable of interacting with a number of devices that the original architecture was not designed to support. This unexpected device interaction should still provide some functionality. However, resilience has a cost; the flexibility that is required in the architecture must be balanced against a number of aspects, such as maintainability and complexity, these will be discussed in chapter 5.

There are several factors that influence the flexibility of a given architecture. Lassing et. al [LRvV99] state that making 'right' decisions, whatever they may be, for a software architecture will not guarantee flexibility; other factors such as the difficulty of making a change to the architecture greatly influence the overall flexibility of an architecture. This dissertation shall take flexibility to be 'the difficulty of making a change' as defined by[1]:

The architecture can be said to be flexible if and only if it can be changed to accommodate the following:

- New contexts of use
- New environments
- New requirements

The range of changes that can be performed to incorporate these features is only possible if the cost of making the changes is acceptable. What passes for acceptable naturally depends on the bounds of the operating environment. A change that is possible but requires a large amount of processing would not be acceptable on a handheld device with limited processing capabilities. Without a notion for 'acceptable change' then the flexibility of an architecture is redundant. The most inflexible architecture could be deemed to be flexible given the above definition provided sufficient time and effort is put in to changing the architecture. As the changes may result in a complete rewrite of the architecture it would be unreasonable to claim that the original architecture was flexible. All that can be said for the architecture was that the changes required to be made were 'acceptable'. This trade off between 'flexibility' and 'acceptability' will be explored through several candidate architectures for a real application in chapter 4.

1.3 Thesis Statement

It is my assertion that the choice of architecture for a mobile context aware system will affect the designers ability to create a flexible, context aware system such that interoperability, data management and content delivery can be maximised.

The comparison of architectures that has taken place during this process, will allow subsequent context aware system designers to have a clearer understanding of the trade-offs that are inherent in their architecture choices. This dissertation will allow them to make a more informed decision about the trade-offs inherent with their choice of architecture in terms of its resilience to change and its flexibility in adapting to changing environments.

1.4 Approach

This section shall detail the approach that was taken to establish the above thesis. First, the context that this work was carried out in. The architecture that was chosen to augment with context awareness was the architecture being developed within Equator's City

[1] This style of boxing to represent definitions through out the document.

project group. The architecture, that is the focus of this dissertation, was developed to support the delivery of data from a mobile collection device to the delivery mechanism. The management architecture was developed with just enough functionality, this allowed for rapid prototyping. However, as the project expanded the lack of flexibility required rewrites of several components. Two new architectures were developed in response to the lack of flexibility, this is highlighted in section 4. Each architecture was developed with a different approach to flexibility. These two architectures were then compared with the first architecture to highlight the advantages that the changes afforded the developers.

1.5 Dissertation Outline

Chapter 2 introduces the Equator IRC, focusing on the City Project, its goals and the two systems that were developed for the project. After the description of each of these systems, a set of issues are introduced that characterise the engineering challenges presented by these systems.

Taking these systems as a starting point for the research into sensed context data management systems, chapter 3 describes other systems that also provide sensed context data explaining how they handle the data and the relative flexibility and dynamism that they give their developers. A comparison of the systems is presented in chapter 4 which highlights the varying architectural features that systems of this type exhibit.

These architectural features were used in the case studies on the two new architectures developed to address the lack of flexibility in Equator's City project.

The evaluation in chapter 5 takes a software metric approach to assessing the relative complexity and flexibility of the three systems. However, as the metric evaluation shows, it is difficult to choose a particular system as the best architectural design. As a result, section 5.4 presents an informal evaluation of the Programming Effort required for each of the three systems, showing that the architectures each have a suitable use. Finally, the types of system identified are revisited with attention to the situations that each would be most suited to project development based on the relative flexibility that they offer.

In the conclusion the limitations and future work areas are discussed along with the achievements of the work.

Chapter 2

Equator and The City Project

This section introduces the Equator Interdisciplinary Research Collaboration(IRC)[1] and outlines the Equator-based Project within which this dissertation is set. The Equator group consists of eight institutions from a variety of disciplines that combine to tackle a series of research challenges through experience projects. The focus of Equator is to investigate the divide between physical and digital experiences in the attempt to make crossing the boundaries seemless. Three key research challenges focus the work within Equator; 'Understanding Interaction' looks at how the interweaving of physical and digital should take place. The 'Devices Challenge' focuses on what sort of device would be required to facilitate the required interactions. The final challenge is Infrastructure, this focuses on developing software that will support the devices. These challenges have defined goals for the Equator IRC, developing and generalising the knowledge emerging from each of the projects. New projects in turn can leverage the knowledge gained from other projects to reduce their own development time.

One of the projects is "City"[2], which is the starting point for this work. The focus of the project is integrating the digital information about a city with the physical experience. The project draws on the Equip dataspace (see section 2.2.3) that was developed to address the Adaptive Infrastructure challenge and uses the dataspace as a means if communication with various client applications.

The initial development work carried out for this dissertation was in collaboration with this group. The investigation focused on the software architecture that was used to transfer the data from a handheld device to a central data store. The project is described in detail in the next section and the initial system that was developed is explained in section 4.7.

This chapter shall explain in some detail the systems that were developed within the City Project.

[1] http://www.equator.ac.uk/

[2] http://machen.mrl.nott.ac.uk/Projects/City/

2.1 City Project

The City Project is a collaboration between five of the Equator Partners. Their goal is to allow visitors of a museum or city to use several different media at the same time.

The project's current system operates in a museum setting. This system was designed to work in the museum space at the Mackintosh Interpretation Centre(MIC), in the Lighthouse[3], Glasgow. The system enables three people to visit the centre synchronously: one augmented visitor, one web visitor and one Virtual Reality (VR) visitor. This Co-Visiting System is described fully below.

This mobile system is still in development and will introduce a more asynchronous nature to the system. In its move from the indoors of the MIC to the outdoors of the city it will increase the range of technologies that can be used and will allow individuals to visit the entire city on their own or synchronously with friends. This system shall be explained in section 4.8.3

The aim of the project is to better understand the user's interaction between the city and the information that can be presented about their current context. In addition, by allowing the user to contribute to the information base, the users can share their experience with others.

2.2 Co-Visiting System

The Co-Visiting system [MMRS02] was developed to allow three users to share the experience of visiting a museum. The visitors were able to talk to each other through the system which also provided a shared map and information about the exhibition.

Only one visitor is actually present within the museum during a visit. This augmented user used a handheld device to display the map. The other two visitors can be located anywhere with a network connection, although during the trials they were located in the next room.

The web visitor has the same map that the augmented visitor has and they use a web browser to view the exhibition.

The final user is a virtual reality (VR) visitor, they use the same web browser to view the exhibition information as the web visitor however, their map is replaced with a virtual reality model of the exhibition space.

The next sections shall explain how the visitors software interacts with the system with respect to the classification of visitor. The augmented visitor is physically present in the exhibition while the other two visitors are remotely connected to the exhibition.

2.2.1 System Interaction

Physical Visitor To display the map the augmented visitor is given a handheld device. This device is connected to an ultrasonic receiver so that it can triangulate its position in the exhibition space. To support this, there are ultrasonic beacons or "pingers" placed

[3] http://www.thelighthouse.co.uk/

throughout the exhibition space. The on board software uses the time it takes between each ultrasonic "chirp" to calculate the devices position. The use of a wireless microphone and headphones completes the augmented audio link to the rest of the users. The handheld is used solely to display a map showing the position of the other users. As shown in figure 2.1 the augmented user is the green icon with the white side showing the direction of facing. The other two users are shown in blue and red representing the VR and web user respectively.

Figure 2.1: Screenshot of the augmented visitor's map.

Remote Visitors The other two visitors are remote, communicating using a computer with a microphone and headphone. The web user has a single web page that has an index on the left hand side see figure 2.2, below the index is a map like the one shown on the augmented handheld.

This map differs as it is 'clickable', thus providing the web user's means of motion. By clicking on the map the user changes position. This change of position causes new information to be displayed in the remainder of the web page. The web pages are created using a link base software from Southampton called Auld Leaky[DTMR01], that creates a web page of URLs based on their classification. The information about items in the museum space are classified and when a particular category or area is displayed then other similar items are displayed as links. The final user has a virtual reality interface for moving around the space. The information about the exhibits is displayed in a web browser, similar to the display for the web user except no map is present. As shown in

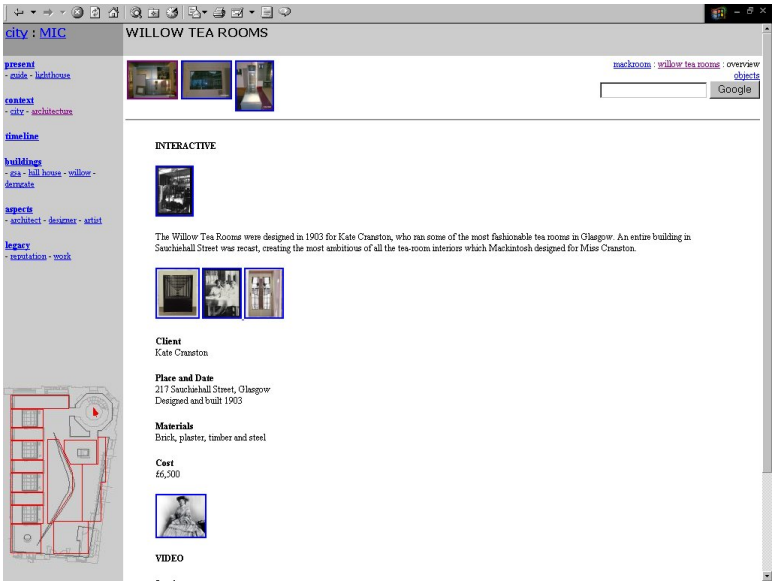


Figure 2.2: Screenshot of the web browser, showing map in lower left.

figure 2.3 the other visitors are represented by avatars. The figure shows the web visitor with the hat on while the female avatar is representing the augmented user.

Figure 2.3: Screenshot of the virtual reality model.

2.2.2 Systems

The system is built around a central dataspace (described in more detail below). A collection of processes uses this dataspace as a means of communication to provide delivery of context-sensitive information. These processes include the collection system that retrieves

user context data and processes it to generate application-relevant information such as the user's context. The user's context is based on sensed values such as their location and orientation, this can be used to understand what is being looked at in the Mackintosh Interpretation Centre. The context information is then shared through the dataspace and can be used by other processes to provide information about the surrounding exhibits as shown in figure 2.4.

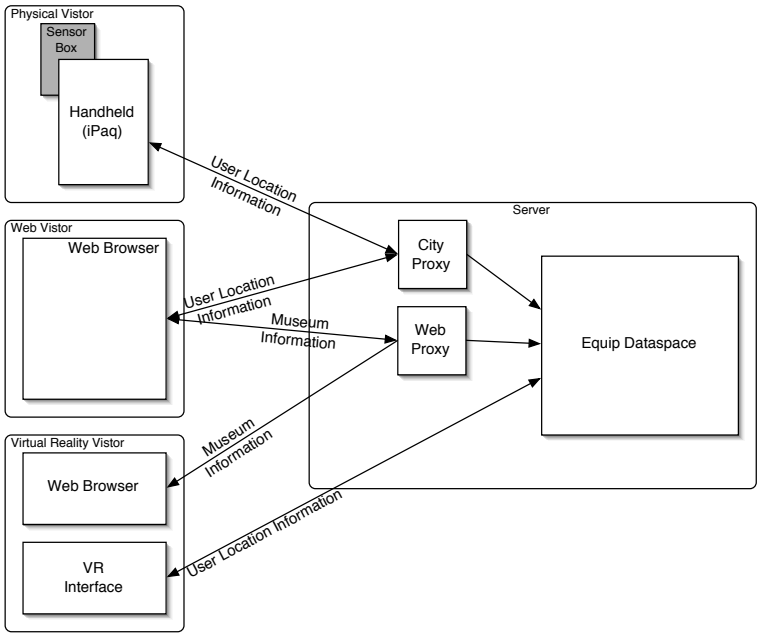


Figure 2.4: Architecture overview of Co-Visiting (OSC) System.

This system is unusual, as all of the information about the user is gathered by the user. What follows is a description of how this information is gathered and then used by the main components on the server.

Collection It is easiest to understand the collection process by describing the components through which the data is processed. Figure 2.5 shows how these components interact. The Sensory Source is the "closed box" of the handheld or other component that has gathered the raw sensory information. The term "closed box" is used, since the functionality is hidden from the collection process. All that is seen is the stream of data. In this case, the box represents the ultrasonic location system. The handheld initiates communication with the system through the 'City Proxy'[4] service that is listening on a known port. This, in turn, creates a handler for the connection. The `MessageHandler`[5] is responsible for receiving and processing the messages from the source. Finally, the relevant tuple is retrieved from the Equip dataspace and updated.

[4]It should be said that the City Proxy is not actually a proxy. It is an interface to the Equip dataspace that handles the translation on behalf of the handheld code.

[5]Again the naming within the project is misleading. The use of the name `MessageHandler` suggests that there are other types of handler. This is not the case. The `CityProxy` simply delegates the responsibility of message translation to this handler.

Data Transmission Initially, the client must make a connection to the server. The connection process requires that the client provide a user name to be associated with the incoming data. The connection to the `CityProxy`, in figure 2.5, results in the creation of a processing object: the `MessageHandler`. Information about the Ultrasonic system's accuracy is loaded from a configuration file during the creation of the `MessageHandler`. This includes the orientation of the axis and a transformation matrix to convert the measurement units into latitude and longitude. This is used in the processing of the ultrasonic messages.

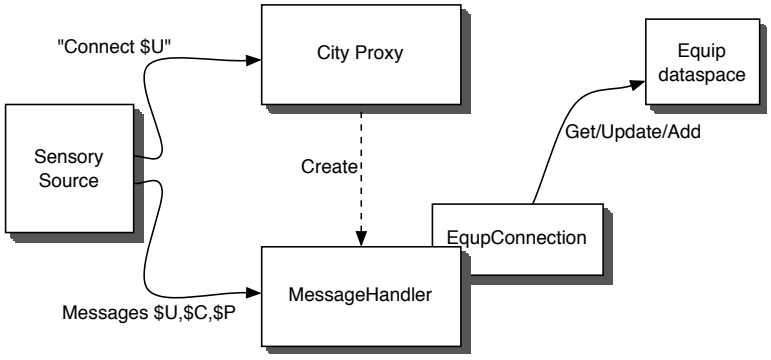


Figure 2.5: Sensor data flow from sensory source to dataspace.

Processing The messages are received and dealt with in the main loop of the `MessageHandler` object. This is done by a switched loop with predefined options. For example, when a `$U` message is received it is switched to the `handleUser` method.

These processing methods perform all the transformations that are required before the Equip dataspace is updated with the new values. This is important to ensure that consistency in the data representation is maintained. The collection of components described above shall be refered to as the Original Statically Compiled (OSC) system to differentiate it from the other modifications presented in sections 4.7 & 4.8.

2.2.3 Technologies

This section describes the technologies involved in creating the Co-Visiting system. These technologies deal with storage, location, transmission and context sensitivity. These are: the Equip dataspace, that is the communication mechanism between the visitors, the proxy server that handles the input from the handheld and web user, and the VR interface.

Tuple Spaces A Tuple Space consists of a number of tuples, that are, ordered sets of typed values much like tuples in a relational database. A tuple space is beneficial as the data lives outside of the processes that may need to access to the data. This enables data sharing without each process having to be aware of each other's existence. The client software has several operations that it can perform on the space. Operations are always performed from the point of view of the client application. The standard operations are

`in` and `out`. That is, the client application can perform an `in` operation if it wishes to move a value from the space to work on it. Similarly, the client performs an `out` operation on tuple values that it has finished working with and wishes to return to the space.

Tuple spaces are designed to maintain the integrity of the data they store. To accomplish this there is only one copy of each of the tuples so if the client application wishes to edit the value it must first remove it from the space with an `in` operation. Tuple spaces use matching to select tuples for the requested operations. The operations that return a tuple must first supply an `anti-tuple`. This is a special kind of tuple that contains the types for the desired tuple. The anti-tuple may also contain exact values to match upon. The anti-tuples are placed in the space as with normal tuples and if a match occurs then the client receives a call-back with the resulting tuple(s). If there are no matching tuples then the anti-tuple will remain in the space. When, at some stage, a matching tuple is inserted, the client will be notified of the new insertion. The anti-tuple is not removed when a match is found; rather it remains with in the system and acts as a form of subscription method.

Equip Dataspace The Equip dataspace [Gre02] is the tuple space used within the City project. The system is designed to support distributed interactive systems, and as such, supports the late joining of clients to the shared space. When a client joins the space they are able to find the current state of the other clients with a simple query. This is the style of interaction that a dataspace provides, however, it is not always important to support late joining. An alternative method for distributed systems is to use a message passing system such as Elvin [Seg93].

Elvin delivers messages based on the content of the message, that means that more than one process can receive the message. The benefit of using a message passing system is that there is no state. This means that there can be no stale information as all messages are sent to the connected processes when they are received. This allows users to know that the data they are receiving is current and to know that the data they send is not going to be made available for an extended period of time. If a client is not connected when a message is sent, then it will never see it.

The decision to use a state based model in Equip was to allow 'late joining users' to be able to see the current state of the users. However, in addition to the state-based reporting method, Equip also allows an event-based reporting mechanism. This allows clients to send their updates as events to other clients that are listening for events[6].

Sensor data values are stored in the Equip dataspace by creating a subclass of a generic sensor item as shown in figure 2.6. These items can have values left blank, this results in an anti-tuple allowing them to be used as search items within the dataspace.

For example inserting the `GPS Item` shown in figure 2.7 with only the value of a `username` will result in the insertion of an item ant-tuple. This will result in a list of items being created from the items that are currently stored in the equip dataspace with that username.

[6]The event mechanism is not being used in the OSC system, but is being used in the new Mobile system described in section 4.8.

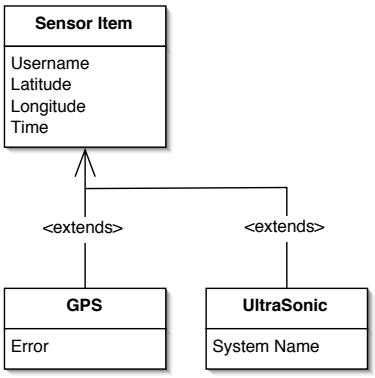


Figure 2.6: Inheritance of sensor data items from a generic storable data item.

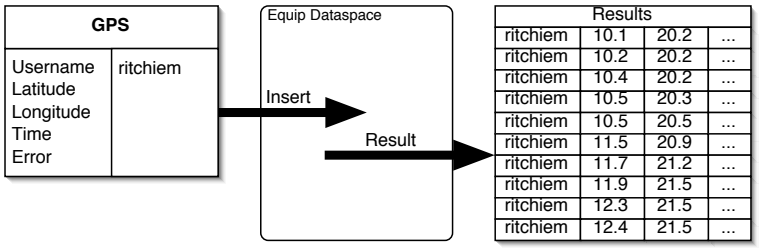


Figure 2.7: Example of a sensor anti-tuple being used to query for a user's location.

The inserted anti-tuple remains and any further item insertions that match will be sent back to the process.

Events are generated in a similar method to items. Initially a subclass of `ReportEvent` is created and a data item, such as a `GPS Item` is added. In addition a metadata object that describes various aspects of the event such as time to live must be added. When this object is inserted all processes that have requested events of the given `ReportEvent` subclass are sent a copy of the new object. The event will stay in the dataspace for as long as is requested in the metadata. This allows for late joining processes to see the event.

Trigger Service Attached to the Equip dataspace is a 'trigger service' that watches for collisions of user positions and defined boxes within the room. This causes information to be pushed to the client's browser from a linkbase containing information about the exhibits in the room. The use of information within the system is very open. Each component attached to the dataspace is allowed full access to the information. This allows the main components to provide the clients with information that may be of use to them

Ultrasonic Location System The Ultrasonic Location System from Bristol [RM01] consists of a black box containing the ultrasonic receiver and a compass to give the 2D orientation of the box. The receiver reports the time between the first chirp and the subsequent transponders that are located around the room. The software on the hand-held can then calculate its position based on the geometry and the known transpon-

der locations. This position, along with the compass orientation, is then sent using an ASCII protocol to the server. The protocol is a simple typed string of the format `$<Type(Character)><Message>;`. The type can currently only be a single character but it is feasible that further types can be defined by adding a second character. This would naturally break the current applications if a currently used initial character was used.

The current protocol consists of the following values:

- $C heading
- $G GPS position
- $L location
- $P (ultrasonic) position
- $U timestamped user and device identification
- $R refresh
- $S stop
- $V visitor

The main dataspace has the y-axis pointing upward. The Ultrasonic system, however, has the z-axis pointing upward. This requires a rotation to be performed on the Ultrasonic readings before the update is performed. This rotation and translation of the data values must be encoded in the description of the location system. If the coordinate system that is being used is not understood then the dataspace cannot maintain a consistent location record.

City Proxy Server The City Proxy server is the point at which the map, as shown in figure 2.8, interface with the rest of the system. As mentioned earlier the City Proxy is not exactly a proxy. The naming of the component is historical and not important although for simplicity it shall be referred to as the proxy in this dissertation. What is important is that the component handles the communication between the external components and the Equip dataspace.

The proxy handles the translation of the ASCII protocol into Equip objects. The majority of the protocol is dedicated to the handheld set up. The 'black box' connects to a piece of software on the handheld which in turn generates an ASCII protocol across a socket connection to the server. By including `Type` information along with the data the proxy server is able to translate the ASCII string into data objects in the Equip dataspace.

The proxy server is very straightforward in operation. The main `MessageHandler` class receives the ASCII protocol strings and uses the type to operate a switch statement to select the handling method. The method then performs the decoding of the string and utilises other methods for updating the various position values so the user moves on the map and within the VR environments.

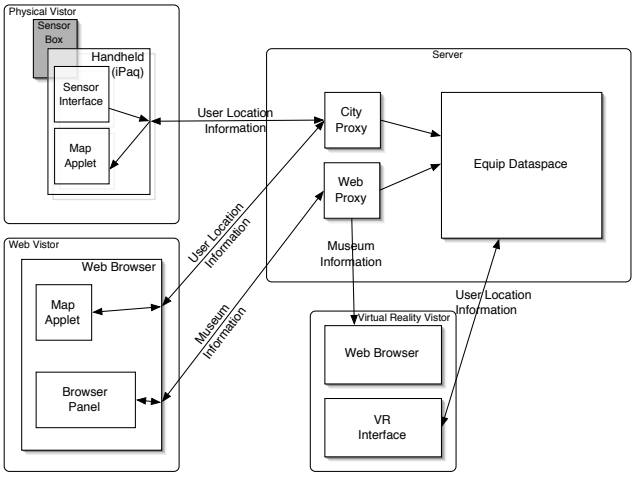


Figure 2.8: Co-Visiting Client connections to the server, through the various proxies.

Web and Augmented Interface The web and augmented visitors both use the City Proxy to relay the information from the map applet (see figure 2.8). In addition to sending their own position updates the clients can request a list of other users that are using the system so that they may be displayed on the map.

Virtual Reality Interface The final component to the system is the Virtual Reality (VR) interface. A number of renderers have been used, the first of which was VR Juggler [CNBH+02], but now the more transportable OpenGL renderer is used. Unlike the web and augmented interfaces, the renderer connects directly to the dataspace from which it has access to the model and other avatar's positions, as shown in figure 2.8. As the visitor moves their avatar around the virtual exhibition space new information is presented on a web page about their position. The information that is presented is the same as the web visitor would receive if they were at the same position. This is important as the web information is the only information about the exhibition that the Web and VR Visitors see. If the displayed information was different then it would be difficult for the two visitors to collaborate.

The Co-Visiting system may be the focus for development work in this dissertation, however, it is not the only possible approach for developing a context aware architecture. The next chapter will introduce other systems in the area that have similar uses to gain a better understanding of the alternative approaches.

Chapter 3

Background

Mixed reality systems are just one class of system developed on a context delivery architecture. This chapter explores the area of delivering context data, beginning with an introduction to the notion of context and what that means in terms of the applications that use context-related data. This view of context will then be used in discussing the current context-aware software architectures. Each of these architectures takes a different approach to data within the architecture and how the client applications should be allowed to access it. The approach taken can affect the functionality of the client in a given environment, particularly with respect to flexibility. If the architecture is highly coupled to the data type then changing the data type will be difficult. Section 3.3.1 discusses how the data is used within an application and here we see how dynamism can improve the end-user experience.

3.1 Sensed Context

This section presents a number of views of 'context'; what it is and how the addition of context to client applications can augment their functionality. The definition used throughout this document is that of 'Sensed Context' [GS01].

---

sensed context

Properties that characterise a phenomenon, are sensed, and that are potentially relevant to the tasks supported by an application and/or the means by which those tasks are performed.

Context Data Categorisation Sensed context only helps define the data sources that can be dynamically managed. The category of data that will be processed is also important. Abowd and Mynatt [AM00] divide 'context' into five categories: Who, What, When, Where and Why. This allows a developer to select data sources that will provide information in one of the five categories.

Of Abowd and Mynatt's five types of contextual information, Who and Where are the simplest to retrieve. These two categories are therefore the current focus of context

sensitive applications. Current applications often ignore information in the When category, though this information can be very important. Timing is inherent in event based systems, and knowing that something happened at a given location, but not when, is not always effective when trying to gauge an appropriate response. For example, a message is sent notifying the reader that a light bulb has broken. The message only says that sometime before that moment the message was received, the light bulb broke. The reader of the message does not know how long ago the bulb was broken and therefore doesn't know if it is likely the problem has already been solved.

Without knowing when an event was sensed, all sense of history is lost. The resulting 'trail' of data that the 'When' data provides can be viewed as a history of activity. From this, comparisons can be drawn to other users to provide an alternative source of data that might provide insights into the current user's activities.

By categorising the data a system should be better equipped to understand the data formatting that is sent by a sensor. For example, by knowing that the data is 'When' data the system may simply be able to try converting the input string to a numeric value that represents date and time. For the relatively constrained categories such as When and Who, processors can be developed to extract the information from new sensors without any prior knowledge.

Data States Chen and Kotz [CK00] use the terms *active* and *passive* to describe two states of context. Active context influences the behaviour of an application whereas passive context is relevant but not critical to the operation of the application. The two terms are akin to dynamic and static data values. Active values are dynamic in nature, reflecting the current context of the user. On the other hand, dynamic data are those parts of context that the application is more focused on, as they bring about a change, which may be of interest to the user or the application. Passive values are more static as they reflect a state of the user that does not change as rapidly, such as their age. Thinking of the data available to each of the systems in these terms highlights the data that the architecture may take advantage of to alter its behaviour. For example, it is fairly easy for the architecture to discover which users are accessing the system. However, if the architecture could identify an increase in demand for given data, more resources could be allocated to acquiring that data or sourcing the data from a more cost effective source.

'Who' is using an application at any given time can be an important piece of contextual information. However, this information is usually static for normal non-context-aware applications. Many text editors for example, do not know who or need to know who is using them. The behaviour of the application remains static. If the application had the ability to sense its user then it would be able to provide the user with more tailored operations. Microsoft Word, for example, enables multiple editors of a document by using the current user name and adjusting the colour of the edited text to suit. Emacs can highlight the text of a document based on the file type, this level of customisation is dynamic but based on 'What' not 'Who' is performing the editing. The problem is being able to provide the required sensed information to allow the application to react

dynamically to the changing user.

The design of the architecture will be impacted by the types of data category required as some data can be gathered locally while other data will require network access. The type of data will also have an effect; data that is passive would work in a simple publish and subscribe architecture where time critical delivery is not a requirement. However, the delivery of active values requires that the architecture be suitably scalable so that the sources that provide information to the active value can be located and connected to the client.

3.2 Context-Aware Software Architectures

Context-aware applications are not usually self-contained products like Word or Emacs. They usually rely on a series of processes that gather and process data from a number of sensors. It is this structural support that is referred to as a context-aware architecture. The term infrastructure, therefore, refers to the hardware components used by the architecture.

---

context-aware infrastructure

The collection of components that comprise a context-aware system.

---

context-aware architecture

The method by which the infrastructure is interconnected.

The current trend in context-sensitive architecture design is in the development of mechanisms that can gather richer sources of context, benefiting applications. The context information can be gathered from any number of sensing technologies, such as physical sensors in the surrounding environment or contextual widgets that gather information, such as mouse movement, about the user's current actions[SBG99]. The current architectures gather this information and present it to the applications, as requested, yet the architectures themselves tend to operate independently of context. The architectures simply provide access to the context information for the applications without evaluating their own behaviour. This limits their functionality to simple message passing systems.

By providing the architecture with the ability to adjust its behaviour dynamically, based on the data that it is transmitting, it can provide a better level of service. For example, knowing that there are no consumers for a particular data stream could allow sensors to be switched off, or if demand were high for a particular dataset then more resources could be allocated to handle the processing.

When considering context-sensitivity in conjunction with mobile computing, not only is there a large amount of context that can possibly be sensed, but also any architecture must be aware of the underlying infrastructure. The architecture needs to be aware of the communication infrastructure so that it may adjust its behaviour to suit its context. If a network connection is only possible with a mobile phone then the cost of sending/receiving data from the phone must be weighed against the importance of the communication. This weighing is currently performed by the user actively connecting their phone and PDA.

The prevalence of wireless connections could result in the PDA being able to connect and charge the owner for the data transfers so any architecture will have to realise there is a cost in sending data from the device. The infrastructure will involve a variety of location and communications technologies having radically different properties which will need to be integrated. An example of such an integration issue is the use of ultrasonic location technologies being augmented by information from the RPM of a motor. It may be that the ultrasonic is inaccurate and information from the motor can improve the accuracy but only if the two measurement systems can be integrated. Taking last known good ultrasonic position and using the speed of the motor in combination with a direction sensor could allow better accuracy in areas where ultrasonics alone were inaccurate. While a number of current architectures handle these problems to some degree, the ability of the architecture to adapt to the current context becomes more important as the combination of sensing technologies changes and the possible combinations become exponential. By providing a dynamic architecture, systems would be more able to react to the presence of an unknown data source or receiver.

When considering context-sensitivity in conjunction with mobile computing, the amount of context that could potentially be sensed increases exponentially. The architecture needs to be aware of the communication infrastructure so that it may adjust its behaviour to suit its context at any given moment.

Current context-sensitive systems focus primarily on location data and delivery of sensed data. Location is one of the easiest contexts to work with. For example, there are various systems that can be used to generate location data about a user. These systems take two forms. First, there are systems that require an existing wired network, such as the Olivetti Active Badge system [WHFG92]. The user can maintain control of their position data by hiding the badge when they do not wish it to be known. However, after the badge reports the location, the user is unable to control access to the data. The second type of system requires the user to have a more sentient device. Radio frequency pingers or other location-sending devices must be installed in the areas where location data is to be gathered, such as those from Olivetti and Oracle [WJH97]. The user's device determines its own location so the user's device is the only technology that knows this information, as such the user can decide when to publish their location data for others to use. While this model gives the user more control over their data, their device must be more flexible to cope with the number of sensing technologies it will encounter. A user's mobile device will encounter more technologies, as compared with a desktop system, as it is more mobile and so likely to be taken to facilities that have differing technologies.

When the architecture gathers the user's position it may not be in the same format as the rest of the system. However, it has all the resources necessary to make the required translation to provide the position value in the correct form for various applications. This is possible as the locating system will only operate with a limited range of user devices. The same is true from the user's point of view. Their device will only operate with the systems they are able to establish a connection with as translators may not be available for every system they encounter. Making the device more dynamic, however, should allow

it to be better at coping with change. This dynamism can be improved in two ways: adapting the device to understand the new infrastructure, or providing translators within the architecture to modify the data for the device. Section 4.7 describes a component-based architecture in which each component is capable of transforming the typed data stream. These streams can hopefully be transformed sufficiently to provide the required data for the client.

3.2.1 Context-Toolkit

The Context-Toolkit [ADOB97, SDA99, DAS01] was developed from context work on the Cyberdesk[DAPW97]. The Context-Toolkit consists of 'Widgets' that gather and process data. The data can be gathered from a number of different sources, which may be physical hardware sensors or software sensors. These sensors do not have to be local to the widget. Widgets can also use the services of an 'Interpreter' to transform low level data into higher level information, such as transforming a user ID into the actual user's name. The widgets can also be composed to create richer data sources. For example, a *Meeting Widget* would use the outputs from an *Activity* and *IdentityPresence* widget assigned to a particular room to gauge when a meeting was taking place.

A sample application that was created by Daniel Salber [SDA99] was a digital In/Out notice board for a building. By creating a single *IdentityPresence* widget that monitors users entering and leaving the building, the notice board would receive notification when the user entered or left the building. This relatively simple application demonstrated a number of the benefits from using the Context-Toolkit widgets. The large variety of sensing technologies that exist across a distributed space means that creating such an application would be difficult for the programmers without some form of abstraction. The use of the widgets helps to create the context from a collection of disparate sensing technologies and by creating abstractions over the raw sensor data allows application designers to focus on handling the contextual data rather than gathering it. The reusable nature of the widgets allows the application designers to simply select the widget required from the toolbox much like developing a user interface.

Finally, since the distributed nature of the widgets allows them to be easily shared, maintaining control of which applications access them can become difficult. One suggested solution is the use of servers as a gateway between applications and widgets. This not only allows access to various sensors to be mediated by a security server, but, in addition, personal servers can be used to reflect a user's privacy model. Therefore, a modification to the In/Out notice board to create a 'PersonFinder' application would require the application to locate and listen to all of the *IdentityPresence* widgets in the building. By using a building server, however, all of the users that wished their widgets to report their presence could connect to the server for publication.

Currently, the widgets in the system only perform a connection request at startup. These requests must be sent to known widgets, and if they are unavailable then the connection is not completed. A resource discovery mechanism is still to be developed. However, without the inclusion of a 'publish and subscribe' method to remember connec-

tion requests, then the application will have to attempt to form the connection itself. The exclusion of some discovery mechanism leaves the Context-Toolkit as a simple basis upon which a context-aware architecture could be built. The use of an abstraction layer to hide the complexities of the individual sensors from the application developers goes a long way to increasing the flexibility. This abstraction layer means that the developer need not worry about the operation of the lower levels.

In terms of flexibility presented in section 1.2, the Context-Toolkit, even though in early development, is capable of being developed into a suitable context-sensitive architecture. The Context-Toolkit is based on widgets so it can be adapted easily by adding new components. The ease with which new components can be added to the architecture highlights the advantages of a component architecture. The ease of adding new components also allows architecture to accommodate new requirements and new contexts of use. The flexibility of the architecture in dealing with a new environment depends on the target environment because the widgets do not require any complex co-ordinating process nor do they require external connectivity. This would allow a simple context-aware system to be deployed on a handheld device. The downside for such a simple device is that it is unable to communicate with other context widgets to connect to a richer set of sensors. The various widgets must be started in the correct order to ensure the connections are made. Should connections be broken or a new external connection become available, the current Context-Toolkit does not have the ability to handle the reconnection.

This reliance on reliable communication works fine for the desktop-oriented systems presented in the various Context-Toolkit papers listed at the start of the section. To develop the Context-Toolkit into a mobile context-aware system work must be done to ensure that communication over the wireless network does not break the functionality of the widgets. While the widgets provide an abstraction method from the individual sensors present on each handheld device, they do not provide an architecture to deliver the data. If the storage mechanism is unavailable at startup, as there is no resource discovery capability presently within the widgets, then the data cannot be delivered. This is a failing that is also noted by the authors of Context-Toolkit along with the inability to scale the system. By using a gateway as a means of addressing the scalability issues, this system provides suitable access control to private user information at the source. This is in contrast to the next system, Solar, that has a robust discovery mechanism whilst access control is only applied just before data delivery.

3.2.2 Solar

The Solar architecture from Dartmouth [CK01a, CK01b, CK00] is modelled on the solar system, a central star surrounded by a number of planets. The Star maintains a Subscription Engine to which applications submit requests for context data streams. These requests are then transformed into a series of operators that can Filter, Transform, Merge and Aggregate the event streams to provide the required information. These operators exist within a separate execution space called a Planet. There can be several Planets in the system, each containing several sources and operators, running on one or more ma-

chines. Each operator runs on its local Planet but an entry describing its abilities is stored in the Operator Space in the Star. This allows the incoming request to the Star to be fulfilled based on the currently available event stream operators, and also allows the reuse of currently deployed operators so duplication of effort is minimised.

As with the Context-Toolkit, the use of operators greatly benefits the system. By adding new operators, the overall power of the system is improved. Where the Solar architecture improves over that of the Context-Toolkit is with the introduction of the Subscription Engine to provide resource discovery [CK03]. This central store co-ordinates the communication between sensors, operators and clients. This allows a far more distributed system to be developed as operator addition and removals can be seen and may not interfere with the operation of the architecture. An additional problem with the Context-Toolkit is each component needs to be referenced directly to enable the communication. This works for the small number of components in any given system, however in a system such as Solar it may prove too complex. If each requester needed to know the type of operator or sensor it requires, then the system would be highly coupled to those instances. Solar avoids this by allowing symbolic names to be attached to single components so requesters need not specify a long chain of components to create their event stream. These symbolic names and names of services such as printers and displays are all stored in the Star and are used to create a *naming tree*. This is a hierarchical tree, containing both static and context-sensitive names. The underlying value providers can change as required, but the applications need not be aware of such changes. For example, the tree may contain static name `/devices/printers/23`, which refers to a specific printer, and name `/labs/005/devices` refers to a dynamic list of devices in lab 005, of which printer 23 may be one of those devices.

The Star is responsible for maintaining these dynamic lists. These dynamic lists can also be used to generate context-sensitive information. A path such as `/users/ritchiem/location` would return the current location of user `ritchiem` which a handheld could use to enable other context-sensitive applications. The In/Out notice board example from the Context-Toolkit could use a path such as `/glasgow/mainoffice/users` to return the list of current users who are in the Main Office.

Now that the Solar project has established a working context-sensitive architecture, issues of Access Control and Privacy are being addressed. In most systems, trying to "bolt-on" security by retrofitting the original system is the wrong way to approach these issues. However, the Solar architecture uses a client component that connects the client to the architecture. This component was modified to include security controls. The approach is to use Access Control Lists (ACLs) attached to each event [MK02]. The use of ACLs was chosen to satisfy a number of objectives: limiting access to events; avoiding central policy authority; allowing user specified context-sensitive access control; and maintaining the transparency of operator sharing. However, it is the ease with which security is added that is of interest to this discussion. The component-based architecture provided sufficient flexibility to allow security, which usually must be designed in from the ground up, to be added later.

Taking Solar in comparison with the Context-Toolkit and the aims presented in section 1.2 it is clear that a component-based approach provides increased flexibility, especially when accommodating new requirements. The distributed nature of the components within the Solar architecture provides scope for expansion. This expansion, however, becomes limited when the issue of disconnection in a mobile environment is introduced. This is due to the reliance on the central Star process to co-ordinate requests and the available components. A system can be set up without the need to have a Star process, however, without this process it is not possible for the various components to perform resource discovery. On a mobile system where the configuration of components is likely to be known at start up, the reliance on discovery is avoided. Unless the Planet on the mobile device is capable of locating a Star on the local network, the functionality of this approach will be limited.

To summarise, the Solar approach provides fixed wire devices with the ability to share resources and provide their clients with sources of context-based data. Mobile users are limited to using only the resources that they can run on their device for time-critical updates. Access to private data is controlled through access control lists, and while this may have been relatively easy to add for fixed wire clients, in the case of mobile users it becomes much harder. When a mobile device is connected to a Solar System then all of a device's resources will be freely accessible to the other operators. This means that data may leave the device against the owner's wishes. While the requesting client's operator will most likely deny access to this data, the fact that it was originally sent from the device using valuable power and bandwidth is of concern.

The Context-Toolkit and Solar both provide complete or nearly-complete architectures for delivering contextual information. In our mobile scenario, however, it is unlikely that we will have the luxury of only encountering a single architecture. Any software that is to run on the mobile device must be flexible enough to operate with any number of new systems and environments it may encounter. This is where the approach of Recombinant Computing, described next, can help.

3.2.3 Recombinant Computing - SpeakEasy

The development of new architectures to handle contextual data is only one step in providing a working solution. There exists a large amount of technology already deployed that could be used in a context-aware manner, selecting a printer based on proximity, for example, or displaying a presentation on a nearby projector. To require the consumer to replace their existing technology with context-aware versions is unreasonable. Instead, a potential solution is recombinant computing: by abstracting a device such as a projector, an interface can be made that allows other components to understand the device. Rather than requiring the physical connection to a projector to send a video signal, the interface can abstract that underlying requirement and ask simply for an image to display.

This work currently being developed at the Palo Alto Research Centre. They are developing the Recombinet[1] software architecture as the interconnecting technology that

[1] http://www.parc.com/research/csl/projects/recombinet/metainterfaces.html

provides the abstractions to allow devices to inter-operate. The abstractions have one or more interfaces: Aggregation/Discovery, Data Transfer, User Interface/Control and Metadata/Context. Each of these interfaces provides generic methods of interaction with a given component in the system. The interfaces can be used to develop contextual components which should allow new components to discover consumers and producers for their inputs and outputs. This form of architecture would allow mobile devices to connect to an available infrastructure quickly and make use of its services.

The Speakeasy[2] project is part of this research at PARC and focuses on the issue of interoperability. A system was developed using the User Interface/Control and Discovery interfaces to support end-user configuration of ubiquitous environments [NSE+02]. This was achieved by developing an interface on a PDA that discovered local devices and services and presented their interfaces. This allows the user to simply control the local projector and send a recent file for presentation. This approach was shown to be beneficial because simply presenting the user with all of the options for all of the discoverable devices proved to be overwhelming for the user. This overload of information caused confusion with the users. A better interaction can be had by minimising the configuration that the user must perform, thus allowing the user to focus on their task. The issue of interface design is not the central idea here, rather, the system highlights the benefit of providing a dynamic metadata based system. Device interoperability is increased as each device provides their interface to the controlling application. The complex job of interpretation is left up to the user who is far more capable.

If the devices in a given area can agree on a protocol for interoperability, then as the user enters a new environment their personal device would be able to discover the available context servers and continue the services that were running. Again, drawing on the aims from section 1.2, the Recombinet software architecture provides a framework from which the implementation of each component can incorporate the dynamism of Solar's Naming Trees. This will encourage its use in new contexts and cater for moving requirements of the system. The benefit of this approach is that devices that were not explicitly designed to operate with one another can find a path of components that allow them to communicate. This new-found communication could lead to networks of components that can allow the user's application to be used in ways outside the original system design.

3.2.4 GLOSS

The previous section on recombinant computing highlights the technology that we are beginning to encounter. The GLOSS[3] project takes that view a step forward to ambient computing. That is, where computing capabilities are embedded into numerous everyday devices and the traditional view of a computer as a desktop or handheld disappears. The aim of the GLOSS project is to allow a user to simply and quickly make use of the technology that surrounds their life.

This vision of ambient technology requires an architecture that can interact with any

[2] http://www2.parc.com/csl/projects/speakeasy/
[3] http://www.gloss.cs.strath.ac.uk/

number of given systems. The first step for creating their context-sensitive architecture is the development of a collection system [RCC02]. The approach is similar to that of Recombinet. Through the use of a series of known interfaces the components in the architecture can communicate and share data. One of the aims of the architecture will therefore be to ensure that it is sufficiently flexible to cope with new environments.

3.2.5 Personal Server

The final architecture covered here is the Personal Server from Intel [WPD+02]. While the Personal Server is not so much an architecture as it is a single device, it is its functionality and interoperability that is of note. The device is slightly larger than the hard drive that is the constituent component. It uses Linux running on a StrongARM processor to co-ordinate the Bluetooth communication with other devices it encounters. The goal of these encounters is information exchange. The device stores all the user's documents and applications and this allows them to use any terminal that can be communicated with via bluetooth. This is one of the GLOSS scenarios, where a user can use a public display—be that a table top in a café or a notice board on the street—to display relevant information about their current context. Currently, the device is focused on allowing the user to run applications and edit documents. However, the device also has context-sensitive capabilities: as the user goes about their daily activities the Personal Server can gather information about the systems and devices it encounters. Merchants could use these capabilities to transfer information to the user's device which may be used later at a more convenient time. For example, the daily specials at the local delicatessen may be stored as the user walks to work and then at lunchtime the device may suggest one of the specials for lunch. A drawback of the system is that Bluetooth discovery and connection takes too long which means using the device will not be suitable for gathering information from a shop by walking past it. It was suggested that in conjunction with another technology the connection time could be reduced. For example, 802.11 wireless networking could be used for data transfer and the availability of a given Bluetooth cell could indicate location. The added range of 802.11 will give the device more time to transfer the data. In order to understand the technological issues surrounding The Personal Server, Intel developed and evaluated the hardware first.

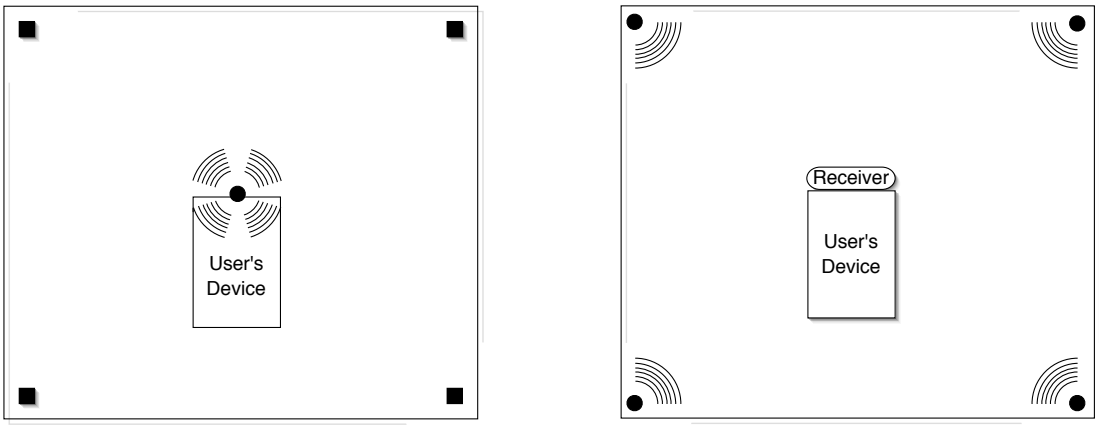
The Personal Server is an example of a device that the previous architectures in this section may be required to support. While the Personal Server does not document how a context sensitive architecture should be made, it does give an idea of how these architectures can be used. By taking all your data with you from place to place, gathering various information on the way, the possibility exists for applications to draw on much richer information bases and use that information to support context-sensitivity. The Personal Server is a development of a software architecture to provide context sensitive information delivery on an individual basis. Distributing simple information nodes around town and providing the individual with a means of gathering that information highlights the difficulties of information delivery in a disconnected environment. The previous architectures have all relied upon a fixed infrastructure to provide data while this approach

from Intel has assumed a disconnected environment which is closer to the current mobile user's situation.

Assuming a fixed or disconnected architecture within the sensed context system will impact the functionality of the final system. The choice of architecture should be based on the types of context data the system is providing to the end user. To understand the types of data that such a context sensitive system may utilise the following section discusses possible approaches to sensor data comprehension.

3.3 Sensing

The sensing of data can occur in a number of ways. There are two distinct classes of architecture for gathering sensed data, location data in particular: either the infrastructure senses the location (figure 3.1a) or else the user's device senses its own location (figure 3.1b).



(a) Environment senses location.

(b) Device senses location.

Figure 3.1: Two different approaches to location sensing.

The choice of which approach to take has trade-offs that will affect what can be done with the software architecture as described above in section 3.2. The Equator projects deal with a lot of sensed data; as such they conceived two frameworks for developing new context-sensitive applications. This section presents both frameworks and their relevance to this work.

3.3.1 Sense and Sensibility

The Sense and Sensibility framework [BSK+03] is designed to help focus the development of a context-sensitive application. By looking at the available information that can be sensed by technology and looking at what information would be sensible to gather a

selection of sensible values that can reasonably be sensed by the current technology is derived. For example, the Augurascope [SKF+01] has a tilt sensor that is used to change the display; however it can only sense in a 90 degree range. If the screen is tilted out of the range then the sensor can no longer provide an accurate value so the behaviour of the sensor is unknown. Knowing that the sensor behaviour becomes erratic allows the developer to design their system around that behaviour. The Augurscope project used this state to pan away from the user's current location and show a top-down plan of the area.

By understanding the range of values that a sensor can output it is possible to map these to responses from the system based on the groupings show in figure 3.2. The use of this development framework is important as it allows the developers to understand the range of outputs that the sensors provide.

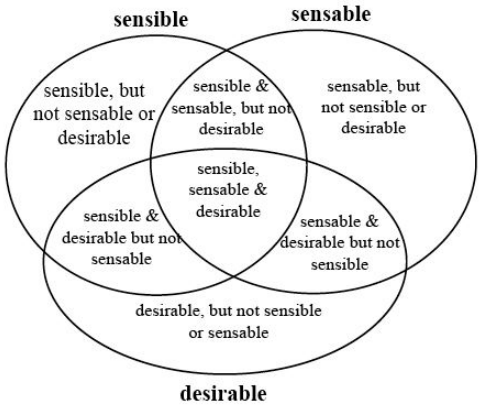


Figure 3.2: Sensible, sense-able and desirable groupings.

Table 3.1 highlights an analysis of the Augurscope's tilt sensor. The 90^o of operation highlights the Sense-able range of values. However, just as a programmer would check the range of values of controlling variable, the table shows the other values that the object the sensor measures could possibly have. Outside of the sensors range there is nothing that can be done so sensible defaults are set. Pans are made to give overviews of the place in view.

Sensor Output	Sense-able	Sensible	Desired Use
$<0^o$			Pan back to a side profile.
0^o-89^o	×	×	The main movement of the display.
$>90^o$		×	Pan up to an overhead view.

Table 3.1: Sensor values analysis for the tilt sensor.

Table 3.2 shows the benefit this approach has for understanding the sensor outputs. If the 90^o tilt sensor was replaced or augmented to sense 180^o a different table would emerge. This time it is important to note that the viewing angle of the display screen is only 160^o. So using any sensor reading out side this range is not sensible as any change in the display at this point would not be visible to the user.

Sensor Output	Sense-able	Sensible	Desired Use
$<0^o$			Not visible.
0^o-159^o	×	×	The visible range of the display.
160^o-179^o	×		Not visible.
$>180^o$			Not visible.

Table 3.2: Analysis of a 180^o tilt sensor.

Instead the designer can now see the display cannot be adjusted based on the output of the sensor. The pans that were previously performed when the sensor was out of range must be moved to the start and end of the sense-able range of the sensor. If the application had taken the screen viewing angle into account at the start then the application would be more flexible given a greater level of accuracy in the input.

The framework highlights the range of values that sensors may provide. However, as the Augurscope example shows, the application needs to be made aware of the range of sensor values. If a new sensor appears to an architecture then simply being flexible enough to integrate the data is insufficient. The architecture must be able to relay the metadata of the sensor so the application can use the data in a sensible way and understand that the sensor may no longer be the limiting factor. The relayed metadata may be the range of a title sensor or its accuracy across its sensing range.

3.3.2 Seamfulness

Current trends in the Equator projects and other context sensitive systems have been to abstract the sensors from the application. This can be useful to allow the application to take a more data-centric view of the world. However, the infrastructure-level changes that are then hidden can aid the user in understanding the application's behaviour. By showing the 'seams' in the infrastructure [MCRS02] the user's experience can be improved.

One example of these seams is with mobile phones. In some town areas reception can appear good yet the phone call is choppy or may be disconnected. Not only is this frustrating for the user but they have no idea why the errors occurred. If the seams in the infrastructure are made visible then the user can make use of that extra information. Some networks allow simple location services such as displaying the local postcode on the phone. This is done by the base station the phone is connected to. In the preceding example, the user would have noted that the displayed postcode was changing, thus hinting that the network was busy and their call was being moved around a number of local base stations. This extra information about the seams in the system can allow the user to either move away from the area served by a number of base stations or to 'see' when the network is busy and to make calls outside of this time.

The *Sense and Sensibility* and *Seamfulness* frameworks are important in this work as the dynamic management software must note how its operation may affect the user's interaction with their application. This means a balance must be struck between abstraction of the sensors/infrastructure and the functionality the architecture provides. It's all very

well to have the architecture provide the application with high quality streaming video but if the infrastructure cannot relay the cost of the data then the user's experience will be affected. For example, receiving video over the local wireless network is probably free whilst receiving the same data via a mobile phone network is extremely costly.

Chapter 4

Dynamic Management of Sensed Data

This chapter introduces a number of issues of stream management before presenting various design features to over come these issues. Each of the features adds to the overall design of the context sensitive systems, however each of them comes with their own 'baggage'. The 'baggage' may take the form of requiring a large amount of code to be written to solve a relatively simple problem or ensuring that a particular style is used. These features, when incorporated into an architecture design, can aid the designer to maximise the flexibility of their architecture. By incorporating the frameworks from section 3.3 the designer can capture the available information and design the system to cope with future enhancements.

4.1 Issues with stream management

This section describes some of the issues that the current platforms do not adequately handle and so should be taken into consideration when developing any new system. These issues are drawn mainly from the work on the OSC system described in section 2.2.

- Ease of Integration

  By abstracting over the physical sensing devices, the client applications are not required to know how every sensor in the system works in order to use its data.

- Consistent Mappings

  By using consistent descriptions of the data supplied from the sensors it would be easier for the system to provide meaningful data responses. If this is not possible then the provision of a translation to a defined description will aid a system in providing meaningful data responses.

  A defined meta-language would also allow the application designers to worry less about the system not being able to generate a transformation from available sensor data. Provided the requested information was defined in terms of this meta-language then the information could be generated by the system.

This problem occurs with the OSC system; the ultrasonic position data is on a different co-ordinate system to that of the the dataspace. Only by providing a transformation to the co-ordinate space of the dataspace can the data be exchanged. Rather than having the client do the transformation the transformation matrix is loaded into the dataspace so that any incoming positions that are from an ultrasonic system can be integrated to the dataspace co-ordinate system.

- Conflict Resolution

  Given that the system will have various sensing capabilities, it is envisaged that several sensors will be able to provide the same data. The system will have to resolve conflicts between the data that is presented. This could be achieved using error or confidence information from the sensors relating to its current detection abilities.

- Data Provenance

  The data that is finally sent to the application will most likely have gone through several stages of fusion. That is the data will have been combined and processed along its route from sensor to application. The choices that were made, when this fusion occurred, could be of importance to the application's acceptance of the data. By providing the ability to check the provenance of the data, the application, some component of the system, or the users themselves can use the provenance as an extra level of confidence. If the provenance shows that a component is unacceptable it can be removed and the system will try and reconnect the gap without reusing the unacceptable component. The unacceptability of the component could be decided at any level from the sensor providing the data to a component in the data route, right up to the application level. The further away from the sensor source the decision is made the more that needs to be done about deciding what data items to keep and which should be removed as they may be 'tainted'.

- Ease of update of components

  Many applications are built from the sensing technology to the interface as a complete solution. This would require that the manufacturer ensure the sensing technology worked in all places that the devices were set to be used. This may be fine in a single deployed system, however, on a larger scale this may be prohibitively expensive. If the application or underlying system can leverage existing sensing technology then the deployment costs should be reduced or at least shared. This would also help cut development time as the sensing hardware and interfaces will already be defined, allowing more time to be spent on design and coding of the new application.

  To allow this sharing of technology the framework must be capable of understanding and interfacing with various sensor technologies. There are many examples of current systems that employ a form of update to either upgrade whole or part of the application. A number of different approaches have been taken to address these issues. If we take the example of adding a device to a PC, invariably the operating

system will either ask for the driver disk to be able to understand and control the device or already contain a suitable driver. However, neither of these solutions is desirable. The handhelds do not have enough storage for a large number of sensing technologies, nor is it desirable to require the user to continually update the software.

- Feedback Profiling about Components

  A component will need to be self-managing so if it is unable to handle the number of requests that it is receiving then the sending components will have to be notified. They may then decide to reduce their data flow or request additional handling capacity.

- Context change detection

  The components that are controlling access to the data will need to be aware of the changing contexts. They use this information so they that can revoke access should the context deviate too far from the authorised usage. For example, displaying a map with a user's private location on a public display.

- Controlling Data Usage

  Linked to the above issue is the method by which control of the data can be exerted even after the data item has left the owner's device. This will allow the handling of access to be linked to context changes as the data is being used.

- User Notification

  The owner of published data items may wish to know how their data is being used and so some form of notification would be required. The larger issue is how to relate this notification to the user through their limited display. If there are several hundred users of the data items, then there could be an inordinately large number of notifications. The issue of user notification extends beyond the bounds of this dissertation. However it is important to understand that this issue exists as any form of user notification will require underlying system support to relay data usage information.

- Static Metadata

  Metadata was used in a limited capacity in the OSC system for event description. In addition metadata is used at connect time to initialise variables about the data stream associated with connection. The connection object has variables for storing these values and it requires that they are provided at startup. This works in the current setup. However, in the more general case it is conceivable that the stream from the handheld may not be associated with a user. The stream may be associated with an object in the room or simply with the room, so requiring the association at connection time limits the use of the streams. The binding of user to stream would also preclude the user from having more than one stream say from a number of devices.

In the OSC system the handheld uses an ultrasonic system to determine its position. This system has to be calibrated from information stored in a file. This configuration file is loaded at startup which contains the accuracy information for the ultrasonic system in use. The metadata about the sensor's operation is static and is not updated at runtime, so only one ultrasonic system can be used.

- Identification through Correlation

  The user identification information may not be available directly from the handheld. It may only be through the use of other sensors that the correlation of the handheld's position and a known user's location can be used to ascertain the user of the handheld. For example, an active badge network may put the position of a user close to the position of the handheld. This may be enough information to determine that they are the current user.

- Lack of dynamic support

  A more dynamic meta-data system would provide a basis upon which a dynamic data flow could be realised. In the OSC system all the transformations and analysis of the data from the sensors are hardwired into the single processor object. If the operations of this object and the circumstances under which they should be used were known they could be applied to the data when it was required. This would allow transformation of streams to allow the interoperability of data steams and sources.

- Closed Code

  The Sensor Box shown in figure 2.8 is classed as a Closed Code component as we are unable to change the functionality of the box in any reasonable timeframe. In the OSC system the handheld sensor box aggregates the streams of several ultrasonic receivers to produce a single position stream. However, by identifying each source, all of the position information could be sent to a client. Thus, the client could use its own algorithm for determining which position information to use. This would allow the sensor interface to be much simpler.

What follows is a presentation of what sensed data is, and what is done with it, from this the various features of context-sensitive architectures are introduced. These features can then be seen in practice in two case studies each taking a different approach and highlighting different benefits.

4.2 Sensed Data Representation

To understand how sensed data management works it is important to know what is being managed. As described in chapter 3, there are a number of classes of system that can be used to gather data. The class of system explored here operates by performing the sensing at the device and then sending the data to the server for use by others. While

there are alternative system architectures, as presented earlier, for this discussion only systems where the user's device senses the location are considered.

The user's device will have a number of sensory inputs which provide the sensed data input for context sensitive applications. The data arrives on the user's device as an encoded stream. This encoded stream may be delivered to the device in a number of ways. In the majority of systems the sensor is connected on the serial port so a single data stream is presented to the application, however, this stream may contain a number of data streams, e.g. Metadata and Data. This data stream is then decoded and stored in the local dataspace as a typed object. From here notifications can be sent to components that are waiting for the data.

As the data is encoded when it is received, it must undergo a decoding process. This is accomplished by prefixing each message with a block of metadata describing the type of payload. This form of prefixing the metadata can be misleading as it requires the handlers to know how to handle each custom formatting. A more flexible solution for interprocess message passing using XML was developed to remove ambiguity. Similar to the SOAP Envelope [SS00] for containing the data, the information was encoded in XML so the data could be easily separated from the metadata component. This format was also more flexible as various components could operate on the data without any prior knowledge about any special data formatting. Using simple categorisations such as position, which in turn contained an x, y and z value, the various components need only share an understanding of how to express the data in which they have an interest.

4.3 Exploiting Metadata

The metadata allows the sensed data to be more easily converted into a universally understandable format. In the City system the messages are prefixed with a string of the form `$x`, where `x` represents a message type, e.g `$G` is a GPS formatted data message. This allows the message to be parsed by code in the `Proxy` that understands the format.

Metadata can be used to handle the data at a higher level. By categorising types of data, aggregation of data values can be performed without any knowledge of the data. For example, a raw stream of numeric pairs could be categorised as position and served to applications looking for position to see if it can decode the value. This of course would only make sense within the application if the data was actually a position. This approach relies on the correct categorisation of the data, without it the data could be used in the wrong situation. This can be likened to the attempt to add meta information to web pages in the form of `Meta` tags. The `Meta` tags were designed to make finding pages easier but were used to drive traffic to websites that had nothing to do with the `Meta` tags on the page.

A similar problem could infect the use of metadata in sensor systems. Commercial interests could result in sensors that only operate with systems from the same manufacturer. As with the `Meta` tags in `HTML`, attempts to create a open standard of categorisation may also fail in sensor systems.

By tagging the sensors the meta description may be more accurate and ultimately useful for integration not just for describing data. Metadata about the sensors could provide reliability, error margins, and operating ranges making the sensor data more understandable.

4.4 Component-Based Design

The component-based design approach allows various shared aspects of a system to be generalised so that the functionality can be shared. This creates a much cleaner and easier to maintain interface to common code. The components in a system therefore have well-known interfaces so replacing a component should have no effect on the rest of the system. This method was used when designing the Metadata Component Network(MCN) system (section 4.7). Each transforming component had a standard interface which allowed the components to communicate without any knowledge of each other. The component-based architecture also provides more flexibility in a mobile setting. The OSC system (section 2.1) used a single class to perform the decoding of the socket stream. By replacing that with a component based architecture the MCN system was able to move the processing from machine to machine.

4.5 Pipe-lined workflow

Component-based architectures can also be used to develop the pipe-lined workflow as in the MCN system.

Component-based architectures stem from batch processing of data. As one set of processing is completed it is sent to the next processor in the pipeline. Given the dynamic framework suggested at the end of section 4.1 and the benefits of metadata, a more flexible approach to pipelined workflow can be realised. Each step of the processing can have a dynamism introduced so processing at each step is based on the metadata. The resulting object could then be sent to any processing component, the selection of which could be based on several factors that would not normally be concerned in a pipe-lined work-flow.

4.6 Run-Time Re-configurability

Run-time re-configurability is an important aspect of a system that aims to dynamically manage sensed data. The live system may encounter new data that it does not know how to handle. In such a situation the system must be able to re-configure itself so that it can incorporate the new data stream. This can be done by locating a new handler or transformer for the stream.

4.7 Case Study 1: Metadata Component Network System

This section focuses on how the MCN System benefits from the issues presented at the start of this chapter.

This case study was aimed at transforming unknown data streams based on the associated metadata. To that end the proxy server as described in section 2.2.3 was re-written to incorporate a more dynamic profile.

What follows is a description of the MCN proxy server and the extra system components that it required. These changes are then compared to the OSC system, weighing up the advantages and disadvantages.

4.7.1 Aims

There were two aims of this case study. The first was a proof of concept for an Automatic Path Creator (APC). An APC would allow the client system to simply specify a data type and the system could select nodes to combine to provide the required data stream. All this has to be performed at runtime. Secondly the network must perform as well as the OSC. Where "as well as" is defined as, providing at least the same data to the client application.

4.7.2 Approach

There was several approaches that could have been taken in the design of a new architecture. The decision was taken to use a node based architecture similar to the reconfigurability of the DJINN[MNCK98] system for multimedia. This approach was taken as it was hoped that it would be possible to investigate the ability to perform 'automatic path creation'. A path is the route that data flows along from sensor to client application. These paths are created as nodes and are connected together based on an ordering that will eventually provide an output that a client requested. The ordering adopts the approach of clustering similar nodes, thus helping reduce the matching problem in finding a successful path[AK96, McD99]. The connections in this case study system were simple string matching. Understanding of the path creation was important so that the client could make a simple data type request thus allowing the computer to form a path through various transforms to select the required sensor inputs. To ensure that no data was lost each node would be given the ability to store data. The output format could be adjusted by a final component so the same content is delivered as in the OSC system.

4.7.3 MCN System

The MCN system was designed around the component model shown in figure 4.1. This model was translated into nodes that would form a network of transformers. In addition to the node network, the system also includes the addition of two services: a resource manager and a query service. In this case study the handheld software was to be left unchanged so it continues to connect to the server using a socket protocol. The following sections describe in more detail the working of the three main components.

Node Network

The nodes in the network were developed based on the Gray Salber Sensed Context Data Model for components [GS01]. Taking this idea of a component and infusing local processing capabilities is similar to the adopted with GRUMPS components [EAB+02]. This gives the nodes the ability to act as transformers of the data.

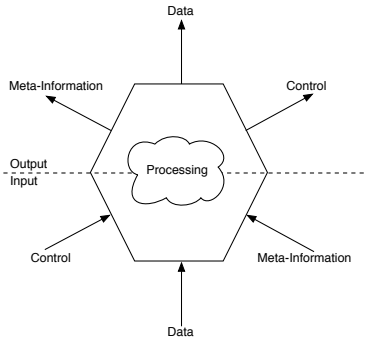


Figure 4.1: Gray-Salber Sensed Context Data Model.

This network is depicted in figure 4.2. It replaces the data flow shown in figure 2.5. The network works by creating components that process messages from a source. In the case shown the user has already established a connection. This is shown in node SH_U, the `SocketHandler` for the user. The connection process creates a User Consumer node(C_{User}) that gathers all the information of a given level, such as Position, and updates the information in the dataspace[1]. The SH_U generates String($) messages which the Transformers ($_{consume}T_{produce}$) use to create richer data. It is this richer data that the Consumer node is looking for. These nodes are created from basic components, which will be described next.

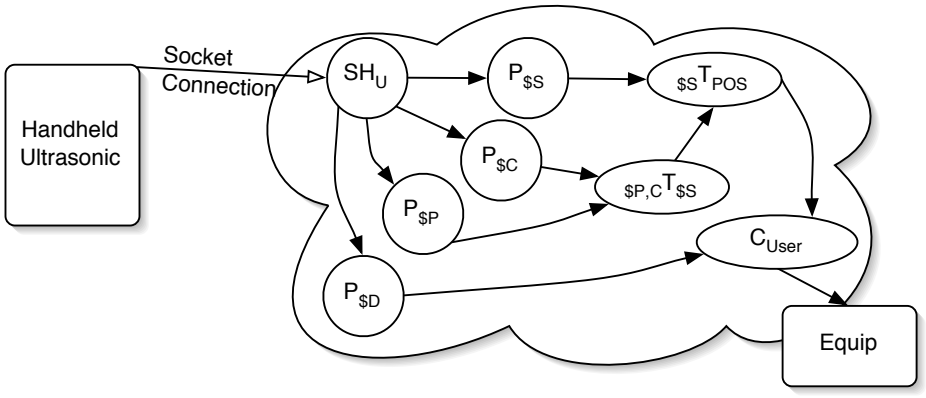


Figure 4.2: Data flow to Equip through the component infrastructure.

Three components were designed in turn, each building on functionality from the

[1]This is not the behaviour the system was intended to produce. The client should not be created by the sensor source. However, as this approach was being integrated with an existing system this was necessary. The creation of the node network was still generated by a request from the Client.

previous component. As such, they were implemented as inherited classes, as shown in figure 4.3.

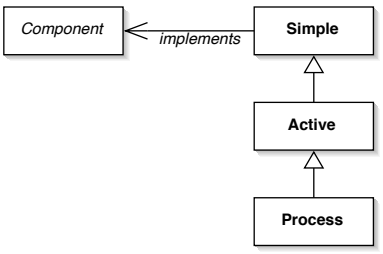


Figure 4.3: Component Inheritance of Nodes.

The basic component `Simple` contains all the code that handles connection between components. It handles the reception of messages along the control channel for assigning and removing listeners. In addition, this component maintains two message queues: one for incoming messages and one for outgoing. This was the simplest method that would allow each component's thread to pass on the message as quickly as possible.

An alternative solution would be to use a similar method commonly found in web server implementations where the incoming message is handled by one thread and that then returns the data. If the components are required to remember any state from previous transactions then the web server model is not sufficient. Data could have been stored in a shared object to provide state between transactions. However, as a number of the processors required access to state data it made more sense for them to store that locally.

The current implementation only has one thread that processes messages and so the internal state of the component can be used for processing. If each message had its own thread then synchronisation would be a bigger issue and internal state would not be possible if each thread influenced the values. The `Simple` component does not have any state, but the next component in the hierarchy does.

The next component is the `Active` component. This component stores the last value it has seen from all of the components that connect to it. This is done so that the path of a data item can be quickly returned on request. The `Active` component is more than a simple filter or aggregator. When a message is received then an operation is performed on the data before it is sent on to the next component. This was an optional operation but it allows for transformations on the data. This sort of operation can ensure that incoming data is formatted correctly.

The final component, `Process`, adds the ability to perform an operation on the data that is sent from the component. This allows the component to perform a function based on all of its inputs. Taking an example from the Equator system, the combination of 'heading' and 'position' messages were converted to 'situation' messages. The component is set to only send a situation message when a new position value arrives. So when a heading value is received no output is generated.

Resource Manager

The Resource Manager (RM) maintains a list of nodes. There are three lists: consumers, producers and all nodes. A request is made via a description of the node. By using the metadata the resource manager attempts to find a matching live node. If the resource manager is unable to locate a live node then it can create the required object provided the description is stored in the resource manager.

The RM contains references to all of the nodes that are currently live within the system.

Query Service

The query system is the point at which the clients and sources register with the infrastructure. The service tries to locate data streams for the client's request. If the request is unsuccessful then this would need to be reported to the client. The current system does not report the unsuccessful queries. The system simply takes the request to add a client as a request to connect it to the infrastructure. The client expects to receive a control message with the connection information for a suitable node. The client then has the ability to cancel the connection or request an alternative query.

The query takes one of two forms: a request for either a `Producer` or a `Consumer` type, which is matched with the available nodes or a query that specifies both `Producer` and `Consumer` types. A list of nodes matching the query is returned to the client so they can either make a connection or carry out further interrogation of the node.

4.7.4 Discussion

This MCN system relies heavily on the supporting infrastructure of the Resource Manager and the Query system to provide the components for transforming the data streams. The network provides an efficient method to customise the data flowing through the context system. The data is only transmitted to those that have requested the information.

As all information flow is requested this allows the generation of the network to occur in one of two ways: either starting at the client and then transforming the data types until a suitable source of data is found, or starting with various data types and transforming their output until the requested data type has been reached. In small setups this will work, where the range of input types is a small set directly relating to all possible client requests. The client requests must be for an input type or a type that is easily obtainable. As the number of inputs and client requests increases, the difficulty in achieving a timely transformation becomes more difficult. In a system from IBM [CPT$^+$] the problems associated with meeting the client's requests in such systems are described along with their `iQueue` system that starts to address the problem. There are approaches, such as taking the minimal cut [Lev00], which can aid in reducing the complexity of this issue however, the ability to transform the data types is still dependant on a developer writing a number of transformers that are capable of using suitably diverse inputs. There is only so much of a transformation that is possible, for example, using a temperature probe to gauge car speed. While this may be possible the logic that is required for this is so application

specific that the processing would be best done by the application not the network.

4.8 Case Study 2: Dynamic Module Loading System

The Dynamic Module Loading(DML) system was based on the MCN system. The needs of Equator's City project changed to include the handling of privacy and disconnected abilities. As such, the previous architecture was radically altered. The changes were as much a result of the increased power of handheld devices as a growing desire to explore the nature of disconnected behaviour.

This section starts with an overview of the new system before describing the modifications that were made to the OSC's proxy server.

4.8.1 Mobile System

The Co-Visiting system operated within a single room. The next stage was to move out into a city-wide system. Through a number of scenarios[2], a new system was developed to cope with the complexities of interaction out of the confines of a single room.

These scenarios highlighted several additional components to enhance the interaction that were not available in the Co-Visiting platform. Staying with the visiting analogy, a user may wish to use the internet to do some research on the place that they wish to visit. This 'pre-visit' information would be stored and serve as pointers when the user was in the city. When the user visited the city they could to take photographs and create annotations on their device so that they could reflect on their experience later. They would be able to make use of larger public displays that were in the city. For example, the user may display a map on a larger screen showing the positions of their friends so that the route is clearer.

This system is designed to record the movements of users, and, as such, generates a large amount of information that could be presented back to the users to inform them of what other people did in similar situations. These recommendations can be used to suggest alternative venues that may be of interest along their current path. Alternatively, the user could explicitly query the system to recommend a place based on their context, such as a place to go for lunch.

System Design

In developing the system a number of solutions were possible: An area of town could be covered with 802.11 wireless, the slower GPRS network could be used, or the device's reliance on permanent connections could be removed. This would require more processes to run on the handheld and the collaboration would suffer as there is less live data available. It was a combination of all three that eventually won out. Reducing the device's requirement for network communication gave the device the flexibility to use any available network to relay information that it had to exchange.

[2] `http://www.dcs.gla.ac.uk/scripts/global/equator/moin.cgi/NewNewThing`

This change in approach required that the mobile device's code be re-written. The development of a pure Java version of the Equip system coupled with the increased processing power of the handhelds allowed the entire dataspace and data handling service to be moved to the handheld. This movement was a benefit as it gave the control over the data back to the user. By disconnecting the network card they could guarantee to be invisible to the rest of the users whilst the local applications still operated.

Since the data handler was now running on the mobile device, several changes had to be made due to increased resource requirements necessitated by the MCN system (section 4.7). The solution was to blend the benefits of MCN handler with the simplicity of the OSC system's `switch` based system.

4.8.2 Aims

There were two main aims for this system modification. The first was to reduce the computational and programmatic overhead of the MCN system. The OSC system was simplistic and difficult to follow, so the DML modification hoped to capture that simplicity. The second aim was to maintain the dynamic plug-in architecture that was introduced in the MCN.

4.8.3 Approach

The approach taken for this modification was to create an abstract component system so any unit could be replaced with ease. The emphasis was on the stream handling components. When a new stream is retrieved an attempt to locate a handler is instigated. This implementation simply did a class path lookup however it was envisaged that the production system would be capable of looking for handlers from the Equip dataspace.

4.8.4 System Components

The mobile system is mainly composed of the same elements as the first system. The main change is the location in which it runs. As shown in figure 4.4, the majority of the code base has been moved to the handheld. The figure only shows the new handheld configuration since the original server-side processes for the web and VR clients are unaffected. They are not shown as this new architecture has more of a mobile focus and as such the full system was envisaged to be made up of several of the mobile units.

The mobile units are capable of running both web and VR displays so there is no loss in functionality. The only limiting factor is the specification of the mobile unit; as the web browser requires a reasonable sized screen and the VR client requires a lot of processing. The interaction with the system was not changed significantly from the previous system Most of the changes were concentrated in the middleware layer with the introduction of `Gizmos` and the `GizmoBelt`

Gizmos and GizmoBelt A `Gizmo` is a single component of the Mobile system. These gizmos are not a complete application on their own. However, with the `GizmoBelt` that

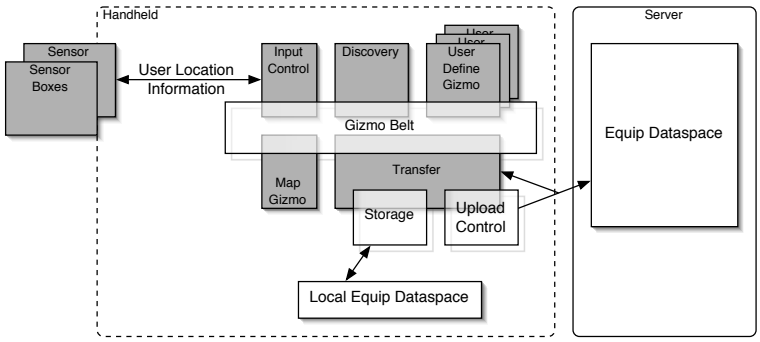


Figure 4.4: Mobile System architecture diagram highlighting the increased mobile processing.

provides a framework of common facilities such as storage, communication and privacy protection. The `GizmoBelt` is based on the Java Applet Context. This allows the various applet properties to be leveraged. The Applet Context provides security and 'functionality limiting' of the individual gizmos. This will stop a gizmo from accessing a resource to which it does not have access, in the same way that an applet cannot, by default, access users files. The belt also allows a simple mechanism for management for adding and removing gizmos.

The `GizmoBelt` is an interconnect between individual `Gizmos`. It provides a suite of standard functions designed for storing and retrieving data from the local storage space. This was developed to allow the new architecture to better handle disconnection. The OSC system used the properties of the dataspace connection for local processing. The connection to the dataspace also included a space at the local end of the connection that could be used to store values before sending them to the main dataspace. The problem with this was that if the connection was interrupted then the data stored in the local end was lost. Creating a dataspace on the handheld capable of storing data will prevent this lose. By running a local dataspace the local applications can use this to communicate with each other without requiring an active network connection.

This was shown to be of immediate value for handling the user's position, as gathered locally by a sensor box. Previously, the location information from the box would have to be transmitted to the global server before any local applications could use the values. Now with the data being written locally by the `InputGizmo` and the `GizmoBelt` the local `MapGizmo` is capable of operating without a permanent network connection.

The introduction of local storage may fix the disconnection problem with the OSC system. However, it introduces a complication for the collaboration of users as described in section 2.2.1 if this DML system is to be used in a similar collaborative environment as the OSC system. Figure 4.5 compares the update events that are generated in the two systems. The OSC system requires all clients to connect to a central server to receive their updates. While this may ensure that individual users can obtain data about all users it does, however introduce a single point of failure. In the DML system the reliance on a central server has been removed however, clients must register with each other to receive

updates.

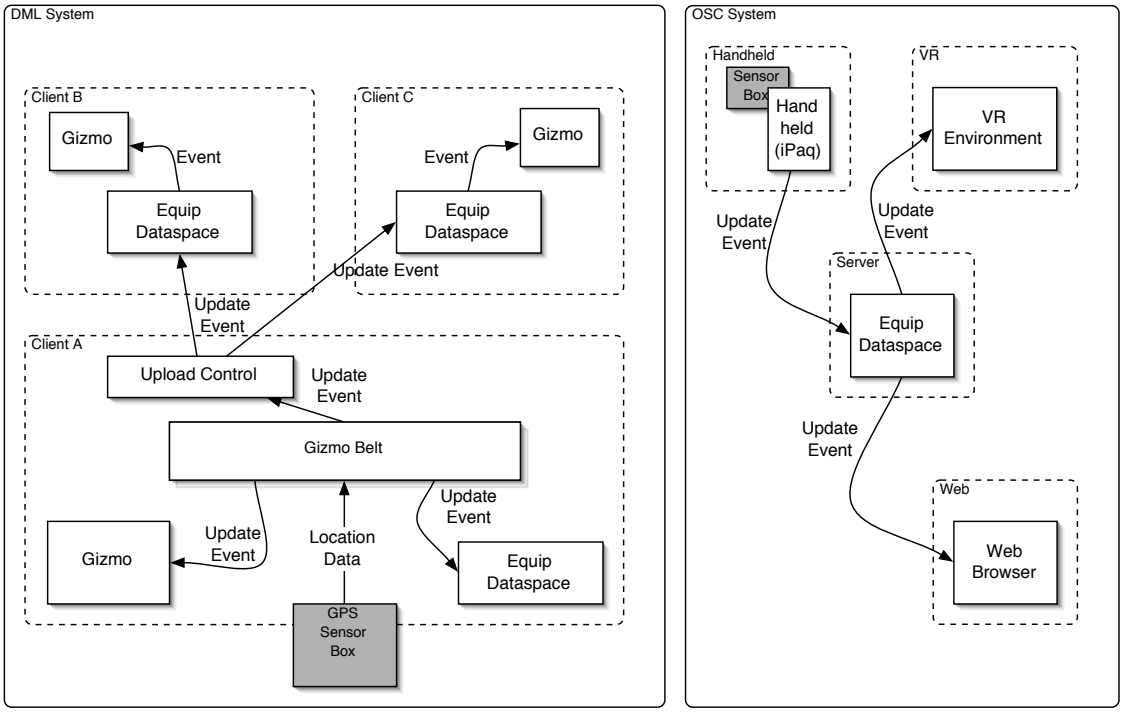


Figure 4.5: Comparison of OSC and DML data paths: The DML data is sent from one user to all other listening users while the OSC data is sent via a central server.

This extra overhead on each client is balanced against a system with improved privacy as the user can control who to send the data to. Also each client is a self contained system not requiring an external connection for functionality.

Global data transfer While the system will function in a peer-to-peer approach with clients sharing data in an ad-hoc mode; to improve collaboration between users the DML system would usually be deployed with a certain level of infrastructure support. A number of shared global dataspaces would be left in strategic places to facilitate data sharing.

The data copying is performed by the final two `Gizmo`s that make up the standard functionality of the new handheld platform: the `DiscoveryGizmo` and the `TransferGizmo`. The `TransferGizmo` is responsible for observing input to the local space. Events and data updates that are not part of the system maintenance, *i.e.* new data from inputs or local processes, are then copied to the remote dataspace on the server, when a network connection exists. The proposed architecture, depicted in figure 4.4, suggests that there may be a single global dataspace. This is unlikely to be the case, as there maybe many more local dataspaces associated with a particular network coverage area. Users would only be able to share information on the local level unless this global dataspace was part of a hierarchy of dataspaces, as described later in this section.

By increasing the number of global dataspaces the mobile user's software will have to handle numerous connections. The `DiscoveryGizmo` does exactly that, using multicast messages, in a similar manner to JINI [Wal98]. The `Gizmo` discovers the currently available

dataspaces so that it can populate them with the information that has been gathered locally.

The data contained in these local dataspaces should contain local information, such as recommendations for local cafés or restaurants. In effect these 'nodes' become a local shared noticeboard. While the user may wish to publish only certain data the system may record anonymized information such as where the user has been.

Mobile dataspace hierarchy The local equip dataspace is a fully functional space equivalent to that which runs on the server. The global space remains as before however the new found mobility in the system introduces the opportunity to have multiple global spaces. These can be connected or operate independently in 'islands' of connectivity.

The hierarchy of dataspaces shown in figure 4.6 allows information to be shared at various levels in the available spaces. A request for data is first made to the local dataspace. If the request cannot be resolved locally it can be passed up the hierarchy. For example, a *User Location Information* request made within the Mack Room space in figure 2.4 may be passed to the Lighthouse space or higher as requested.

The hierarchy of dataspaces would be useful, especially in cases like the Lighthouse, when dealing with valuable information. If the data is only available when you are connected to the given dataspace then physical access control can be placed on the area where the dataspace can be reached. The ability to place small-coverage local dataspaces allows more detailed information to be stored and maintained by local people. Users who connect to these dataspaces will be able to benefit from the richer data now available to them.

Islands of connectivity While the current implementation only operates with a single global space, the possibility is there to implement a hierarchy of dataspaces. The `TransferGizmo` would then be able to write to only a subset of visible dataspaces in the knowledge that the information would be passed to be available to a larger community. Figure 4.6 shows an example of what the available dataspace coverage could be like. A selection of connected spaces that can share information (shown with solid objects) and isolated spaces in 'islands' of connectivity (shown with dashed lines).

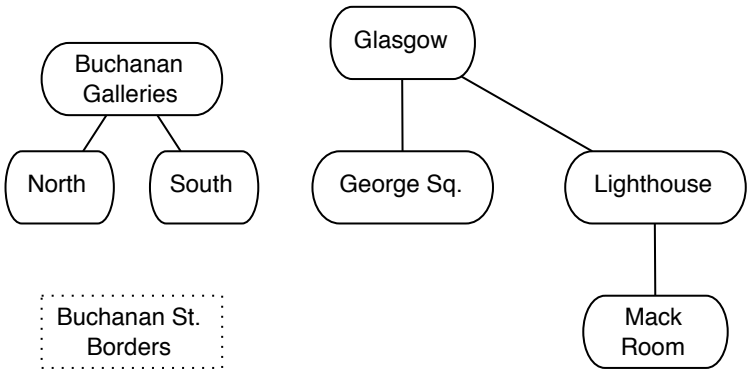


Figure 4.6: Example of the hierarchical dataspaces.

DML Proxy The operation of the proxy was very similar to that of the OSC proxy. When a message is received, a request is made for a handler that understands this form of message. The handler is then loaded by the proxy. If more than one handler is discovered the proxy is able to display any list of handlers for the user to choose between before attempting to search further a field for a suitable handler.

In essence, the OSC proxy was reduced to a single handler by replacing the `switch` statement with a dispatcher that passed the input data to a module handler that would dynamically load the required module to process the stream[3].

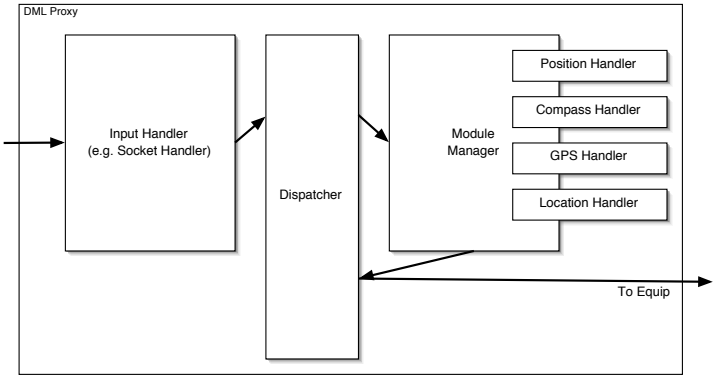


Figure 4.7: The DML proxy separated the responsibility of input, dispatching and handling.

The input handler could be replaced with an alternative such as a file handler that would play back a log file as if it were live data. The environment in which the proxy runs has also been changed. In the original system the proxy ran as a separate Java application, the new design for the handheld provides a single JVM as shown in figure 4.4 within which all the applications should execute.

4.8.5 Discussion

The DML is more suited to the Mobile system's type of architecture. The Mobile system's self contained approach can operate in a larger range of locations as it requires no external connections for data or processing. It is important that the device running the mobile system continues to function given the different sensors that could be encountered. This is made possible by the DML proxy's ability to load processing modules. On encountering a new sensor system the device could request or could search the available resources for a processing module to allow the new sensor data to be integrated with the current sensor data requests.

[3]In the example all input streams had modules. However, in a live system some streams would not be able to be processed without an appropriate handler

Chapter 5

Evaluation

This chapter introduces the approach taken to evaluate three approaches, OSC, MCN, and DML for the Co-Visiting and Mobile system. The evaluation is based on two methods: a metric evaluation in section 5.3 and an analysis of the programming effort required for each of the architectures in section 5.4.

5.1 Software Description

The Co-Visiting system (chapter 2.2.3) operated in the classic socket server model, spawning a handler for each new incoming request with the OSC system. The MCN system changes (section 4.7) attempted to apply various transformations to the incoming data to change the type to one compatible with the system. Finally, the Mobile system with DML (section 4.8) used a dynamic module loader to augment the server model to allow functionality to be added at runtime.

Many of the design decisions in the three systems were made to incorporate the schedules of the project. As such, all three systems have areas that are very much works-in-progress.

5.1.1 Co-Visiting System with OSC

The Co-Visiting system static loaded values from a file whose contents represented the current room parameters. This approach worked well for the small component that it represented. The proxy was designed simply to transfer the socket-based protocol data from the handheld to the Equip dataspace. The development of the proxy did not follow rigourous design methodologies but rather implemented minimal functionality. This caused problems for later development. The offset values for the ultrasonic location system were specific to each installation and, had to be loaded from file. One problem lay in the need to recompile when altering hard-coded values and references. This, coupled with the dependence on the local file structure to provide the required parameter file, results in a system that is not very flexible. This is also highlighted by the coupling to the Equip data space as the proxy must create the required objects and maintain a reference to the space to store and retrieve data.

5.1.2 Co-Visiting System with MCN

The MCN system was an attempt to address some of the issues with the original system and also introduce some new elements of research. By introducing the component architecture the system was made more distributable. This would allow the project to look more into the issue of disconnection, since code could be migrated in preparation for disconnected operation or re-created to maintain the local operation without the main data space.

As described in section 4.7, this method used a component architecture based on the Gray-Salber Sensed Context Model [GS01]. This allowed components to be combined dynamically at runtime to transform the data streams to suit the requested need, provided a suitable transformation could be found. These transformers could be loaded and unloaded at runtime to improve maintainability over the OSC proxy. Using discrete components with a unified communication protocol allows the storage medium to be uncoupled from the processing as the final object that gathers the transformed data in the system need be the only one that knows anything about the Equip dataspace. Thus, any changes in the storage system will have a limited effect on the incoming data streams.

This system improved the runtime dynamism for handling new data streams and data requests, the advantage over the OSC system here is that no restart or rebuild is required to take advantage of any new functionality.

5.1.3 Mobile System with DML

The MCN system was intended to introduce new technology to the old architecture. The mobile system is the next step using a new architecture. This architecture, described in section 4.8.3, was a different approach to the problem. The handheld now runs a larger portion of the software, thus allowing for better support under disconnection. This naturally results in the desire to have a smaller and more compact input handling system. The DML system draws from both the simplicity of the OSC system and the dynamism of the MCN system. The result is the `InputGizmo`. Using the component model from the MCN system and a dynamic component loader results in a system that is capable of requesting new components from the local file system and across the network, if required. Unlike the MCN system, which uses a known interface to achieve the dynamism, the `InputGizmo` uses a naming convention to link incoming data streams to the handling component which can be modified at runtime.

The new architecture also introduces new abilities that were not present in the previous systems. As the `InputGizmo` is part of the larger `Gizmo` architecture, it benefits from the shared environment. This permits decoupling from the storage medium, so any change in storage will not affect the processing. The socket connection that provides the data input to the DML is replaceable as described in section 4.8.4 so the individual handling modules need only be capable of receiving the string input for processing. Finally, as the entire `Gizmo` platform operates on the handheld, the `InputGizmo` need only handle a single connection at any one time. This gives an advantage over the other systems as state can now be stored between data items in the handler.

5.2 Qualitative Evaluation of Software

Given the diverse nature of each of the architectures for input handling, an approach for comparison was needed. The solution was to look at the amount of effort that was required to add a new sensor stream. By evaluating the work required by the software engineer to enable the new stream in the architectures the aims of flexibility and dynamism can be evaluated.

What follows are five areas that highlight the relative differences in each of the architectural approaches. The areas aim to show the flexibility and dynamism present in each of the system.

Cost of implementation The cost of implementation is hard to demonstrate using numeric values, yet it is an important factor for choosing an architecture. The chosen architecture may provide much of the desired functionality. However, if the cost of implementation is too high then the functionality is likely to be left out of the system. Simply listing the code is insufficient to quantify the cost of implementation as the number of lines of code will not reveal anything about the level of complexity within the system. Only the overall complexity within the system can be an indicator of the cost involved in making changes. The code can give some idea as to the cost of implementation by observing the effort that was required to perform a similar task within the current system, such as adding a new input device.

What is the cost of implementing a new input device?–This can be dealt with in two parts: first, how much code must be reviewed before the developer can understand how to add a new input device table 5.1; second, how much code must be written to add the new code. By analysing these details, the flexibility within each of the systems should be identifiable. By reducing the amount of 'glue-code' that is required to be written whilst minimising the code that must be reviewed, the cost of implementation for a particular architecture can be reduced.

System	Reviewed Code
OSC	Switch Code for method. Storage Mechanism.
MCN	Transfer classes for method. Metadata code for transformation code.
DML	Abstract Dynamic class for method. Class Naming structure.

Table 5.1: The sections of code that must be reviewed in each system to understand where the new method code must go.

This analysis highlights the individual way in which the three systems work. While the MCN and DML solutions both only require the creation of a new class, the OSC system required the developer to change the existing code base as shown in table 5.2. A mistake here may render the system unusable.

System	New Code
OSC	Add switch code to access method.
MCN	Create Transfer class.
DML	Create Dynamic class.

Table 5.2: The new code that must be written in each system to incorporate the new sensor.

Cost of change Adding a new input device may require changes to the current system if the sensor protocol is unknown. This comparison should reflect both the system's flexibility and dynamism. The act of adding functionality to the architecture, described above, has an associated cost. There are several different types of cost that could be looked at, financial, time, functionality etc. This analysis looks at the cost of the developers' time and what must be done to make the change in addition to the development of the solution. It should be noted that this is a one-off cost for the initial development and deployment of that functionality. In any system it is the cost of ongoing maintenance or changes that are greatest. The cost of change covers the amount of change that is required for the additional functionality. By minimising the number of changes or localising changes, the cost can be reduced. Table 5.3 lists what must be done as a result of making a change to each of the systems.

Changes due to additions

System	Cost of Change
OSC	Switch code to access method, recompilation, down time during redeployment.
MCN	Manual loading of class to Resource manager.
DML	Ensuring new class is made available.

Table 5.3: The qualitative cost for making a change to the systems.

The cost of change highlights the major reason for a redesign of the OSC system. All modifications required a recompile and restart of the main process. The revised systems can be fully modified at runtime and so maximise the system uptime. This allows a much greater range of changes than previously possible.

Range of change The range of change is a measure of how flexible the system is with respect to any change before the design becomes a hindrance to the proposed change. Given an architecture, there is a limit to the number and type of changes that can be made before the initial design of the architecture becomes a hindrance to further changes. There are two sides to this: run-time and compile-time changes. Compile-time offers the ability to perform a much larger set of changes. The system must be restarted before these changes are available which implies downtime. This larger set of changes, however, is still limited by the framework of the architecture. Run-time changes are only possible with an

architecture that supports them. These are the changes that are most interesting as they give the architecture increased flexibility and dynamism.

System	Range of Change
OSC	As the entire proxy is a single class then any compile time changes can be made subject to the cost of change. No run-time modifications are possible.
MCN	New transformations can be added without interrupting the current configuration. New types can be added for storage purposes however this requires replacing the client storage unit which may result in duplicated data if the inter-connections are not carefully connected.
DML	Storage is not handled by the proxy so is dependant on the storage mechanism's flexibility. The proxy is unable to replace currently loaded handlers however it could periodically check to see if a newer handler is available and replace the current hander for the incoming data.

Table 5.4: The range of changes possible with each system.

The three systems present various levels of flexibility as described in table 5.4. While the OSC system can be seen to be highly flexible, this is only due to the fact that it is a single class and so its flexibility comes from being small in size. The developer can rewrite the code to do anything. However, if the code base was a much larger monolithic system changes would be much harder. The other two architectures must work within their frameworks if the components are to maintain communication. This results in the two systems being limited by some form of their framework. The storage mechanism limits what data can be stored in the MCN system. So, provided that the MCN architecture can transform the incoming data to something that the storage mechanism understands, it will cope. This is because the MCN system must couple a client component to the dataspace. If a new stream cannot be absorbed into a currently known type in the data space it must either be ignored or the client upgraded. It is the format of the data packet from the sensor that is the controlling factor in the DML system. If the data format is changed then the dynamic dispatcher must be changed to understand the new format. Currently this cannot be performed at run-time. However, due to the use of a small core thread that then starts the other processing components within the DML system it is possible to allow dynamic replacement of the dispatcher.

Points of coupling The final area for comparison is Coupling Between Objects (CBO) as it is a better measure of complexity for object-oriented code than more simplistic measures such as Cyclomatic Complexity (CC) [MCIHB99]. All three systems have common couplings, such as the storage mechanism which is common throughout. It is the way in which the coupling is handled that affects the coupling levels in each.

The OSC system only has two classes and so CBO is a poor measure as the two classes are highly coupled due to their design. This makes the complexity easier to manage. In

System	Points of Coupling
OSC	The OSC system is only two classes so CBO is limited to the coupling within the storage class. In this case the CC gives a better understanding of the overall complexity.
MCN	The coupling in this system should be quite high due to number of classes and the hierarchical approach.
DML	The complexity with this system should be the lowest due to the reduced number of classes and the self contained design approach.

Table 5.5: The points of coupling within each system.

comparison, the MCN and DML systems mask the coupling of several objects from the developer by requiring the use of an interface.

The work required by a software engineer varies for each system. The approach to take for a particular system will depend on more factors than have been presented in this section. Small development systems would appear to suit the use of a OSC system due to the small code overhead. However, both MCN and DML systems offer a long term scaleability that OSC systems cannot match.

The next section takes a numeric approach to metrics, comparing the amount of code required to perform a simple change to each system. This should provide a clearer understanding of how each system is best suited.

5.3 Quantitative Evaluation of Software

This section presents a numerical analysis of the code using a selection of metrics to enhance the findings from the previous section. These metrics were chosen as they best highlight the differences between the three systems with respect to their architectural approaches. That said, Lines Of Code (LOC) and McCabe Cyclomatic Complexity Numbers (CCN) are not always a good indicator for the complexity of a given system [MCIHB99]. This is especially true when talking about object-oriented programming languages, such as Java. Coupling Between Objects (CBO) is a better measure that compares the number of references to other objects for a given object. The metrics should highlight the relative complexity involved in developing each of the three systems, with an emphasis on the individual handling routines for the input streams. The results of these metrics should yield some background into selecting an architecture that provides a suitable balance between flexibility and complexity.

5.3.1 Source code

Wherever possible, the source code used in generating these values was chosen to give a similar starting level of functionality from each system. The chosen metrics are not dependent on external classes being available as it is only the code for a given class that

is required to perform the analysis. In each of the three systems, the entire code that was required for transmission of the input data from the socket reception to the final classes that modify the data was used. The classes that actually deal with communication to the dataspace have been omitted from this comparison for two reasons:

- The first two systems share the same supporting code for this function and the DML system uses a shared resource as part of the new mobile framework. Including the values from these classes would not alter the evaluation as each of them would require the same modifications to cope with a new data type.
- The differences between the two versions of the data storage routines are in class naming and access protection only. They are in essence the same code so to include it would not highlight a difference between the systems.

The full list of classes/methods chosen for each metric can be seen in the tables of Appendix A.

5.3.2 Metric Generation

The metrics presented here were generated using two programs: JavaNCSS[1] and TogetherSoft's Together development environment.

JavaNCSS was used as it provided method-level analysis of the code. The software calculates the Non Commenting Source Statements `NCSS`, *i.e.* lines of code and Cyclomatic Complexity Number (McCabe metric) CCN, a measure of the complexity of the code. JavaNCSS's NCSS values more accurately represent the program length than LOC metrics from other software. The simple description of what is counted is approximately each ';' and '{'.

The CCN is calculated by taking the program in a linear fashion and counting the decision points π. The CCN is then calculated by : $CCN = \pi + 1$

Together's measure of Coupling Between Objects (CBO) is calculated on a class basis, counting reference types that are used in attribute declarations, formal parameters, return types, 'throws' declarations and local variables, and also types from which attribute and method selections are made. Primitive types, types from `java.lang` packages and supertypes are not counted. The more independent the classes in the system are, the easier it is to perform a single change to the system without affecting other classes.

The JavaNCSS values for each method in the system were calculated for NCSS and CCN. The method-level analysis was performed so that the amount of code to implement a single input function, such as positioning, could be evaluated.

5.3.3 Code Analysis

This section presents the three measures that were performed; Total Lines of Code, Processing Position and Adding Position. Each highlights a different aspect of the three systems, but all ultimately reflect the inherent flexibility and dynamism.

[1] Version 21.41 was used and is available at
`http://www.kclee.de/clemens/java/javancss/`

Total Lines of Code The total lines of code was calculated by including all of the classes that were used by the systems to perform equivalent tasks. These equivalent tasks covered the transfer of input from the socket connection to the last class that processes the data. The final summed values are shown below in table 5.6. The figures show, as expected, that the OSC system is the simplest and smallest. The coupling is higher than the similar DML system, highlighting the added complexity in the OSC system. The MCN system does not perform well in this test as there is a large amount of framework associated with its operation, as described in section 4.7. The increased Object Count (OC) helps explain the elevated coupling and line count. While the increased values here represent an increased complexity of the overall system, this is counterbalanced by the fact that the MCN system has a higher level of dynamism than the other two due to the run-time binding of data stream handlers based on their metadata. The final column Coupling Between Objects per Object (CBO/O) highlights the complexity inherent in the OSC approach compared to the MCN system that spreads the complexity of the system across the code base.

113/45

System	NCSS	CCN	CBO	OC	CBO/O
OSC	338	71	42	2	21
MCN	1592	480	113	45	2.5
DML	365	127	37	10	3.7

Table 5.6: Total values for the entire system.

Processing Position To gain a better understanding of the three architectural approaches, the number of lines of code that are executed to transfer a single message were calculated. These values can be seen in table 5.7. The table shows clearly that the OSC system requires the fewest lines of code and is least complex in raw terms of CNN. This is to be expected, since the OSC main component was a single class with a switch statement to the required methods. As the MCN system is larger it requires more code, specifically in generation of the `Metadata` packets. Comparing these values to the total values in table 5.6, the code required to perform positioning is around half of the total code. However, the percentages show the MCN to be lower than the others due to the size of the supporting frame work. Similarly, the majority of the DML system is used in the processing of the data. In comparison, the MCN system has more code than the other two systems but uses a much smaller proportion of its code base to perform the processing. A large amount of the processing is consumed by the framework which provides the dynamic bindings between components.

Adding Position The varying quantity of code required to add positioning to the system is shown in table 5.8. These values show the amount of code and its complexity in terms of adding a new input type. This metric highlights the flexibility of the systems and it is here we see the difference between the OSC system and the DML system clearly. The

System	NCSS	CCN
OSC	167 (49.4%)	44 (61.9%)
MCN	384 (24.1%)	133 (27.7%)
DML	296 (81.0%)	104 (81.8%)

Table 5.7: Values for the code required to perform positioning in the system, with percentage of total code.

DML system needs only a small class file to be written that handles the incoming data stream, while the OSC system needs the same code in a method and that method must be tied into the switch statement that selects it. The MCN system loses out to the other two systems but the extra lines are required for setting the metadata values that allow the new transformer to be co-opted into the system.

System	NCSS	CCN	CBO
OSC	33	6	-
MCN	86	23	8
DML	13	3	2

Table 5.8: Values for the code required to be added to perform positioning in the system.

5.3.4 Code Analysis : Discussion

While the figures show the OSC system to be the smallest, simplest and most dynamic, and the MCN system requires far more lines of code and complexity, there are other costs that are not evident in a simple numerical comparison. The DML system relies on a mapping from the input to the handling class before it can be dynamically discovered. The OSC system must be recompiled and restarted with the new changes. The MCN system, on the other hand, does not require any recompilation or name bindings since the `Metadata` allows the component to be self-describing, hence the complexity increases but flexibility is maximised. The class currently must still be manually loaded into the resource manager. These are only a few of the differences that cannot be quantified but which must be taken into consideration during development. In the following section, these differences shall be investigated further.

5.4 Evaluation : Programming Effort

In the previous sections a metric-based approach was used to evaluate the systems. However, metrics do not give the full picture. The complexity of the MCN system is due to the way in which it operates. It is this complexity that allows the system to adapt to new data sources and sinks without any user intervention. This section looks at the programming effort that each of the systems requires of the developer, beginning by using a *cognitive dimensions* approach to evaluate the difficulties in working with the systems.

Dimension	Using
Abstraction Gradient	×
Closeness of Mapping	
Consistency	
Diffuseness/Terseness	
Error-proneness	×
Hard Mental Operations	
Hidden Dependencies	×
Premature Commitment	×
Progressive Evaluation	
Role-expressiveness	
Secondary Notation and Escape from Formalism	
Viscosity	×
Visibility and Juxtaposability	

Table 5.9: The dimensions applicable to analysis of the frameworks.

It finishes with an evaluation of each architecture, considering their respective strengths and weaknesses.

5.4.1 Cognitive Dimensions

An alternative approach to evaluating the architectures is to look at them in terms of the cognitive demands of the developer. Using Green and Petre's [GP96] cognitive dimensions as a basis, the complexity that each system puts on the developer can be analysed. The cognitive dimension framework consists of many dimensions, not all of which can be applied to this architectural analysis. A subset of the dimensions for this evaluation has been chosen, which is shown in table 5.9. This set of dimensions was chosen as they best highlight the areas of similarity and contrasting approaches.

- Abstraction Gradient

  Abstraction gradient is a measure of how much information the developer must grasp in order to understand the system. For example, an area of research in the video processing and information retrieval world is scene identification. As a film watcher, the scene changes are easy to spot. The computer must do colour histogram analysis to perform the same task. However, this analysis may be difficult for the programmer to grasp the first time. By adding a layer of abstraction, such as analysing the people and objects that compose a scene, the task is simplified.

  In the proposed architectures the data is of interest, and for the OSC and DML systems to use and process this data they must know its format and location. This creates a steep learning curve for the programmers as they must understand the data before they can move it around the system. The MCN system, on the other hand, attempts to address the gradient by introducing metadata. This allows the programmer to write new transforms that are independent of the data format. Simply looking for a stream of `Position` values will result in a connection to that stream.

The stream then provides an `X`, `Y` and `Z` value which the programmer can access without needing to know any proprietary format information.

- Error-proneness

  Green and Petre take error-proneness to mean how easy it is to introduce a fault in the program as a result of coding. In the three systems, each written in Java, the error-proneness could be influenced by a number of factors. Number of comments, complexity of the algorithms or the number of components could all affect the error-proneness. However, it follows that the more coding that is required, the greater the chance an error could be introduced. So in this case the evaluation ties back to the metrics presented earlier. If the programmer has less code to write to implement a new feature then the opportunity to make an error is smaller.

- Hidden Dependencies

  Green and Petre's paper[GP96] explains the classic problem of side-effects from functions that modify global variables. With respect to the thesis case studies, each of the systems uses the Equip dataspace as a central storage space. This allows each of the applications to communicate their data. However, this communication is a one-way stream, flowing from the producers to the consumers. This worked for the simple OSC system where the consumers simply displayed the information that was of a given format. As the system and the research developed, there was a desire to have the clients operate in a number of ways based on the input. For example, the map tool originally showed the positions of the visitors because they were always known to the system. When the software moved outside and started using GPS, the position information was less accurate, so reflecting this inaccuracy on the map would aid the interaction.

  The problem here is that while the OSC sensor source (section 2.2.3) provides an accuracy value, it was hard-coded to a value of 1, the newer sources continued to provide the same value and not the accuracy from the GPS receiver. The sensor sources attempted to provide an abstraction over the data that was received for the system. This would allow a number of other devices such as a pedometer to be attached to the sensor source without the system requiring knowledge of the source. In practice this reduces the development overhead for the middleware, however it reduces the functionality of the clients as they cannot change their representations to reflect any additional data the sensor source may provide.

- Premature Commitment

  The idea behind premature commitment is that the programmers must make a decision before the relevant information necessary for making the decision is available, for example, attempting to write a contents list with page numbers before writing a book. The OSC system required all of the input types to be known before the system could be deployed. While this may be acceptable in a development system, a production-scale system would not tolerate having to recompile the system each

time a new input type was discovered. The second, dynamic, system attempted to reduce this problem by introducing a level of dynamism when dealing with the data streams. If the streams can be modified dynamically at runtime, then by adding the required transformer the required stream can be found. This solution again works well on the small scale, however when trying to scale the information flow to a reasonable number of clients the processing required to broker the messages becomes much more difficult as described in the work from Strom et al. at IBM [SBC+98] on information flow.

Armed with all this information the final DML system was developed to be dynamic from the start. The system was designed to allow each of the main components to be exchanged at compile time, components such as connection mechanism and dispatching protocols. This was a reasonable approach as the project only had one hardware sensor system and had no plans to create a new system with a new protocol. The flexibility to be able to handler multiple devices connected to the iPaq was considered unnecessary especially at runtime. So it was considered too much to allow these components to be replaced at runtime. If such a sensor device was created then it would not be swapped while the system was running. The core of the solution was a dynamic dispatcher that did not understand any of the messages that it was sent from the connection mechanism. It simply knew how to find a component that would understand what to do with it. By allowing the dispatcher to dynamically search for a handling component when none was available gives this system a more flexible operation.

- Viscosity

  Viscosity is how fluid a liquid is, how easy it is to move. This is applied to software to mean how much effort must be applied to affect a change. This dimension really sums up what the previous metric section is aiming at. A measure of how flexible the system is to program reflects its ability to dynamically manage the data it handles. So, taking the above examples, the viscosity or "ease of change" can be seen. The OSC system was a single document requiring the maintenance programmer to sift through many lines of code to find the point at which to add the new methods. The other two systems allowed extra code to be added simply by writing the required class.

  When looking at the challenge of adding new functionality to the system, the relative levels of viscosity are more clearly shown. The problems of providing the accuracy information from the input source, so the client could adjust its display to suit, could be provided by all three systems. However, in their current state, none of the systems can send a 'quality' value to represent the value. This additional information would be useful if a number of sources were available and the client wished to make an informed decision about the available inputs. For example, a device reporting accurate values (90%) may have a very low quality (within 100m) but a less accurate device that had a higher quality may be more suited if you need a location on a street.

Each of the systems would require that the protocol from the black box be modified to send this information, but only the MCN system with the rich metadata model would be able to transmit it without any modifications.

5.5 Issues of Stream Management Revisited

Several issues relating to managing the data streams as shown in section 4.1 were highlighted with the OSC system. Taking the software that was developed in response to these issues and the evaluation presented above allows us to revisit these issues to gain a better understanding of the trade-offs. This section shall look at how the new systems addressed the original concerns, highlighting the improvements, if any.

The ideas of *Ease of Integration* and *Ease of update of components* were removed from the architectures by using the functionality that the custom black boxes provided. As new input devices were 'fed' through the boxes the architecture and so the client applications had no problem using the data. However, *Closed Code* was highlighted as a possible problem area and as shown in the previous section this abstraction caused problems when the client applications changed.

These changes were in response to the issues of *Data Provenance*. As the client became more complex, a greater understanding of what and where the data was coming from was suggested to aid the interaction. The MCN system was capable of handling this change in requirements as it had been developed for the express purpose of dealing with *Conflict Resolution*, *Consistent Mappings* and *Static Meta-Data*. The chain of components in the MCN system can be queried so the provenance of the data established. This in turn could be used in some way to identify the veracity of the data. The DML system, while less able to deal with changing requirements from the client due to its lack of feedback, is just as able to handle *Lack of Dynamic Support* by loading handlers as required. The DML system does not have the same concept of *Data Provenance* as the data is either processed by a single handler or is rejected as unknown data.

Feedback profiling about components was an issue that was never reached in the small scale systems that were being used in the project. It was envisaged that within the MCN a large number of sensors generating information and transformers modifying it would result in degradation in the efficiency and number of streams in the system. This would require a shift from a 'push' to a 'pull' model whereby the client must explicitly request a type of data rather than simply looking for it. It was argued that these two models could be implicitly linked due to the architectural model in use. As the Equip dataspace is a tuple space, the clients must request the information by placing a "request" tuple into the space. This insertion could be monitored and used to initiate the data 'pull'. This would in turn create the path of transformers between client and available sources.

Finally, consider the notion of *Context change detection*, *Controlling data usage* and *User notification*. Each of these relate to the larger issues of security and privacy that have not been touched on in this discussion but are now becoming of greater importance in the architecture and are left as future work.

The issues are summarised in table 5.10, The MCN and DML systems are listed showing how they compare against the OSC. Where the listed system is an improvement on the OSC a + is given. Issues that are handled just as well, *Closed Code* for example, receive no value. The newly developed systems were designed to be an improvement over the OSC and as such none of the issues are handled worse than the OSC. *Controlling Data Usage* and *User Notification* are issues that were beyond the scope of this development cycle and so not present in the table. However thoughts on how to proceed with these ideas are detailed in appendix B.1.1. The final issue that has not been touched upon is *Identification through Correlation*. The ability to identify the current user through a range of alternative sensors has been discussed in this dissertation but there has been no work carried out on attempting to perform the correlation through the presented architectures.

Issue	MCN	DML
Ease of Integration	+	+
Consistent Mappings	+	
Conflict Resolution	+	+
Data Provenace	+	
Ease of Updating Components	+	+
Feedback profiling about components	+	
Context change detection	+	
Static Meta-Data	+	+
Lack of dynamic support	+	+
Closed Code		

Table 5.10: A comparison of MCN and DML's abilities at dealing with the issues of section 4.1.

5.6 Analysis of Architecture Environments

The previous section highlighted the issues that the new systems tackled. However, the developer is still stuck with the question of which approach to use and when. This section shall present a method for selecting an architecture based on the developer's current understanding of their project.

First a quick recap of the three systems. The OSC is a single class with a simple switch that selects methods to perform the calculation. The DML is a step up from that as it abstracts the switch in to a dispatching class. The incoming methods are dispatched to a separate handler for processing. Finally this approach is similar to the MCN which has a number of handlers that transform small chunks of data hopefully such that something desirable can be presented to a client.

How do these descriptions help us? Well the problem that developers face when they start working on a project is that they may not be able to tell the scope of the project. If the system has been fully described and is a closed system then the developer should be able to design an architecture to handle that data. However, it is when the uncertainties of an open and evolving system are introduced then the direction for the developer is less

clear.

Figure 5.1 shows a block component diagram for each of the three systems. The figure shows the similarity of the systems. All three systems have a `Handler` that acts as an interface between the `Input Source` and the specific `handler`. The MCN and DML systems are remarkably similar. Their difference is down to their method of handling the data. The MCN relies on a server process to handle the processors while the DML has a smaller system contained within itself. This distinction coupled with the scale of the component network in the MCN when compared to the lighter-weight handlers in the DML result in the two systems having particular deployment strengths.

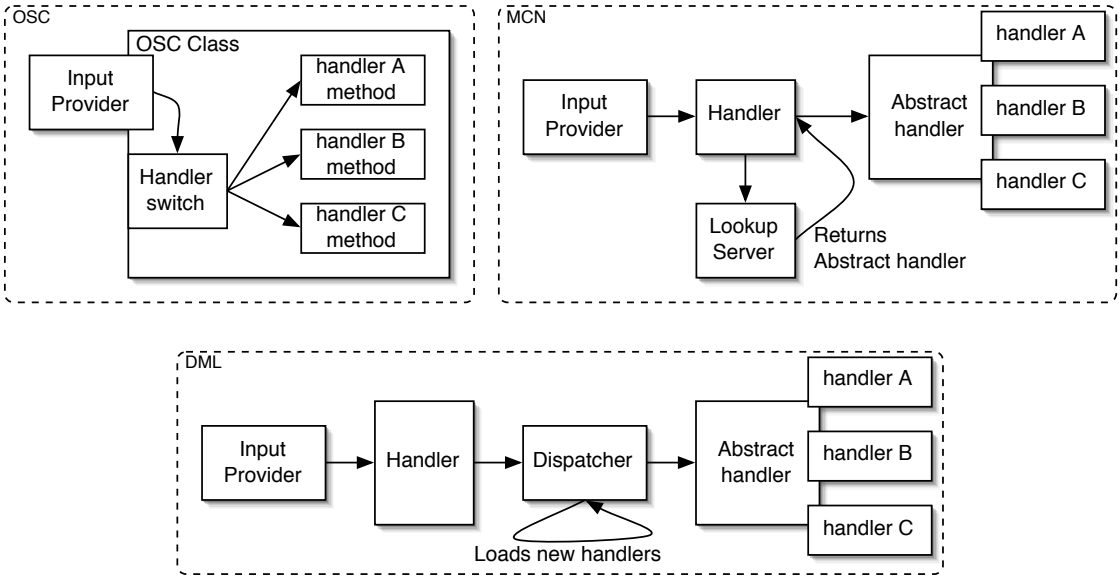


Figure 5.1: Component comparison of the three systems (OSC, MCN and DML).

Taking this approach the only choice that the developer has to make early on is the eventual scale of their system. If the system is going to be relatively small and only interact with a known set of sources then the OSC system will provide the necessary balance between amount of development and time.

If the system contained a number of unknowns then one of the other two systems is a better approach. This would be the case in a mobile situation where a handheld is discovering numerous unknown sensors and devices that it wishes to establish communication. While a desktop application may need a large number of new sensor sources. Without having some idea of the scale of requested or queried data streams it is important to be able to scale the system to cope.

Having a large number of requested streams and a large number of unknown devices could occur in any given system which is why, by taking a modular approach to building the architecture the functionality of both systems can be leveraged.

Table 5.11 highlights the limited abilities of the OSC platform. It is of note though that the DML system is fairly well suited to small system as well as more dynamic environments. This is due to the small development overhead in implementing a simple dispatcher. Larger

systems that require a variety of sources are naturally more suited to the MCN which can aggregate and locate the sensor data that the client is requesting.

Platform	Small Stable System	Numerous Queries	Numerous Devices
OSC	×		
MCN		×	
DML	×		×

Table 5.11: A matrix to help decide what system to choose.

The MCN's advantage comes through when the client applications start to make larger demands as the component network can adjust the data flow to better meet the client and the networks needs.

5.7 Architecture Choice

The three systems were built to perform the same overall task. The different approaches used in developing the case studies reflect the type of architecture that was used in each case. Each of these architectures has strengths and weaknesses and the developer needs to be aware of the trade-offs that are inherent with each design if their architecture is to benefit from the design. The following three sections explain the circumstances under which each of the three architectures presented might be appropriate.

5.7.1 Data Coupled

The Co-Visiting system presented in chapter 2.1 is a quick and easy approach. The requirements of the OSC proxy were very simple: transfer the data from the handheld to the datastore. As such, the proxy performed that with relative ease, by using a single object to decode the incoming data packets and perform a small amount of processing before sending it on to the datastore. This resulted in a small and compact code base.

The downside to the use of a single class to perform the processing was that any changes to the protocol required the class to be examined closely to find the section that required change. If the position data had been changed, the developer would have had to trace through a number of methods to find out which one performed which transformation. It is because of this reliance on the data format that this approach has been categorised as 'data coupled'. The system relies entirely on the format of the data to drive its processing.

This low flexibility and lack of runtime adaptability are the trade-offs for this type of monolithic design. While it may work for small projects or defined interfaces, as changes to the protocols will require minimal change, provided the underlying data remains unchanged. If the structure of incoming data is modified then this will result in a greater quantity of work for the developer, as the usage of each data packet must be located and modified.

The benefits of ease of addition that this approach yields the developer can be incorporated in other designs that give the developer an easier maintenance task.

5.7.2 Data Un-coupled

The MCN system was almost the complete opposite of the OSC system. The low runtime-adaptability and data coupling were replaced with a highly adaptable and uncoupled approach. This approach was also used to investigate the feasibility of adapting requests from clients to match the currently available data sources.

As described previously, this system has a much greater object count. The two objects from the OSC system were replaced with a new object for each of its sub-tasks. This allows the developer to easily see what processing can be done on each data item. Placing the processing into its own class allows updates to affect all transformations that use it. There is a marked increase in flexibility compared to the OSC system as all the processors are clearly visible to the developer so there is no replication of code as there was originally.

This approach creates a highly dynamic solution as processors can be connected at runtime. However, the dynamism comes at the cost of size. This is the largest of the three systems as there is a lot of core code to support the flexibility. The complexity of adding a processor has been minimised and so keeps this version in line with the original. The major drawback to this approach, much like that of Solar from Dartmouth is the reliance on a centralised controlling process. This process handles connections between objects and serves as a lookup mechanism.

It is possible to move this control functionality into the individual processors as demonstrated within the GRUMPS project [EAB+02]. This would result in a highly distributable and adaptable architecture. The only downside to using such an architecture is the overall complexity may pose a maintenance problem. A better solution for small to medium projects would be the use a naming convention.

5.7.3 Naming Conventions

The previous two systems were described by their relationship to the data. This system is only partially reliant on the incoming data. As described in the DML Case Study, section 4.8, this system operates with a single dispatching class handing the incoming data packets to individual processors. It works by understanding the naming convention built into the first few characters of the data packet; this is where the binding is.

While this approach is not a flexible as the uncoupled approach above, it does yield a system that is highly adaptable at run time. This allows new processors to be loaded and used as required, making it ideally suited to the City project. The small size of the code base reduces the maintenance task. However, some replication of code may be required depending on the processing requirements of the incoming data, compared to the uncoupled approach that allows each transformation to be separated and reused where needed.

This solution is ideally suited to small-medium systems as it provides the architecture with a flexible system for accessing various resources. Provided that the protocols do not vary significantly, the maintenance can be limited to the processing classes.

The next chapter shall describe the limitations and future work areas, along with the achievements of this work.

Chapter 6

Conclusion

In this final chapter the discussions throughout this dissertation will be concluded. To begin with the aims and objectives will be re-examined, the chosen approach and alternatives analysed. At this point there should be a clear understanding of what the approach can do and so the limitations of the system should be apparent. The dissertation will end with a look at the achievements of this work, a brief look at some possible future work and some concluding remarks.

6.1 Aims and Objectives

The aim of this dissertation is to describe a number of approaches for integrating dynamic management facilities into a context-aware architecture without requiring the end-client to have explicit knowledge of the management. Integration should also improve the resilience to change of the integrated system.

The integration of dynamic management facilities important to Equator's City Project due to the rapid prototyping of clients and changes in sensing hardware. The existing system was a burden to maintain for the developers as such a new approach was required. The analysis of the architectures was intended to lead to the recommendation of a new architecture choice for the City projects middleware.

6.2 Critical analysis of approach

The three systems used in this dissertation were each an evolution on the previous. The original (OSC) system that was in use was augmented with a component architecture to form the MCN. The advantage of doing this was the separation of the processing. As each new data value was sent from the sensor pack it could be demultiplexed so the processor only had to deal with one type of data packet; GPS, Compass, Location, etc. While this was advantageous as the code base became easier to understand and extend. There was a large penalty in supporting framework. This "managing software" introduced a single point of failure. If this managing process crashed then no new streams could be serviced. Existing streams would still operate however the additional point of failure in the system demonstrates that there is more to flexibility than the 'right' decisions. The additional risk

of a managing process to maximise flexibility may not increase overall flexibility. That is why the DML approach is a better representative of a flexible system. Its lack of external controlling processes simplifies the system so reduces the risk of failure present in the MCN.

6.2.1 Alternative approach

An alternative approach would have been to compare standard design patterns for flexible traits. The problem with this approach, while being able to cover a much larger range of architectures, is that the City project needed a new approach to help them with their development. The selection of a new architecture through pattern analysis would have hindered the project that needed additional functionality immediately.

6.3 Limitations

This dissertation focused on the flexibility of the developed architectures. However, the dissertation does not address issues of performance, scalability, and security and privacy. The performance of the architectures was not evaluated as all of the systems were capable of providing sufficient throughput that would cause the Equip dataspace to throttle its connection.

The question of scalability is more interesting as each system would have a different response to large amounts of data and therefore handlers. The OSC would become an unwieldy class with a lot of methods. The MCN may require such a large number of transformers that the message passing consumes all the processing capacity. The DML would be the most interesting one to analyse as it would only require instantiating a handler for each type of new data stream that was discovered. Further work would be required to understand the full implications of scale on each of these systems. While security and privacy was not a focus of this research it was of interest and as such a number of approaches to dealing with the issues were investigated and are presented in appendix 6.5.

This work is based upon the two case studies presented in chapter 4. These systems represent two different class of system, the Co-Visiting system is a centralised client server approach, so all the clients must be connected to the server to function. The Mobile system can operate in the same client server way but its real benefit is in its peer-to-peer operation. Each client contains all the services for disconnected operation. This allows each client to exchange information without the requirement of a centralised server. While these two systems were developed with a focus towards the Equator City project they also have applicability outside of that setting.

6.4 Achievements

This work has achieved its goal of showing the architecture choice will affect a designers ability to create a flexible context aware system that maximises, data management and

content delivery. Using a small hard coded approach won't scale while taking a larger framework based approach can introduce multiple points of failure. The City project used the MCN system during a number of the trials in the Lighthouse project during which time the software worked without a problem. It also gave the developers useful feed back about the data flowing through the system as each sensor stream could be monitored individually. The contribution offered to the community by this dissertation is an understanding of the various considerations in choosing an architecture for context sensitive system development.

6.5 Future Work

The major issue that is left as future work is that of security and privacy they were not of addressed as they were not of interest to the City project. Ensuring the security and privacy of the data that these systems have access to should be considered at the start of the development cycle. Adding forms of controls on after the system has been built can be difficult. Solar from Dartmouth managed to do this however the interaction of data within the system may still result in 'data leakage'. While protected information may not be made available its affect on an unprotected data value may be evident. While work in this area was beyond the scope of this dissertation appendix B describes possible directions this work could take.

6.6 Conclusions

This dissertation has shown the effects of three different approaches to handling incoming data to a system. The three approaches all have their merits in terms of the aims in section 1.2. However, when compared to the thesis statement, the solution must improve flexibility and maintain the qualities of the original system. While the uncoupled data approach, presented in section 5.7, provides the greatest level of flexibility, the overhead of maintaining the software would be too much for a small project. The small code base and minimal effort required to code a processor resulted in a suitable replacement for the OSC system, the DML system.

It is important when designing the architecture to understand the level of development within the system. If the message protocol has not yet been clearly defined it would be best to work with an input system that can be easily modified as development continues, such as the MCN or DML. In addition to the maturity of the protocols, the level of scale required both in terms of data throughput and distribution must be clearly identified. If these cannot be defined before hand then ensuring that each component as shown in figure 5.1 should be uncoupled as possible to ensure that it can be replaced with the minimum of impact to the rest of the system.

Bibliography

[ADOB97] Gregory D. Abowd, Anind K. Dey, Robert Orr, and Jason A. Brotherton. Context-Awareness in Wearable and Ubiquitous Computing. In *ISWC*, pages 179–180, 1997.

[AK96] C. J. Alpert and A. B. Kahng. A General Framework for Vertex Orderings, With Applications to Netlist Clustering. Technical report, UCLA, 1996.

[AM00] Gregory D. Abowd and Elizabeth D. Mynatt. Charting Past, Present, and Future Research in Ubiquitous Computing. *ACM Transactions on Computer-Human Interaction*, 7(1):29–58, 2000.

[BSK+03] Steve Benford, Holger Schnadelbach, Boriana Koleva, Bill Gaver, Albrecht Schmidt, Andy Boucher, Anthony Steed, Rob Anastasi, Chris Greenhalgh, Tom Rodden, and Hans Gellersen. Sensible, sensable and desirable: a framework for designing physical interfaces. Technical Report Equator-03-003, Equator, February 2003.

[CK00] Guanling Chen and David Kotz. A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000-381, Department of Computer Science, Dartmouth College, November 2000.

[CK01a] Guanling Chen and David Kotz. Solar: Towards a Flexible and Scalable Data-Fusion Infrastructure for Ubiquitous Computing. In *UbiTools Worksship at UbiComp*, 2001.

[CK01b] Guanling Chen and David Kotz. Supporting Adaptive Ubiquitous Applications with the SOLAR System. Technical Report TR2001-397, Department of Computer Science, Dartmouth College, May 2001.

[CK03] Guanling Chen and David Kotz. Context-aware resource discovery. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, March 2003. Accepted for publication.

[CNBH+02] Carolina Cruz-Neira, Allen Bierbaum, Patrick Hartling, Christopher Just, and Kevin Meinert. Vr juggler an open source platform for virtual reality applications. In *40th AIAA Aerospace Sciences Meeting and Exhibit 2002*, January 2002.

[CPT+] Norman H. Cohen, Apratim Purakayastha, John Turek, Luke Wong, and Danny Yeh. Challenges in Flexible Aggregation of Pervasive Data.

[DAPW97] Anind K. Dey, Gregory D. Abowd, Mike Pinkerton, and Andrew Wood. CyberDesk: A Framework for Providing Self-Integrating Ubiquitous Software Services. In *ACM Symposium on User Interface Software and Technology*, pages 75–76, 1997.

[DAS01] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction*, 16:97–166, 2001.

[DTMR01] Mark J. Weal Danius T. Michaelides, David E. Millard and David De Roure. Auld Leaky: A Contextual Open Hypermedia Link Server. Technical report, Dept. of Electronics and Computer Science, University of Southampton, 2001.

[EAB+02] Huw Evans, Malcolm Atkinson, Margaret Brown, Julie Cargill, Murray Crease, Steve Draper, Phil Gray, and Richard Thomas. The Evolution of the GRUMPS Architecture. *Software — Practice and Experience*, 2002.

[ES98] Maria R. Ebling and M. Satyanarayanan. On the Importance of Translucence for Mobile Computing. Technical report, Carnegie Mellon University, 1998.

[FSBJ97] E. Ferrari, P. Samarati, E. Bertino, and S. Jajodia. Providing flexibility in information flow control for object-oriented systems. In *IEEE Symposium on Security and Privacy*, pages 130–140, May 1997.

[GP96] T. R. G. Green and Marian Petre. Usability analysis of visual programming environments: A 'cognitive dimensions' framework. *Journal of Visual Languages and Computing*, 7(2):131–174, 1996.

[Gre02] Chris Greenhalgh. EQUIP: a Software Platform for Distributed Interactive Systems . Technical Report Equator-02-022, University of Nottingham, MRL, September 2002.

[GS01] Phil Gray and Daniel Salber. Modelling and Using Sensed Context Information in the design of Interactive Applications. In *ECHI*, 2001.

[HL01] Jason I. Hong and James A. Landay. An Infrastructure Approach to Context-Aware Computing. *Human-Computer Interaction*, 16:287–303, 2001.

[JHL02] X. Jiang, J.I. Hong, and J.A. Landay. Approximate information flows: Socially-based modeling of privacy in ubiquitous computing. In *Ubicomp 2002: Ubiquitous Computing*, pages 176–193. Springer-Verlag, 2002.

[JL02] Xiaodong Jiang and James A. Landay. Modeling Privacy Control in Context-Aware Systems. In *IEEE Pervasive Computing*, 2002.

[Lan02] Marc Langheinrich. A privacy awareness system for ubiquitous computing environments. In G. Borriello and L.E. Homquiest, editors, *UbiComp 2002*, pages 237–245. Springer-Verlag, 2002.

[Lev00] Matthew S. Levine. Fast Randomized Algorithms for Computing Minimum 3,4,5,6-Way Cuts, 2000.

[LRvV99] Nico Lassing, Daan Rijsenbrij, and Hans van Vliet. The Goal of Software Architecture Analysis: Confidence Building or Risk Assessment. In *First BeNeLux conference on Software Architecture*, 1999.

[McD99] A. Bruce McDonald. *A Mobility-Based Framework for Adaptive Dynamic Cluster-Based Hybrid Routing in Wireless Ad-Hoc Networks.* PhD thesis, University of Pittsburgh, 1999.

[MCIHB99] Karl S. Mathias, James H. Cross II, T. Dean Hendrix, and Larry A. Barowski. The role of software measures and metrics in studies of program comprehension. In *Proceedings of the 37th annual Southeast regional conference (CD-ROM)*, page 13. ACM Press, 1999.

[MCRS02] Ian MacColl, Matthew Chalmers, Yvonne Rogers, and Hilary Smith. Seamful ubiquity: Beyond seamless integration. Technical Report Equator-02-020, Equator, September 2002.

[MK02] Kazuhiro Minami and David Kotz. Controlling access to pervasive information. Technical Report TR2002-422, DARTMOUTH, February 2002. `http://www.cs.dartmouth.edu/~solar/`.

[ML97] Andrew C. Myers and Barbara Liskov. A decentralized model for information flow control. In *ACM Symposium on Operation Systems Principles*, 1997.

[MMRS02] Ian MacColl, Dave Millard, Cliff Randell, and Anthony Steed. Shared visiting in equator city. Technical Report Equator-02-021, Equator, September 2002.

[MNCK98] S. Mitchell, H. Naguib, G. Coulouris, and T. Kindberg. Dynamically Reconfiguring Multimedia Components: A Modelbased Approach, 1998.

[NSE+02] M.W. Newman, J.Z. Sedivy, W.K. Edwards, T. Smith, K. Marcelo, C.M. Neuwirth, J.I. Hong, and S. Izadi. Designing for Serendipity: Supporting End-User Configuration of Ubiquitous Computing Environments. In *Designing Interactive Systems*, 2002.

[RCC02] Gaëtan Rey, Joëelle Coutaz, and James L. Crowley. The contextor:a computational model for contextual information., 2002.

[RM01] Cliff Randell and Henk Muller. Low cost indoor positioning system. In Gregory D. Abowd, editor, *Ubicomp 2001: Ubiquitous Computing*, pages 42–48. Springer-Verlag, September 2001.

[SBC+98] R. Strom, G. Banavar, T. Chandra, M. Kaplan, K. Miller, B. Mukherjee, D. Sturman, and M. Ward. Gryphon: An information flow based approach to message brokering, 1998.

[SBG99] Albrecht Schmidt, Michael Beigl, and Hans-W. Gellersen. There is more to context than location. *Computers and Graphics*, 23(6):893–901, 1999.

[SDA99] Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In *CHI*, pages 434–441, 1999.

[Seg93] Bill Segall. Elvin, content based messaging. `http://elvin.dstc.edu.au/` last checked 2004-09-30., 1993.

[SKF+01] Holger Schnädelbach, Boriana Koleva, Martin Flintham, Mike Fraser, and Paul Chandler. The augurscope: A mixed reality interface for outdoors. In CHI Programme Chairs, editor, *Proc. CHI 2002*, pages 1–8. ACM Press, April 2001.

[SS00] K. Scribner and M.C. Stiver. Understanding SOAP: The Authoritative Solution, January 2000.

[Wal98] Jim Waldo. Jini architecture overview. Technical report, Sun Microsystems Laboratories, July 1998.

[WHFG92] Roy Want, Andy Hopper, Veronica Falcao, and Jon Gibbons. The Active Badge Location System. Technical Report 92.1, Olivetti, ORL, 24a Trumpington Street, Cambridge CB2 1QA, 1992.

[WJH97] A. Ward, A. Jones, and A. Hopper. A New Location Technique for the Active Office, 1997.

[WPD+02] Roy Want, Trevor Pering, Gunner Danneels, Muthu Kumar, Murali Sundar, and John Light. The Personal Server: Changing the Way We Think about Ubiquitous Computing. In G. Borriello and L.E. Homquiest, editors, *UbiComp 2002*, pages 194–209. Springer-Verlag, 2002.

Appendix A

Code Analysis

In this appendix the code classes are listed showing which metric they were involved in generating. A $\times$ represents a total inclusion in the generation of the metric while a $\otimes$ is used for partial inclusion.

A.1 OSC System

Class	Total	Procssing	Adding
CityProxy.main	$\times$		
CityProxy.CityProxy	$\times$		
MessageDaemon.MessageDaemon	$\times$		
MessageDaemon.run	$\times$		
MessageHandler.MessageHandler	$\times$	$\otimes$	$\otimes$
MessageHandler.run	$\times$	$\times$	
MessageHandler.handleGPS	$\times$		
MessageHandler.handleLocation	$\times$		
MessageHandler.handleUser	$\times$		
MessageHandler.handleSituation	$\times$		
MessageHandler.update3DXYZH	$\times$		
MessageHandler.handlePosition	$\times$	$\times$	$\times$
MessageHandler.update3DXYZ	$\times$		
MessageHandler.updatePosXYZ	$\times$	$\times$	$\times$
MessageHandler.updateO3DXYZ	$\times$		

Table A.1: List of OSC components used in each metric derivation.

A.2 MCN System

Class	Total	Procssing	Adding
CDecode	×		
CityProxyBootstrap	×		
GDecode	×		
PBound	×		
PCTrans	×		
PDecode	×	×	×
PInterpolate	×		
Position	×	×	×
SBound	×		
SDecode	×		
SInterpolate	×		
Situation	×		
SocketDaemon	×		
SocketHandler	×	×	⊗
User	×	×	×

Table A.2: List of MCN components used in each metric derivation.

A.3 DML System

Class	Total	Procssing	Adding
InputGizmo	×		
InputProxyGadget	×	⊗	
inputproxy.InputProxyHandler	×		
inputproxy.InputProxyHandlerCompass	×		
inputproxy.InputProxyHandlerDefault	×		
inputproxy.InputProxyHandlerGPS	×		
inputproxy.InputProxyHandlerLocation	×		
inputproxy.InputProxyHandlerPosition	×	×	×
inputproxy.InputProxyHandlerTimestamp	×		
inputproxy.UnknownProxyHandlerException	×		

Table A.3: List of DML components used in each metric derivation.

Appendix B

Future Work

B.1 Managing Sensed Data

In this future work section I shall discus aspects of managing sensed data that have not been raised so far. These aspects focus on the security of the data. These context systems will be generating large amounts of personal data that the owner may wish to control. There may be several reasons for controlling the data in addition to the protection of privacy. The data being gathered or used on the user's device may travel over a network that costs the user money and so may need to be limited.

B.1.1 Security

The systems developed as part of the Equator City project have a number of areas that require securing: 1) The transport mechanism: communication between components would have to be encrypted to prevent eavesdropping; 2) A form of security will be required at the infrastructure level to ensure components cannot be placed or replaced without authentication; 3) Once these components have been deployed it is important to ensure that they have not been modified at run time. 4) Ensuring the security of the data and the software components is only part of the problem. To be sure that the data has not been compromised a secure hardware platform would be required. However, this is beyond the scope of this document.

Information Flow Theory

Security models have two goals: preventing accidental or malicious destruction of information, and controlling the release and propagation of that information. Access control methods can handle the first goal but the second goal is much harder to achieve. Information flow models and theory on information propagation highlight methods by which the information can be controlled. However, these can unduly restrict the computation that can be performed. The approach presented by Myers [ML97] is based on code level control, checking the integrity of data reads and writes. The system works by labelling communication channels and variable slots with owner user identifiers as a form of access control list. By allowing these to be checked at compile time, the security of the data can

be ensured without any performance penalty. The system also allows provision for checks to be carried out at run time however checking every data packet against the ACLs will incur a performance hit.

By analysing the labelled channels, information leakage can be proved not to occur at run time. This analysis can be partially performed statically at compile time. However, the dynamic run time analysis may yield information leakage. A failure at compile time can only give information about the execution of the program as no data is currently in the program. If a failure occurs at runtime then information leakage may be based upon the data that the current channels hold. A program could use this style of attack to reason about the information that the channels hold.

Agent Security

Context-Fabric [HL01] is a component-based infrastructure that focuses on information delivery to meet the client's requests using automatic path creation and proximity-based discovery. The infrastructure is only beginning to address the issue of security and privacy.

Jiang, Hong and Landay [JL02, JHL02] suggest a model for increased privacy and security based on additional metadata accompanying the data. The model is spatially based, allowing physical spaces to be defined as areas within which objects can have control applied. This spatial model of control applies to both physical and digital objects.

Through the use of 'Information Spaces', an owner can specify the spatial extent to which they wish their objects to be limited. Should a physical device be removed from a specified space, without permission, then the owner would be notified of that object's removal. For example, if someone removes your laptop from your room then you will be sent notification to that effect. The drawback is that at least the boundaries of the physical space must be visible to the system. The devices must also be tagged in such a way that the system can detect the crossing of the spatial boundary, such as using RF tags. This level of control is naturally much less than can be imposed on a digital object. If each device that is to receive a digitally tagged object contains software to control the receipt of these special objects then the control of the objects can be enforced.

Each item (physical or digital) is tagged with a privacy tag. These include a space handle, a privacy policy and a privacy property list. The first two of these define the spatial extent and usage information, the property list contains values, such as object lifetime. This, along with the space handle, can be used to remove digital information from a user's device. For example, if presentation slides were given to a user for reference during the presentation then at the end of the presentation the slides could be deleted. Alternatively, when the user leaves the room the detection of a space change would result in the slides being deleted.

The model requires that each client install trusted control software to ensure that the privacy policies are carried out as requested. The model also allows the system to be set up with a centralised server that can verify access for applications to the data based on the stored privacy tags.

While this does require the trust of all the components in the architecture or a central

authorising server, it does not prevent the visibility of the existence of data in the system. By allowing a user to know that a certain piece of data exists is a security leak in itself even if the user cannot access the data.

B.1.2 Privacy

The pawS (privacy awareness System) [Lan02] begins to examine some of the issues that are becoming more important in the context-sensitive area. Their focus is the issues of how to deal with sensing systems over which you have no direct control. The six general principles and requirements that are laid out are: notice, choice and consent, proximity and locality, anonymity and pseudonymity, security and access, and recourse. Their approach is a policy driven solution using P3P, "Platform for Privacy Preferences Project"[1]. Using this XML[2] based policy language as a means for announcing the collector's intent with the data consequently provides the user with a policy of how their data will be stored. Although trust is still a big issue in such a system, it is not the focus of the work as the openness of the policies should allow self-regulation of the systems.

The P3P standard comes from the W3C[3] and as such is focused towards Web-based applications. However, through the use of extensions to the standard, the policies can be adapted to the ubiquitous computing environment. The role of the handheld in the paw system is to configure the current setup of the online privacy proxy which handles requests on behalf of the user. The centre of the implementation would be the privacy database that stores the data along with the user's privacy policy. This is used to ensure that the data is handled according to the wishes of the owner. If the user was not online and updating their values then the privacy proxy can relay the older value if that is the wish of the owner. The advantage to this approach is that the updated data is sent from the device only once any online request for the data can be satisfied online and so reduces the amount of network traffic require for such a system.

No guarantee is made about what happens to the data after the request. By wrapping the data and policy the data request would be able to be authenticated or approved before the access could take place by the data object. That way the owner would be able to verify the recipient's intentions with respect to the data, this idea is explored more in section B.1.3. This is idea is different from the work in the paper as that system works by simply attempting to meet the policy requirements of individual users and system announcements.

Services are discovered implicitly by an actively searching the handheld or by announcement from the service as the handheld enters the area. When discovered the proxy sends the user's privacy specification; this in turn adjusts the working of the device. For example, disabling a camera or attaching a data usage policy with the outgoing data.

The biggest problem that is evident from the description of the P3P specification is that it is only machine-readable. Users either require software or a third party to setup

[1] `http:/www.p3p.org/`
[2] eXtensible Markup Language. `http:/www.xml.org/`
[3] WWW Consortium. `http:/www.w3c.org/`

an initial policy for them to adjust. This leaves the opportunity for someone to adjust the policies slightly to give them a backdoor access to the data.

B.1.3 Data Access Control Strategies

This section outlines a data access control strategies proposed by me to allow increased flexibility in access control than can currently be obtained.

The biggest problem is controlling access to data once it has left the owners device. A similar proposal to the one presented here suggested a policy based approach to access control[FSBJ97] such that a write could not take place after a privileged read had occurred.

In most cases the recipient of data will be trusted not to divulge the information that has been given to them. However, occasionally they may use the data in a way the owner would not authorise. As such, the owner would like to ensure that their information is treated as they would like. This includes data sharing, where some information that a receiver has is given to a third party. In fact, any change in context should trigger a re-negotiation with the data owner (or representative) to use their data in the new context, such as displaying their position on a larger public access display.

Data Control

An access-oriented paradigm is used in the LOOPS project at PARC[4]. If an Active Values is changed then it will have its associated change methods triggered. I propose to adjust this approach so that an access to an instance variable causes a procedure to be triggered. The first step towards achieving this is to wrap the instance variable as an access controllable object. This object-oriented paradigm allows the access to the datum to be controlled via code the data owner can deploy as shown in figure B.1.

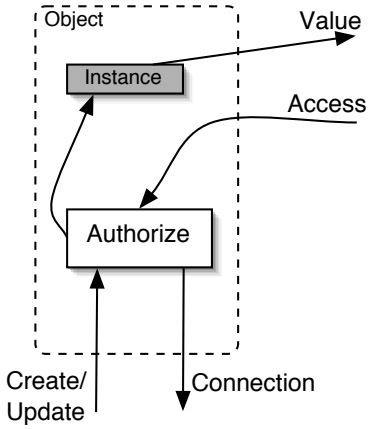


Figure B.1: Controlling access to instance values.

The control process involves the amalgamation of several properties. The data object would contain an Access Policy, the instance value and a connection method back to

[4]http://www2.parc.com/istl/members/stefik/loops.html

the data source, such as a URL. Additional properties may also be included, such as a description of the data and deployment information.

The local access policy specifies how to authenticate requests for the data. There are two states here, when the owner is connected or disconnected. When connected the user could be asked to allow access however this would have to be done in a non-disruptive fashion[ES98]. The disconnected operation may contain authentication capabilities for set situations; such as only allowing access to the value for 2 minutes after the updates stop arriving. It may be prudent to include the current data value should a connection to the source be unavailable. The final property is the method of connecting to the data source. As suggested, this may be simple HTTP or a more complicated socket-based protocol.

Additional properties may provide further information on how the infrastructure should handle the object. This will be of importance if the main contents are encrypted to protect them from the infrastructure. For example, providing a description may allow the infrastructure to perform billing or refuse transportation. However, these can only work if the description is required and adheres to a standard. The other suggested property is deployment information. If the data producer is mobile then they may wish to leave a more permanently connected handler to allow operation when they are disconnected. As such, the property would cause a property aware infrastructure to create an online component that would be used to authenticate data and access requests on behalf of the data producer. One way in which this type of component may be created is to use the data control approach above.

Event Control

Controlling the event flow is the most heavyweight (with respect to system resource usage) solution to ensure the control of the information that is being sent. This is because each event is sent with controlling code to restrict access to the data. The operation of event control would be similar to data control. By including code that can provide authorisation or notification of access with the event message, when a component attempts to access the information, the code can be executed. This code may execute at each component in a system, not only slowing down the message transit but also requiring compute cycles on the component's machine. As shown in figure B.2, requests for publicly available data would be allowed to access that information but data deemed sensitive would be controlled by the enclosed code. That data packet may also have additional access control, separate to the event access.

The wrapping of events can be done by the publishing component that sends data from the owner's device. By having a single component responsible for sending data from the device, it can be ensured that no data escapes the device that has not had some control applied to it. As the controlling of the data involves user interaction, a tool would be required to manage the publishing component. The level of control can be set by the user for a particular type of data or allow the system some freedom to select the control based on the context. There may exist some data, perhaps system data, that the user may simply allow access to in which case this data would not need a tool. However, by

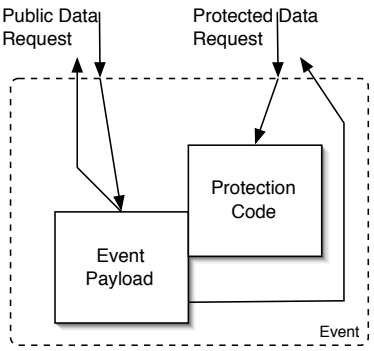


Figure B.2: Controlling access to event data.

default, all data should be protected as it leaves a device lest a user forgets to enable the protection.

B.2 Implementation

Securing the two systems with these approaches would be possible but for different reasons. The Co-Visiting system is closed system and as such securing connections is a matter of securing each endpoint. The addition of the MCN would require securing each of the components as well as the endpoints. Using the ideas from information flow theory it should be possible to secure the data in transport. All the software is controlled by the data owner or friends so data loss is not quite so important. In the Mobile system securing the data after it has left the users device is much harder but is helped by the nature of the `GizmoBelt`. As the belt is based on applet technology it is easy to limit the access any code has on the system. This would allow code to be send along with data items to control their access.