Cashua: Integrating Semantics and Content-Based Networking for Context Distribution

A thesis submitted to the

University of Dublin, Trinity College

in fulfillment of the requirements for the degree of

Doctor of Philosophy

Ruaidhrí Power

Knowledge and Data Engineering Group,

Department of Computer Science,

Trinity College Dublin,

Ireland.

November 2008

Declaration

I, the undersigned, declare that this work has not previously been submitted to this or any other University, and that unless otherwise stated, it is entirely my own work.

Ruaidhrí Power

Dated: November 3, 2008

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Ruaidhrí Power

Dated: November 3, 2008

Acknowledgements

Firstly I'd like to thank my supervisors Declan O'Sullivan and Vincent Wade, whose patience and gentle persistence kept me motivated through the many revisions of this work. Their ever-present support and feedback raised its standard immeasurably.

This work simply would not have happened without Lucy - the love of my life, best friend and co-conspirator.

Thank you to my parents and brother Diarmuid, for always being available for chat and a friendly word of advice on any topic. There's nothing more important than family.

Finally, thanks to the members of KDEG, particularly Alex and Ian for their friendship and for introducing me to Café di Napoli!

Ruaidhrí Power

University of Dublin, Trinity College

November 2008

Abstract

To support a vision of ubiquitous computing where computers fade into the background, context systems must provide information to highly mobile people and devices in ubiquitous computing environments.

In these environments, each person is served by many applications hosted by many different devices. These applications will be heterogeneous, as it is unlikely that a single standard model for context will emerge. The system must support dynamic routing of context information to allow user mobility. Finally, the system must support not-yet-developed applications, with new context models.

The Cashua system uses ontology models of context for two purposes - to manage heterogeneity and to route context information over a content-based network (CBN). The CBN is used to route queries and results, but also routes mappings between context models, allowing heterogeneous applications to communicate where an appropriate mapping between models has been provided on the network.

By coping with both heterogeneity and dynamic environments, this work advances the state of the art, using a novel approach to context dissemination which is evaluated using a number of experiments and case studies.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Mark Weiser, chief technology officer at Xerox Palo Alto Research Centre (PARC), envisioned a means of enhancing the way we work with computers by "making many computers available throughout the physical environment, but making them effectively invisible to the user"[70] [71]. Weiser felt that the modern paradigm of desktop computing was preventing real work by grabbing the user's attention and was becoming a barrier to accomplishing our tasks. Instead, he advocated ubiquitous computing that would disappear from our consciousness and become a tool to help people accomplish their tasks.

One of the key goals of ubiquitous computing as defined by Weiser is that computers recede into the background, prompting researchers in this field to look to new ways of giving applications the information they need to perform their tasks without requiring human attention to give them explicit input. By giving these applications information about the environment in which they are operating, they can then adapt their operation to the current circumstances automatically.

This information can collectively be termed *context* information, as it informs applications about the context in which they are running. This information might come from the physical environment—such as the user's current location and

activity, the people who are nearby or the time of day. Also relevant might be non-physical information such as the user's contact details, affiliations, preferences and so on. With this extra information, applications can adapt to the current situation to best serve the user's needs without being a distraction from the task at hand. Many definitions of context exist[58][59][26], and these will be explored further in Section 2.

Context management systems support ubiquitous computing by delivering context information to context-aware applications. The characteristics of ubiquitous computing environments are central to the decisions made in the design of a context management system. These characteristics provide key challenges to the implementation of such a system.

Firstly, these environments are highly dynamic. They contain large numbers of devices, each of which is capable of running multiple applications simultaneously. This is due largely to the falling prices of hardware, which is moving us beyond the personal computer paradigm where one person is served by one device, to the ubiquitous computing paradigm where one person is served by many devices. Many of these devices are mobile, and will frequently change network location as their users move from space to space. Most current approaches to context management use static, centralized servers to manage context. These servers create a bottleneck to communication, and require large amounts of computing resources.

To avoid this problem, context queries and responses must be routed dynamically to where they can be answered to remove the need to rely on hard-coded or manually configured data locations. Dynamic routing allows graceful handling of the changing location of context producers and consumers, allowing routing of messages to their destination without reconfiguration of either the producer or consumer. Some context management systems have recognized this issue and have begun to adopt publish/subscribe and peer-to-peer (P2P) approaches to context. Content-Based Networking (CBN) systems are a class of publish/subscribe systems, routing messages based on the payload of the message, rather than a specific destination address. This thesis proposes content-based networking as an

appropriate method for distributing context information.

Secondly, the devices within these environments will also be heterogeneous; they will be produced by different vendors who will each have different priorities and design philosophies. Many models for context exist for different applications, but differ either in the scope of information they cover or in how the information is modelled—for example location may be measured as latitude and longitude by one system, but as the name of the room the person is in by another. Many context-aware applications also choose to design their own context model, rather than reusing existing ones. These models will also naturally evolve over time, as applications evolve and new devices and applications emerge. Due to a combination of these factors, it is unlikely that a single standardized model for context will gain widespread adoption.

To cope with this heterogeneity, context information must be modelled in a way that can be shared among heterogeneous devices and applications for easy reuse. These models also assist in application deployment, providing formal descriptions of context information which can be used to integrate a context-aware application into the environment without reconfiguring the application. Many approaches to context modelling have been explored[64] but the work in this thesis makes use of ontological models due to their formal basis and powerful expressions for describing context models.

In summary, the dynamic and heterogeneous nature of ubiquitous computing environments poses challenges to the design of a context management system. Firstly, ubiquitous computing environments have very dynamic sets of participants. Large numbers of devices are present in this space, and large numbers of these devices will enter and leave the space on a regular basis. Dynamic routing using content-based networking is proposed as a solution to this challenge.

Secondly, these devices will be highly heterogeneous, maintaining context information using different models. Ontologies are proposed as a formal means of expressing these models, allowing the context management system to reason over the available context information and overcome this heterogeneity.

This work introduces **Cashua**, a Context Awareness Service for Heterogeneous Ubiquitous computing Applications. Cashua is a context management system which demonstrates the use of content-based networking and ontology modelling of context to overcome the challenges of dynamism and heterogeneity within a ubiquitous computing environment.

1.2 Thesis Statement

This thesis proposes that an integration of content-based networking and ontology modelling of context can be used to deliver context information to devices and applications in a dynamic and heterogeneous ubiquitous computing environment.

In order to investigate this thesis, a number of objectives were derived.

- The first objective was to conduct a survey on existing approaches to context modelling and the routing of context information in context systems, to establish the state of the art in the area. This survey identified lessons learned by others when building context management systems.
- The second objective was to establish mechanisms to be used by a context management system to address the challenges of dynamism and heterogeneity in ubiquitous computing environments. A survey was undertaken to evaluate technologies for addressing these challenges. Content-based networking and ontology modelling mechanisms were selected to form the basis of the design of the Cashua system, which was then implemented.
- The final objective was to evaluate the integration of ontology and content-based networking through a set of experiments. These experiments would examine key performance metrics and investigate the feasibility of the approach.

1.3 Contribution

The first contribution of this thesis is the novel integration of content-based networking and ontology modelling to support context dissemination in distributed, heterogeneous and dynamic ubiquitous computing environments. The formal expression of context models as ontologies allows Cashua to dynamically route context information. This routing is performed using content-based networking, which supports the dynamism in the environment by updating context routing information according to the ontology models published by context producers and consumers as they become available and move within the environment. Ontology models are also used to overcome heterogeneity through the use of mappings between models. This novel integration therefore supports both dynamism and heterogeneity in the environment, unlike existing context systems which focus on one of these issues.

A second, minor contribution of this thesis is the use of content-based networking to automatically distribute the mappings used to overcome the heterogeneity of context models within ubiquitous computing environments. The use of mappings supports the development of context-aware applications independently from the environment in which they are deployed, as integrators can take heterogeneous context models and relate them to each other by composing mappings. Cashua supports automatic distribution of these ontology mappings, allowing them to be reused in different places on the network. This allows easier reuse of mappings provided by an integrator, which are sent across the network to where they are needed, without requiring manual intervention. This automatic distribution is an important aspect of supporting a dynamic ubiquitous computing network, facilitating the movement of applications around the network once the initial integration has been performed.

1.4 Technical Approach

In order to explore this thesis, the steps identified in the objectives were carried out. This consisted of designing, implementing and evaluating a context management system that would use content-based networking for routing context queries and responses, while using ontologies and ontology mappings to express heterogeneous context models.

A literature review was carried out to assess requirements identified by pioneer context management systems and to compose a requirements framework in Chapter 2. This identified key requirements for the context management such as a method for describing an application's context information, and a mechanism for discovery of resources on the network. Then the current state of the art approach to context modelling and supporting highly dynamic environments was explored, as presented in Chapter 3. Examination of the state of the art systems also motivated additional requirements for a context system. Routing of context information was also studied, focussing on solutions which would efficiently route context information to where it was needed in dynamic environments, such as Semantic Context Spaces (SCS)[33] and Rebeca[31]. An investigation of publish/subscribe systems led to the selection of content-based networking for the routing of context. An investigation of approaches for representation of context led to the selection of ontologies to express context models and the mappings between models. Ontology models for context such as SOUPA[19] and the MoGATU BDI ontologies[49] were surveyed in Section 3.2.2.

The integration of these two technologies laid the basis for the design which is explored in Chapter 4. Cashua's architecture consists of an integrated network of Context Service Nodes (CSNs), connected by a content-based network. These nodes resolve context queries on behalf of applications, by communicating over the network.

This design was implemented as the Cashua system, and the details of this implementation are discussed in Chapter 5. The CSNs which form the basis of the Cashua system are written in Java, use the Jena library for ontology processing,

and communicate with each other over an Elvin content-based network to resolve context queries.

A number of experiments were specified to evaluate the implementation, and these are presented in Chapter 6. Two pieces of existing software were made 'context-aware', querying for context information through Cashua. Experimental work evaluated the feasibility of integrating with Cashua for developers of context-aware applications. A separate experiment produced a set of test queries to evaluate the performance of the context service, and the distribution of mapping information.

Chapter 2

Background

Many attempts have been made to provide a definition of context. Early work concentrated solely on the location of users in the environment, but this definition of context was soon seen as narrow and was extended by Schilit to include a device's "location of use, the collection of nearby people and objects, as well as changes to those objects over time" [58]. This definition was later expanded by Schmidt to define context awareness as "knowledge about the user's and IT device's state, including surroundings, situation, and, to a lesser extent, location."[59]. Dey defines context as "any information that can be used to characterise the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and application themselves."[26]

These very general definitions show that almost any information could be considered context by some applications. Thus a context management system must be able to provide applications with any kind of information that is available in the environment, even information that had not been considered when the context management system was initially designed. For the purpose of this work, context is any information which a context-aware application wishes to retrieve from a data source in the environment.

In order to characterise the evolution of context management systems, we look

at the evolution of these systems from early designs to more modern and robust architectures. The design of these systems has influenced the current state of the art in context management, and they are studied here in order to establish a set of key requirements for context management systems, as identified over the last ten years. These requirements are used to propose a requirements framework in Section 2.6, which was taken into account in the design of Cashua, and will be used as part of the evaluation of state of the art systems in Section 3.3.1.

The systems chosen for examination are the Stick-e notes system[8], CoolTown[40], CyberDesk[27], EasyLiving[9] and the Context Toolkit[25]. Each of these systems was chosen to highlight a particular aspect of their design. They are all widely referenced in the literature and are therefore likely to have been very influential in the design of modern context systems. They are presented here to give a representative cross-section of the evolution of approaches from simple early systems (represented by Stick-e notes) to more advanced and general systems (represented by the Context Toolkit).

Stick-e Notes was chosen as an early context-aware system supporting a single application which provides electronic Post-it notes, and required significant manual control from the user.

CoolTown was chosen as a ubiquitous system designed to support nomadic users. Its model, based on the transmission of URLs between devices, offered services and other information to users without requiring a centralized server.

CyberDesk was chosen as a system that handled simple data from multiple applications and could choose appropriate transformations on that data.

EasyLiving was chosen as an example of a distributed system handling multiple application types, supporting heterogeneity using well-defined interfaces to services and supporting dynamic configuration changes.

Context Toolkit was chosen as a system that adopted a generalized approach to context, modelling context facets as reusable building blocks and removing the

need for knowledge of the context source to be coded into the application.

In the following sections, each system is first briefly described, an analysis of the system is presented and then key requirements arising from the system are identified.

2.1 Stick-e Notes

The Stick-e notes[8] system, developed at the University of Kent, works by attaching context information to small electronic documents. Notes are attached to a particular 'context', and when that context is satisfied the note is triggered. A simple motivational example is an electronic Post-it note. The user carries around a PDA which is able to sense its location through some means such as GPS. The user can create a note on the PDA, and the note is automatically attached to the current location by specifying that its 'context' is that location. When the user later returns to that location, the PDA will automatically display the text of the note once its context becomes active. As well as location, Stick-e notes can be attached to other contexts such as adjacency to other objects, the states of sensors or computers, or time. Arbitrary contexts can be defined for contexts that are difficult to sense, such as the user's current task. When the user manually selects one of these contexts, all the Stick-e notes in that context are activated.

More complicated contexts are represented by allowing context elements to be combined, such as with a logical AND. Stick-e documents are collections of notes, and might represent a tourist guide to a city, a set of notes relating to an activity, or a route and delivery instructions for a courier. Stick-e notes and documents are stored in a repository on the user's system for processing. However once written, they can be exchanged with other users or published on the network.

Analysis

The Stick-e notes architecture takes a centralized, rule-based approach to context information. The user must specify the relevant context for each note when it is

composed. Each PDA or other device maintains a repository of all Stick-e notes and continually compares the rules specified in the notes' contexts with the current situation.

As the size of the repository increases over time, and particularly when notes are shared with others, evaluating the notes' contexts may become a processing burden in this centralized architecture as the current context must be compared to each Stick-e note. In addition, the user might be overwhelmed by a large number of irrelevant notes triggering. The solution proposed is for the user to manually mark only some of them as 'primed'—ready to trigger when their context conditions are met. This manual specification of which notes are relevant adds a sizeable management burden to the user, although it has the hidden advantage of reducing the processing required to decide which notes to trigger.

Requirements

The Stick-e notes system allows users to select information to be displayed automatically based on their preferences. These preferences might simply be the location they are in, which allows the system to trigger the notes for that location, or by allowing the user to activate some arbitrary context to which the notes are attached. The key requirement which arises is that users (or applications working on behalf of users) should be able to specify what context information is relevant to them. This is termed *context specification*, in our requirements framework in Section 2.6.

2.2 CoolTown

The CoolTown project[40] by HP Labs aimed to make the context of people, places and things available to others by associating them with a URL. The URLs of these entities can be broadcast to their local environment via a beacon, allowing other entities to discover them without a requirement for a global lookup service. When a

sensor picks up a beacon in this way, this "physical discovery" of an entity allows the CoolTown software to retrieve the entity's webpage over HTTP, as the CoolTown developers assume ubiquitous web access. This webpage might then be annotated with links to other pages on the Internet, which the user can browse to in order to find more information. These URLs can also be 'squirted' from one device to another, or sensed by using tags attached to entities. Although CoolTown's distributed architecture requires small embedded web servers to be placed on all devices that which to share their information, gateway web servers that may run on a traditional server machine can be used for devices that do not have the capability of running a web server.

Software called a "web presence manager" is used to represent relationships between entities other than their physical location. This software stores descriptions of these entities—their URLs and attributes, including the services they provide—in an XML database. It aggregates these entities into directories to represent arbitrary relations between them such as ownership and co-location. These directories are used to annotate the web pages of entities, for example annotating the web page representing a place with the list of users who are present in that place. Communication between nomadic users is handled by using a redirector service called WebLink. When a user enters a new space, he can provide the URL of a resource for communicating with him to WebLink. When another user wishes, they click on that person's webpage and request to communicate with them, and the redirector service provides the communication URL.

Analysis

CoolTown's use of URLs allows for simple representation of the connection between entities, however the focus here is on displaying the information to the user rather than making it available to applications. In order for an application to make use of the contextual links attached to a CoolTown entity, they must understand the semantics of those links. Human users are able to interpret the information displayed on a webpage without explicitly being given those semantics, but an application will

not be able to understand the links to correctly adapt to the user's needs. While applications could be built with the semantics of those links embedded in them, no mechanism is provided for evolving the system to allow applications to make use of information encoded in new types of link without rewriting the application.

In addition, the assumption that every device is globally accessible may not be realistic, as organisations may wish to control access to entities at administrative boundaries by structuring the network topology so that direct interaction between devices cannot take place. In these scenarios, additional redirection proxies may be needed to allow users and applications to follow links to remote entities.

Requirements

CoolTown aims to extend the web to allow for nomadic mobile users. Ad-hoc access to services and location awareness are seen as key requirements for these users. The embedding of URLs into devices in the space and the transmission of URLs and commands to devices via infra-red and 802.11 radio allows users to access the services that are in their area in an ad-hoc manner. Applications are made aware of their location by detecting other devices in their locality, and the services they offer. This allows these devices to provide location-dependent services to users, rather than providing user locations to some tracking system. These requirements are termed *network communication and availability* in our requirements framework in Section 2.6.

2.3 CyberDesk

The CyberDesk project[27] aimed to enhance the integration between applications by allowing users to share context information. Inspiration for this work came from the integration features available in software suites such as office suites. These typically feature good integration between applications within the suite, but users want integration between all applications.

While the user is performing their tasks, the CyberDesk interface continually offers the user a list of services that are relevant to the user's current context. For example, when the user selects the name of a person in an e-mail message, the system can offer services from other applications such as searching for that person's name on the web or searching for that person in a contact manager. The aim here is to provide an infrastructure for self-integrating software in which the integration is driven by user actions. The term self-integrating refers to the automatic mediation between services that the CyberDesk framework provides.

Applications register a description of each 'service' that they can perform for the user with a registry. The registry maintains a list of components in the system and the interfaces that each supports. Components have the option of registering their interest in other components with the registry, and will then be notified when components join or leave the system. The Integrator uses the registry to automatically find services that can operate on the current user data. This offers benefits over traditional applications, where developers must know the specific applications they will integrate with. With this system, the matches of the service descriptions are displayed to the user via the CyberDesk interface, allowing the user to select the appropriate interaction method. Wrappers are used to integrate services which are not CyberDesk-aware into CyberDesk. The LlamaShare[52] infrastructure is used to provide routing for information requests via software called the LlamaServer between the elements of the system.

Analysis

CyberDesk provides an infrastructure for self-integrating software in which the integration is orchestrated by the user through their actions. While this model for integration is quite powerful, it is limited in requiring user input to perform each interaction. This paradigm does not allow for intelligent behaviour of applications based on sensed context, as all decisions are made by the user. This allows the user to clearly understand how the systems are interacting but requires human attention at each step, limiting the scale of functionality that can be orchestrated in this way.

CyberDesk also uses a centralized architecture, with a single Integrator and registry required by all applications to operate in CyberDesk. The LlamaShare communication mechanism also requires the centralized LlamaServer software to pass messages between applications.

The system offered the user a set of available execution paths based on their current activity, but would frequently give too many options as the Integrator simply displayed all available possibilities rather than intelligently determining which to suggest. This project showed that more full-featured adaptation would require access to more detailed context information about the user's task.

Requirements

The CyberDesk designers use the concept of a 'service wrapper' around services, which is provided by the application programmer. This wrapper provides a description of the services the software provides—the actions it can perform on certain datatypes, and the datatypes it produces as output. The wrapper also handles execution of those services when they are called.

Students at Georgia Tech wrote many of the applications for CyberDesk, and this allowed the integration of these applications into CyberDesk as their source code was available, exposing the API they used. However, the applications were integrated only by inspecting the API, not by modifying the source code. Here, the source code was only modified to add extra functionality, not for integration purposes. The authors point out that if only the API were available, and not the source code, a wrapper could still be written to take advantage of the application's existing functionality.

The Cyberdesk designers also saw the need to convert between different types of information provided by different applications. An example would be to convert a user's position from coordinates within a space to the name of a room inside a building. CyberDesk uses *type converters* to provide this functionality. The separation of type converters into Java classes allows them to be upgraded

independently of the applications in the environment, to provide new and more relevant context. This separation also allows them to be reused by different applications who wish to make use of the conversion they provide. This requirement is termed *conversion and interpretation* in our requirements framework in Section 2.6.

2.4 EasyLiving

The EasyLiving[9] project at Microsoft Research is concerned with the development of an architecture and technologies for intelligent devices which allow the dynamic integration of diverse I/O devices into a single coherent user experience. These devices include traditional keyboards, mice and displays as well as less traditional devices such as entertainment systems, lights, speakers and so on. The aggregation of these technologies into a coherent user experience is supported by the system's middleware.

This middleware is responsible for tasks such as perception of the environment to determine the context of its entities. One of their design goals is that input/output devices should not be tightly coupled to applications, but should allow the interaction mechanism to be changed without requiring change to the underlying application. This encapsulation through abstract description of the entities allows a much more reconfigurable environment.

By basing their model on independently operating standalone devices, the EasyLiving researchers have aimed for increased reliability and tolerance of failures by removing dependence on a central server. Also considered are the problems with synchronous communication and static addressing of devices in such dynamic environments. Microsoft's InConcert[9] middleware is used to provide asynchronous message passing, machine independent addressing and XML-based message protocols. Most interesting is the machine independent addressing, which utilises a naming and lookup service to register the names and locations of devices and services. When InConcert needs to route messages, it asks the lookup service

for the current location of the device.

Analysis

The EasyLiving approach to managing ubiquitous computing environments through the use of the InConcert middleware is an interesting one, addressing many functional requirements such as coping with dynamic sets of services and devices, thereby allowing components to be easily moved to new locations. The design uses many good design practices such as the encapsulation of service functionality and well-defined interfaces to services and devices to achieve these tasks.

Requirements

While some of the EasyLiving research focuses on the geometric model of ubiquitous computing environments and user and device location sensing, the designers of the InConcert platform identify a set of requirements for context middleware. The first of these is asynchronous inter-machine communication, to allow applications to communicate with multiple information providers simultaneously. This requirement is termed *network communication and availability*, as introduced earlier. Though this requires a multi-threaded programming model, it allows applications to continue working even if some of the providers fail. An asynchronous communication model is also more efficient as multiple threads can receive information in parallel, rather than communicating synchronously with one provider at a time.

The second identified requirement is the ability to allow dynamic configuration changes. Traditional distributed communication mechanisms use explicit destination addresses for their messages, and need applications to be reconfigured whenever these addresses change. These changing addresses are likely to happen when users' devices and services change location. This will happen as users move between physical spaces, and services are migrated either to follow users or as part of a load balancing mechanism.

2.5 Context Toolkit

Anind Dey's work on the Context Toolkit[25] was a seminal work in developing a reusable framework for context information. Context widgets are responsible for collecting information from the environment through the use of software or hardware-based sensors and mediate between the application and its operating environment.

Dey took inspiration from the advantages of Graphical User Interface (GUI) widgets to come up with a design for a Context Widget. GUI widgets act as reusable building blocks for developing user interfaces, reducing the effort required for interface development by mediating between the operating system and the application. They hide the specific details of the user interface interaction from the application so that the developer can concentrate on the application's specific needs. This encapsulation allows a developer to use a GUI widget without needing to know the specifics of the user interface, such as whether a mouse, trackpad or keyboard shortcuts are used.

While a GUI widget provides an application with user interface information, the context widget provides context information from its environment. They provide a layer of abstraction between the application and the sensor, which has a number of advantages. Firstly, the widget handles much of the difficult work of retrieving context. Widgets are programmed to communicate directly with the sensor in question, removing the need for knowledge of the sensor to be coded into the application. This abstraction allows the widget to be reprogrammed to use different sensors without requiring the application to be changed, and also allows multiple applications to use the same widget.

Widgets also provide context information in the form of attributes at a level of abstraction that is useful for the application, by interpreting the raw sensor information. For example, a widget with access to a user's precise location through a GPS receiver might abstract that information to the name of the area that the user is in, as this level of detail is likely to be more interesting to applications. The widget can then provide updates to the application via registered callbacks only

when the user changes location to a new area, rather than every time a new sensor reading is taken. Finally, context widgets provide a standard interface for accessing context, easing the programming burden on developers who subsequently use them.

Building Context Widgets

A context widget is composed of three components. The first of these is the list of context attributes that the widget will export to applications. The second is a list of callbacks that the application provides so that it can be notified when relevant context changes occur. The final component is the code to provide connection to the underlying sensor technology. This code is responsible for abstracting the widget's context information to the level of its published attributes. When the widget's attributes are queried, this code handles querying the sensor to retrieve the relevant data and relate it to the list of attributes. This code is also responsible for executing registered callbacks on the application when its context information changes from the application's point of view.

Context System Requirements

The requirements for a context system identified by Dey are as follows:

Context specification a method and language for developers to specify what context the application requires.

Separation of concerns and context handling a separation of how context is acquired from how it is used, allowing for ease of reuse and maintainability of the system.

Context interpretation a mechanism to raise the abstraction of context information retrieved from sensors as abstract context about people and devices is more likely to be immediately useful to applications. Dey presents the requirement that it should be possible to compose this interpretation to provide any required level of abstraction. Applications can then request the level of abstraction that they want, without needing to know whether interpretation is required or not.

Transparent distributed communications a common method of communication between sensors and applications that hides the distributed nature of the environment from the application. This removes the need for applications to communicate directly with distributed sensors.

Constant availability of context acquisition components that provide context should execute independently from the applications that use them. This allows the developer to concentrate on using the context data rather than setting up components to deliver context. It also allows for multiple applications running at different times to access the same context data.

Context storage a facility for storage of context data for historical purposes, to allow applications to analyze past context.

Resource discovery a method of discovery of sensors, to retrieve details such as the hostname and port where they are located and how to communicate with them. Handling sensor discovery allows the application to know what context is available in the environment without having to explicitly query for it.

Context Toolkit Architecture

When context widgets, aggregators and interpreters start within the context toolkit, they register themselves with the context discoverer. When a context-aware application is started, it contacts the discoverer to find the location of components that can provide the context the application is interested in. For example, an application concerned with the location of users would ask for Location components. The architecture is general in that aggregators can subscribe to widgets in the same way that applications can, to allow them to aggregate context from multiple widgets.

Applications specify a list of attributes that they are interested in to the discoverer. They can also optionally specify a set of constant values that the attributes must have for them to be relevant to the application. For example, an application interested in a particular user might specify that the user component of the attribute must have the value 'jbloggs' to be relevant. If no relevant provider for the needed

context is available, the discoverer communicates this to the application, which can then decide how to react. The application must make this decision as it may require that context to function, so may choose to terminate or wait until the discoverer says that it is available.

Interpreters are responsible for mapping between context representations and performing computation on collections of context to raise their abstraction level where needed by applications. An example of this would be converting a username to a person's real name. Aggregators are similar, but manage context about a particular entity (person, place or object). By querying that aggregator or subscribing to it via the discoverer, the application can receive information on that entity at the level of abstraction that it needs.

All Context Toolkit elements (widgets, discoverers, interpreters and aggregators) are instances of Java classes, and inherit common abilities such as network communication from the BaseObject class. Elements communicate with each other through XML messages sent over HTTP. They support messages to allow for querying of available context and callbacks, to subscribing to attributes.

Analysis

The Context Toolkit is widely regarded as a seminal work in the area of context management, providing a useful framework for the distribution of context. Its focus on the practical needs of the application developer, particularly by insulating the developer from the specifics of the sensor in question, provides many advantages. The Context Toolkit has also been tested in projects such as Georgia Tech's Aware Home[39].

The requirements identified by Dey provide good instruction for the design of a context management system. Dey provides Aggregators and Integrators to cope with application heterogeneity, which can be programmed with arbitrary code to transform widget attributes into other formats for use by applications.

One of the issues with Dey's approach is the use of a central Discoverer component

to orchestrate the use of context. This component quickly becomes the single point of failure within the system. If the Discoverer module becomes unresponsive, then applications will be left without context information. Dey allows for the federation of Discoverers, which allows registration with higher-level Discoverers to form a hierarchy. While providing an attractive mechanism for sharing context across a larger scale organisation, the higher-level discoverers would therefore become the single point of failure for inter-domain communication, though intra-domain communication could still continue as normal.

Dey approaches the diverse nature of ubiquitous computing environments by considering platform heterogeneity. The choice of XML messages over HTTP is a completely platform-neutral one and elements of the context toolkit could be written to work with almost any platform. A separate issue is application heterogeneity—the different representations of context in different applications. Applications must register the type of context they are interested in with the Discoverer. For example, an application might register interest in Location attribute on a widget. However, now the application is hard-coded as requiring Location data in a particular form, such as room number. If the environment is upgraded in future and a new location representation is used, say GPS coordinates, then an Integrator must be programmed to convert between the two representations.

2.6 Requirements Framework

In order to help evaluate how the current state of the art has evolved to meet the challenges posed by context-aware computing, the requirements identified in the work discussed in this chapter will be used as a subset of criteria for evaluating the current state of the art in Section 3.3.1.

2.6.1 Context Specification

In order for context systems to accurately distribute context information on behalf of applications, they require information about what context is relevant to the application. This poses the requirement that applications should specify (in a machine-understandable way) what context the application can provide to other applications, as well as what context it requires itself. This requirement is identified by both Stick-e notes and the Context Toolkit.

Once this information has been received, the context system can make informed decisions about where to route queries and responses. This requirement also facilitates the evolution of the system, as the context specification can be updated as applications evolve and require new sets of context. Simply by providing a new description of the application's context, the system can adapt its routing to the new configuration.

2.6.2 Communication Abstraction

The Context Toolkit identifies a need for separating how context is acquired from how it is used, in order to allow for ease of reuse and maintainability of the system. This abstraction layer assists applications and context providers in being upgraded independently of each other, as long as they maintain the same interface to the abstraction layer. This requirement was also identified in the CyberDesk project.

2.6.3 Resource Discovery

A context system should relieve the application the difficult work of discovering what context is available on the network, without having to explicitly query for this context itself. This requirement is identified by Dey's work on the Context Toolkit and in the CoolTown and CyberDesk projects. Resorce discovery provides ad-hoc access to the services in the environment. Handling resource discovery on behalf of its applications also allows the context system to distribute cached information on

available resources, thereby reducing network traffic.

2.6.4 Conversion and Interpretation

Work on both CyberDesk and the Context Toolkit provides the requirement that context be converted between different formats. This conversion takes a number of forms. Firstly, it should be possible to convert between the information formats used by different applications. This allows heterogeneity in the system, and removes the need for a global schema. Secondly, it should be possible to reuse format conversions in other applications who wish to make use of them. This is especially important as any format conversions are likely to be expensive in terms of human time and effort, so we must extract as much benefit as possible from this work.

In addition, it should be possible to raise the abstraction level of low-level context information retrieved from sensors into higher-level context information which might be more useful to different applications. This requirement is identified by the Context Toolkit.

2.6.5 Network Communication and Availability

Finally, a context system should allow for asynchronous inter-machine communication, so that applications can communicate with multiple providers simultaneously. This allows applications to receive as much information as possible, and also insulates them from context providers who may have become disconnected from the network or fail to respond. Instead of waiting for queries to those providers to time out, the application should be able to retrieve the information it needs from any other provider in the environment. This requirement is identified by the Context Toolkit.

A related requirement is that the system should support dynamic configuration changes, allowing information providers and consumers to change network location without any reconfiguration of the applications. This is a particularly fundamental

Feature	Stick-e Notes	CoolTown	CyberDesk	EasyLiving	Context Toolkit
Context Specification	Y	N	N	N	Y
Comm. Abstraction	N	N	N	N	Y
Resource Discovery.	N	Y	N	N	Y
Conversion	N	N	Y	N	Y
Network Comm./Avail.	N	N	N	Y	Y

Table 2.1: Evaluation of Background Systems Against Requirements Framework

requirement, as changing network location will be a common occurrence in pervasive computing environments.

2.7 Evaluation Of Background Systems Against Requirements Framework

This section summarizes how the systems presented in this chapter address the requirements in the requirements framework. The purpose of this comparison is to compare adoption of these requirements in background systems with adoption in state of the art systems. State of the art systems are evaluated using this requirements framework in Section 3.3.1.

Table 2.1 clearly demonstrates the importance of the Context Toolkit in the development of context systems, as the work addressed the major requirements identified by peer systems. This is emphasised in Table 3.1, which illustrates how state of the art systems have adopted these features in their own work.

2.8 Summary

The requirements identified by these systems support the assertion that dynamism and heterogeneity are key challenges to providing context in ubiquitous computing

environments. Dynamism is recognised as an issue in the resource discovery and network communications and availability requirements, and heterogeneity is an issue in context specification, conversion and interpretation. These requirements were later used to focus the design of a system to manage dynamism and heterogeneity.

Chapter 3

State of the Art

Too many context management systems exist to evaluate them all individually. Instead, the search is narrowed by excluding systems which have a different focus to this work. This chapter concentrates on systems which address the support of dynamism or heterogeneity in context delivery.

Early context management approaches such as the blackboard model considered only relatively static environments. This allowed the use of centralized servers to store context information, which in turn simplified client configuration as the 'context server' could reliably be found at a particular network location. However, difficulties occurred when the context server became unavailable, or changed network location. As the context server was used to store all available context, it became the bottleneck for all the clients who wished to connect to it. This limited the scale to which these systems could grow, as performance would quickly degrade once the number of clients passed a threshold. In contrast, state of the art context systems consider the effect of dynamism in the environment, and provide support for applications to minimize its impact.

These systems were very tightly coupled, in two ways. Firstly, they were hand-configured with the locations of context sources. This meant that either a context server queried for context from a set of manually-configured sources, or the applications queried these sources directly. Manual configuration became tedious

and error-prone as the number of context sources increased, limiting the scale to which these systems could grow. Secondly, the clients who queried for context information were hard-coded to expect the information in a particular format. When the format of the context changed, the source code of the clients would need to be updated to use the new format. State of the art systems support heterogeneity in the environment and allow applications to make use of this heterogeneous information.

In addition to the investigation and evaluation of state of the art systems in the following sections, Section 3.3.1 compares selected systems using the requirements framework presented in Section 2.6. Additional evaluation of these systems is described in Section 3.3.2. This evaluation provides a summary of the current state of the art in this area.

3.1 Context Systems Architecture

There are many approaches to providing an architecture for context-aware systems[4]. The axes along which these approaches differ will be explored in this section. By categorising these systems according to their design axes, a clear picture of the current state of the art in context systems emerges.

The first distinguishing feature of context systems is their method of access to context data. The simplest approach is used by systems which access sensor and application data directly. This approach is highly performant as the code can be optimized to retrieve context, at the sacrifice of generality.

Systems which wish to reuse context in multiple applications typically adopt a middleware approach, encapsulating the context information into a piece of software which can be reused to greatly speed up development time for future projects.

The third context access method is a form of context server, which exists on a network to serve context to clients. This approach is very common in the ubiquitous computing field, as context sources and applications are already likely to be distributed over a network. This context server can provide context to numerous

applications, saving effort particularly for resource-constrained clients. Two variants exist among the systems that use the context server approach: centralized and distributed context. These variants are explored in the following sections.

3.1.1 Centralised context

Within the systems that choose a context server approach, some choose to retrieve all context to a central server for processing. This approach means that the central server becomes a single point of failure for the system and a potential bottleneck, but allows the most complete view of the state of the system.

There are two approaches to this architecture—applications discover a central server, or a server is configured with application and device locations. Centralised approaches attempt to keep an up-to-date 'world view' which requires a large amount of resources as context information consists of large data flows, even if none of the information is required by current clients.

Context Broker Architecture

An example of a system using centralized context is the Context Broker Architecture (CoBrA)[15]. The CoBrA system was designed to support smart meeting room applications, such as EasyMeeting[20]. CoBrA is designed around the concept of a Context Broker, a networked agent for providing context information to other client agents. As part of their work on CoBrA, the authors developed the CoBrA-ONT[18] ontology which was adopted as part of the SOUPA[21] collection of ontologies.

EasyMeeting

EasyMeeting is a typical example of a context-aware application, providing intelligent management of a smart meeting room. The system provides context-aware support for helping speakers and audiences during a meeting presentation. The services that make use of CoBrA are a speech recognition service, a user-controllable presentation service, a lighting control service, a music playing

service, a user greeting service and a service that controls which webpage is displayed on subscribed user devices.

The context broker retrieves information from sensors in the environment such as the Bluetooth Sensing Agent. This agent detects the presence of Bluetooth devices in the meeting room, from which it can infer the presence of the owners of those devices. This information is combined with context knowledge in the form of CoBrA-ONT ontologies which define user profiles, list available PowerPoint documents, meeting schedules, privacy and security policies. This information is combined by the Context Broker into a model of the contextual state of the meeting room. This model is then used by the MajorDemo agent to control the different services in the environment.

In a typical meeting scenario, CoBrA will access the room's schedule to conclude that a meeting is about to take place. As the meeting participants arrive, the Bluetooth sensing agent records the presence of their Bluetooth enabled devices. Accessing the meeting information allows the context broker to determine which users perform certain roles such as speaker or distinguished guest, and greet them accordingly via the greeting service. Once all the listed key participants have arrived, the MajorDemo agent starts the meeting by stopping the background music and dimming the lights via the Music and Light Control services. When the speaker walks up to the microphone, the presentation service is invoked to display the presentation on the screen. The voice control service can then be used to control the PowerPoint presentation until the end of the meeting.

Context Broker

Central to CoBrA is an agent called the Context Broker, which is responsible for acquiring context information from the environment, reasoning about this information and using it to build an up to date shared model of context. It will attempt to resolve any inconsistent knowledge received from the environment, and share this model with applications in the environment who wish to make use of it.

The broker consists of four components, these are the context knowledge base

which handles storage of the context broker's knowledge, the context reasoning engine which reasons over this information as well as aggregating it and resolving inconsistencies, the context acquisition model which acquires information from sensors, agents and the web, and finally the privacy management module which manages the users' privacy policies and controls the sharing of their private information.

Each context-aware client in a particular location connects to the Context Broker that is responsible for that location, such a meeting room in a building. The clients are assumed to have prior knowledge about the presence of the context broker. The broker can then answer all queries for context about that particular location. However, if the client wishes to retrieve context about another location it must connect to that location's context broker directly. This requires clients to know of the existence of these brokers and to manage communication with them itself.

Centralised Architecture

The authors recognize that the centralized design of CoBrA means that the context broker is the single point of failure of the system. To alleviate this problem, a mechanism for federating context brokers is proposed. To do this, a persistent team approach is adopted as described in the Adaptive Agent Architecture[41]. This provides a mechanism for teams of context broker agents to cooperate in order to cope with the failure of broker agents by contacting client agents once the failure of one of its members is detected. The members of the broker team are responsible for maintaining a shared model of context so that each is ready to resume the duties of a failed broker. The number of broker agents in the team is maintained by spawning new agents once the number of members in the team falls below a certain threshold.

While this solution provides a level of fault tolerance to the system, it does not address the fact that a centralized context broker becomes a 'bottle neck' to the flow of context information in the system. If a context broker is to support an environment with a large number of context clients, two things will happen. Firstly, the context broker will retrieve information from a large number of sources to store in its knowledge base. This will cause considerable network traffic, and also impose

a processing burden on the context broker as it must maintain an up to date context model of the information from these sources. Secondly, the context clients in the environment must all send their queries to the broker. The broker must resolve each of these queries and return the result to the clients. This solution is perfectly adequate with a small number of clients, however it will have difficulty scaling to larger client populations as each query requires significant processing.

The addition of a federation of context servers exacerbates this problem. This is because the federation must continually synchronise their stored context knowledge across all members so that each is prepared to continue serving clients if necessary. This adds processing and network burden to each of the brokers, particularly in the case of a large context model.

Use of Ontologies

The broker uses a set of ontologies called CoBrA-ONT[17][18] as a basis for the context model of the environment. All agents share a common ontology with the context broker. This common ontology includes the vocabularies and the associated semantics used for describing contexts and modelling reasoning assumptions, and it is proposed as a common upper-level basis for expressing context in a ubiquitous computing environment.

CoBrA-ONT is expressed as OWL ontologies in RDF/XML format. The key top-level ontology describes "Person", "Place" and "Intention". The each Person is defined to have a name, an e-mail address and a homepage URL. In addition, subclasses of Person include "PersonInBuilding", "MeetingParticipant" and "Speaker", allowing the system to reason about the roles of Person individuals.

The Place class defines the naming and containment policies for places in the environment. Subclasses such as "Building" and "Room" can be annotated with "isPartOf" and "hasPartOf" properties to specify containment relations which allow the expression, for example, of the set of rooms that make up a building. The Intention class defines the notion of user's intentions, for example for the speaker to give a presentation and for the audience to receive handouts of the presentation.

As the context broker communicates with sensors to find the current state of the environment, it places this information into its knowledge base. This information is reasoned over to discover any inconsistencies, and made available to client agents for querying. Because all CoBrA agents must share a common ontology with the broker, they use this ontology to express their context information and to pose their queries. However, requiring agents to commit to the semantics of this ontology will force the agents to upgrade if the ontology changes in future as the system evolves. It also means that clients must be specifically designed to work with the ontology used by the CoBrA system. Although an ontology other than CoBrA-ONT could undoubtedly be used with CoBrA, the system does not support the use of different ontology models within a single context broker.

Policy-based privacy protection

The CoBrA system uses policies specified by the users to control access to their context information. These policies are expressed using the SOUPA policy ontology, which defines the vocabularies for describing rules that permit or forbid agents to perform actions. Actions can be a request for information, or an invokation of a particular method or service.

The decision taken to allow any action is made by classifying the action using OWL-DL reasoning to decide whether it is in the classes PERMIT, FORBID or UNDECIDED. The UNDECIDED classification results when the agent does not have enough information to answer the query. Exceptions may occur if policies conflict or do not provide enough information, and in these scenarios the agent will prompt the user or take a default action.

If the response to a query is FORBID, the granularity of the query can be adjusted, for example to give information about what building a user is in without disclosing which particular room. Once the granularity has been adjusted, the agent can re-check its permission to see if it can retrieve the less specific information.

Resolving inconsistent context

CoBrA brokers use a Prolog version of the Theorist system[53] for resolving

inconsistent context. This system takes context facts from the knowledge base and applies assumption-based reasoning.

Analysis

CoBrA allows applications to specify their own context information, however they must express it using a fixed set of ontologies. This requirement of a specific data model greatly limits the flexibility given to an application developer, requiring communication with other applications to be in terms of this upper ontology.

The system provides discovery of resources within a particular environment managed by a context broker, however it provides no mechanism for discovering any resources outside the current environment. The federation method described simply provides a level of fault-tolerance in the system, but does not provide an efficient means to access remote context.

CoBrA does provide a good communication abstraction, network communication and availability, alleviating much of the burden of context acquisition from the application. However, its requirement for a common ontology for expressing information means that no conversion between different models is required.

3.1.2 Distributed context

Other systems operate in a more distributed manner, allowing peers to communicate with each other to access the needed context information. The distributed approach can offer great benefits in terms of network and processing overhead, as peers can intelligently exchange only the information that is needed, avoiding needlessly sending large data flows across the network. However, the distributed approach is often more complicated to implement.

One of the most popular approaches to distributed context information is to leverage existing work in peer-to-peer (P2P) systems for dissemination. These P2P systems act as a distributed database which can be queried for the location of a particular

piece of information, and have attractive properties for context management.

Semantic Context Space (SCS)

Gu et. al. present their work on a peer-to-peer context information search mechanism in [33]. In contrast to traditional P2P approaches, where data placement is governed by a distributed hash function, the authors recognize that it is more efficient to store context information close to where it was generated, and where it is likely to be used.

To achieve this, they present a system called the Semantic Context Space (SCS). SCS operates as a semantic overlay network, where peers are grouped on creation into clusters according to the semantics of the data that they manage access to. When a peer receives a query, it processes it to determine which semantic cluster the query should be sent to, and forwards it appropriately. Once the query reaches the correct semantic cluster, it is flooded to all nodes within the cluster who perform the query. Each peer in SCS maintains a local RDF data repository which can be queried using RDQL. The peer can then return the results.

The authors choose to use ontologies to express the semantics of queries and results, allowing better handling of synonyms and polysemy. A two-level ontology approach is used, where an upper ontology defines common concepts understood by all peers. Each peer can then define its own concepts in lower-level ontologies, based on the needs of its applications. When a peer joins the network, it adds RDF instances to the combined upper and lower ontologies to form its local knowledge base. The predicates in this local knowledge base are then mapped into semantic clusters based on its *rdfs:range* if the predicate is an *ObjectProperty* or the most specific class if its *rdfs:domain* if it is a *DataTypeProperty*. The cluster chosen is the one which maps to the most specific concept in the upper ontology, not in the lower-level ontology.

The authors distinguish between a *cluster* which is simply a group of nodes and a *semantic cluster* which is a groups of clusters that deal with the same semantic concepts. These semantic clusters are organized in a one-dimensional ring formation.

Each peer must decide based on its ontologies the semantic cluster to which it should be assigned. In the case where a peer might fit into multiple semantic clusters, the one with the closest-matching concept is chosen. Peers maintain links to the other peers within their own cluster and to two peers in adjacent clusters, so that the clusters are joined by these links into this ring formation.

Peers then publish indexes of available information within their own cluster, however they also choose another random peer within each semantic cluster to publish their index to. The links formed by this publishing provide 'short cuts' so that peers do not need to always forward queries around the ring to their destination. A routing protocol is then used on each peer to determine how messages are routed based on available links. The short cuts are required here to avoid the need for all queries to be routed around the ring in either direction.

Analysis

SCS concentrates on the routing of context information expressed in RDF. This routing is based on the semantic clusters derived from concepts in an upper ontology. Its use of ontologies allows applications to register a description of the context that is relevant to them. However, it requires that any lower-level ontologies are related to an upper-level ontology. This is due to the classification of peers into clusters based on terms from the upper-level ontology. This two-level ontology approach is preferable to a single global ontology as it allows applications to change their context model as they evolve. However, any changes to the global ontology would require changes to applications.

Resource discovery is provided here, allowing applications to discover information on the network simply by querying their local peers, who will take responsibility for forwarding queries as necessary to their neighbours in order to find relevant information. However, this resource discovery is quite coarse-grained as it only uses concepts from the upper ontology in its routing decisions, which will cause some queries to be routed to peers containing only information about a common parent concept, rather than the concept which is being queried for.

SCS provides a communication abstraction, as applications send queries to peers, who then forward those queries on their behalf. However it does not provide a mechanism for conversion and interpretation between formats used by different applications. Ontologies are used to express the models of application data, but the reliance on a shared ontology for routing purposes requires that applications commit as much as possible to that ontology in order to achieve the most efficient routing.

Finally, SCS provides network communication between applications and constant availability of the context information registered with the network. Dynamic configuration changes are supported through a process of cluster splitting and merging in order to maintain the network of clusters, and through the use of local peers communicating on a dynamic network, context acquisition is always available to query applications.

Rebeca

Fiege et. al. from Darmstadt University of Technology present Rebeca[31], a message-oriented system that attempts to address the scalability of traditional publish-subscribe systems through the use of *scopes*. Scopes are used to control the visibility of notifications outside of application components. They are proposed as a means of adding customization and modularity to publish-subscribe systems, as well as for improving scalability and dealing with heterogeneity.

The authors wish to maintain the advantages of publish-subscribe systems, such as their loose coupling and abstraction of communication mechanisms, when they are deployed in large-scale distributed environments over multiple administrative domains. Scopes bundle a set of applications together into a group which can be managed independently. In addition, scopes can contain other scopes, allowing some scopes to act as regular producers and consumers within other scopes. A specification language is used to construct scope graphs.

In order to implement scopes, the routing table of each broker is divided into multiple

tables, allowing connected brokers in the same scope to transfer messages related to that scope. Another table, called the scope routing table, records links to brokers in scopes that this broker is connected to. Administrative commands are used to control the scopes a broker is connected to, thereby allowing a scope's overlay to be extended as required.

Rebeca considers data integration between applications, expressing the vocabulary of concepts within a scope as an ontology. When subscriptions and notifications are distributed outside a particular scope, the *scope administrator* can specify the mapping of terms between the ontologies. This allows notifications that pass across scope boundaries to be translated into the model of the remote scope, so that they can be understood by remote subscribers.

Rebeca can be configured by an administrator, supported by a plugin to the Eclipse programming environment to construct the scope graph from which the configuration of the individual brokers can be specified. The authors also address security through the use of certificates to access management functionality and encrypted inter-broker communication.

Analysis

Rebeca builds on work in the field of publish-subscribe communication, and exploits some of the features of that paradigm such as resource discovery, communication abstraction and network communication and availability.

Rebeca uses ontologies to express the information model used by a particular scope, however conversion between models is only done between different scopes, requiring all applications within a scope to use the same model to communicate. The conversion functions (or mappings) could undoubtedly be reused when new scopes are added to the system, but it is unclear whether these conversion functions could be automatically discovered by the relevant brokers.

Context Spaces

Tarlano and Kellerer present Context Spaces[66], a framework for organizing the distribution of context information. They propose an overlay network which is divided into a set of 'context spaces', each of which contains a number of 'space peers'. Space peers communicate with each other using a standardized protocol called the Application Messaging Interface (AMI). Context spaces can also contain 'space groups', to which peers can subscribe based on a particular interest. These space groups can act as containers for other objects, such as peers or other space groups.

All context entry objects within the system are of one of three types: Content Objects which record environmental state or attribute information for use by peers in that environment, Logic Objects which perform logic tasks at runtime for example to test environmental conditions or to take action based on those conditions, and finally Space Objects which contain all the context peers for a particular space, thereby categorizing them according to their interest in that space.

Space objects can be contained within other space objects, which leads to a hierarchical structure. Context objects which are part of these space objects are then located by using XPath path matching. Containment within these objects produces an "in-a" relationship hierarchy, which could be viewed as a simple ontology.

This hierarchy allows for a method of context acquisition similar to the Object-Oriented Programming model of inheritance. When a space doesn't contain the content object specified in the XPath expression, the object can be retrieved from any of the space's ancestor objects.

The testbed for Context Spaces uses the JXTA peer-to-peer protocol. JXTA allows the authors to create and loosely couple the context spaces without implementing a new P2P system. JXTA's peer groups are formed as a collection of peers that implement a common set of services, are used to allow space peers to discover and connect with each other.

Analysis

Context Spaces provides support for hierarchical modelling of pervasive computing environments. This modelling allows space groups to act as channels for context information relevant to a particular topic. Peers can connect to these space groups to discover other peers giving relevant information, and query from them. This grouping approach allows more efficient distribution of context, as peers need only query the group to discover relevant context, and do not need to flood the network with a query to be sent to all peers.

However, these context spaces are defined a priori and their JXTA identifiers must be known in advance by applications in order to make use of them. Thus, they would be very sensitive to the evolution of the system, as particular peer groups become obsolete and applications would need to be reconfigured to use the updated versions.

Another issue is the use of fixed XPath queries between the context consumer and producer. No mechanism is given for the translation of these queries to different context structures used by other applications. This results in quite an inflexible system which will be difficult to evolve as the needs of the environment change.

Fulcrum

The Fulcrum system[6] uses content-based publish-subscribe routing to forward single-schema context information.

The authors exploit the advantages of content-based networking to reduce traffic on the network. Fulcrum is built on portions of the Siena[11] content-based network and the Jabber[37] protocol. The Fulcrum application uses presence information from a Jabber client as well as location information detected by the ActiveCampus system as its context information.

By reimplementing the event brokers of the Siena content-based network, Fulcrum allows applications to specify their own method for evaluating a subscription. These 'active subscriptions' are used to allow applications to reprogram the content-based network for their own needs. The example given is to allow an application to detect

the proximity of two users. Here, an active subscription can be used to suppress location reports at an entry node until a particular time has passed, a user's rate of movement changes, or either moves half the original distance between them.

The event broker architecture consists of a primary event broker which handles normal advertisement, subscription and publication processing. It forwards events to other event brokers, or to the appropriate active subscription if one is defined. The broker also contains a wrapper which provides a miniature broker environment consisting of a message buffer, an event router and the active subscription defined by the application.

The subscription distribution algorithm propogates subscriptions across the network as in a conventional content-based network. However, the active subscriptions are only maintained for the entry nodes in the network, not for any intermediate nodes. These active subscriptions are used to suppress event information that is not significant to the application from crossing the network, thereby reducing network traffic.

To support the use case of detecting the proximity of two users to each other, these active subscriptions are used to implement 'range rings', which define a physical area that the application believes the user to be present in. Once the user leaves their 'range ring', the distance to the other user is evaluated to determine if they are close to each other. By continually re-evaluating these range rings once the user steps outside them the number of events sent across the network can be reduced.

Analysis

Fulcrum provides a means of allowing applications to reconfigure the message forwarding strategy of the network in order to reduce network traffic. These active subscriptions act as filters on the edges of the network.

However, Fulcrum provides no support for heterogeneity of applications connecting to the network. These applications are therefore limited to a fixed message schema which cannot be changed without updating the code of all clients using that schema. Independently-developed applications will not be able to make use of the context

expressed in this schema without being programmed to use it themselves.

MoGATU

Perich's MoGATU[50][51] system looks at the problem of distributing information to mobile clients in a dynamic network.

Issues addressed by MoGATU

The authors recognize that pervasive computing environments pose different challenges to those found in traditional mobile data management. For example, data management in traditional environments will assume that certain resources such as data managers will always be available in the environment, and that clients will know their locations a priori.

Other relevant issues noted by Perich include the highly dynamic set of neighbours that each device will have, the lack of a global catalog or schema, and the lack of a guarantee of either connection or collaboration between devices. The dynamic neighbourhood implies that query results will vary frequently acoording to available information in the vicinity, as these mobile devices are limited to communicating only with devices in their vicinity. The lack of a global catalog or schema means that devices do not have access to any resource that would tell them where information is located in the network, as they would in other scenarios. This is due to the large number of devices in the network and also their high mobility, which makes maintenance of an up to date global catalog difficult.

No guarantee of reconnection between devices means devices must cache any information they might subsequently need, and that only best-effort service is possible. In addition, there is no guarantee of collaboration among devices. Devices may refuse to share reliable information, or the information they share might be unreliable. When information is shared, questions regarding permission to modify and re-share that information arise.

MoGATU Architecture

The MoGATU architecture represents all devices in the environment as Information

Providers, Information Consumers, and Information Managers. *Information Providers* each hold a piece of information that is to be made available to the environment. *Information Consumers* are entities that can query and update data. These entities could be anything from humans to autonomous software agents. Finally, an *Information Manager* (InforMa) must be present on each device. Each InforMa maintains a list of nearby information and the list of devices that can supply that information, as well as a cache of recently-accessed results.

The profiles, queries and descriptions of providers are all annotated using DAML+OIL. MoGATU uses profiles to express the beliefs, desires and actions of users. The InforMa uses these profiles to decide what information it should retrieve and how it should be cached. A rule system is used to convert these beliefs and intentions into query restrictions and standing queries. The example given is that a rule might create a query to search for petrol stations when a person is driving in a car that is low on petrol. The InforMa's reasoning engine uses these profiles together with additional information such as the current time and location to generate the appropriate queries.

Routing in MoGATU

MoGATU uses a hybrid mechanism combining discovery and routing for routing queries (i.e. data discovery) among peer InforMas in multi-hop networks. The algorithm uses a source-initiated approach and attempts to rebuild disconnected routes, but does not guarantee message delivery. As query results could be received from multiple different providers, InforMa maintains cached route entries for as many providers as possible. InforMa will prefer to maintain routes for providers who are just one hop away, but will also maintain routes for multi-hop providers if they are part of a long-term interaction or the InforMa is caching advertisements for those peers. In order to reduce the traffic in the environment, solicitations and advertisements are limited to one-hop neighbours.

Analysis

The context specification in MoGATU is based on a single-model approach. Context

is specified using the MoGATU ontologies and no mechanism is given for allowing applications to specify alternate context information that they might be interested in. Similarly, issues arise when ontologies or software are upgraded, invalidating the application's commitment to the old ontology.

MoGATU's model of information providers, consumers and managers effectively provides a reusable communication abstraction with network communication and context availability. However, it does not provide any mechanism for conversion of information between different formats, a certain requirement in ubiquitous computing systems.

3.2 Heterogeneity

3.2.1 Context modelling approaches

Strang and Linnhoff-Popien survey the multitude of context modelling approaches in [64]. They classify these models into the following groups.

Key-value models are the simplest form of markup—each particular context attribute is represented as a key, and the application simply reads the value associated with that key to retrieve the result. *Markup scheme* models are hierarchical models, defining both attributes and content for each tag. Because of this, they are often expressed in XML or another SGML variant. *Graphical* modelling applies existing modelling approaches such as UML and Object-Role Modeling (ORM) to context information. The generic nature of these approaches allows them to be easily extended to include features such as dependencies between context facts.

Object-oriented models apply the traditional benefits of object-oriented software design such as encapsulation and reusability to context. Objects are used to hide the details of context acquisition, exposing a context interface at an abstraction level which is useful to an application. This allows these context objects to be upgraded in the future without affecting the applications that use them. *Logic based* models

represent context as a set of facts and concluding expressions that are true in the environment. A formal system is used to apply rules which allow additional facts and expressions to be derived.

Finally, *ontology-based* models also exploit formal models of context to express concepts and relations between them. This survey rated ontology models highly for their support for distributed composition, partial validation, richness and quality of information, support for incompleteness and ambiguity, level of formality and applicibility to existing environments.

3.2.2 Context Models

Of the published ontology models for context, clearly the most influential is SOUPA[19]. The SOUPA project began in November 2003 as part of the Semantic Web in Ubicomp Special Interest Group. The SOUPA ontologies are freely published online, and are frequently cited as a good example of ontologies for context.

SOUPA

The SOUPA ontologies consist of two sets of separate but interlinked ontologies that form SOUPA core and SOUPA extension. While SOUPA core is used to model fundamental concepts such as Person, Action, Space and Time, SOUPA extension models higher-level concepts such as Schedule, Meeting, Contact Preference and Conditional Belief.

These ontologies are designed to be used separately if required, so that application developers may choose to make use of only some of the ontologies in their application, to reduce complexity.

The designers of SOUPA elected to borrow terms from other ontologies, but not to import them directly. SOUPA references terms from a number of ontologies such as the Friend-Of-A-Friend ontology (FOAF)[7], the spatial ontologies in OpenCyc[42], COBRA-ONT[16] and the MoGATU BDI ontology[49].

The SOUPA ontologies have been used in a number of projects. For example, Fuchs et. al.[32] describe an implementation of an intelligent answering machine application whose ontology maps to the SOUPA ontology and the FOAF ontology[7], allowing interoperation with other applications which also map to these ontologies.

CoBrA-ONT

SOUPA has also been extended in the CoBrA-ONT ontology (by the same authors) to allow the CoBrA system to manage smart meeting rooms. The CoBrA-ONT[16] ontology defines some of the common relationships and attributes that are associated with people, places and activities in a pervasive computing space. The CoBrA system uses this ontology, along with instance data provided to it, to answer questions such as "Is X currently in a meeting place in building Y?" or "Is X the speaker of meeting Z?".

At the highest level, the CoBrA-ONT ontology describes "Person", "Place" and "Intention". For example, the Person class defines properties of people such as their name and e-mail address, while more specific subclasses of Person such as "Speaker" or "PersonInBuilding" are used to define additional properties such as the building the person is in. The "Intention" class defines user intentions such as the speaker's intention to give a talk. This class is defined as the union of all its subclasses, as it is the collection of all defined user intentions.

MoGATU BDI ontology

Independently from SOUPA, the MoGATU BDI ontologies[49] are used in the MoGATU system, and have a slightly different focus. The Agent ontology represents human users or intelligent software entities. Agents can express their Beliefs, Desires, Intentions and Goals. The statements that express their beliefs can be unconditional statements, which always hold true or conditional statements which are asserted if a particular condition statement is true. An agent's desires can conflict with other desires, but its goals are a set of non-conflicting achievable desires. An agent can

assert a set of intentions, the actions it will perform to achieve its goals. These actions can be combined into an ordered sequence which is called a plan.

Agents can express the priority of these desires, goals, intentions, etc. using a weighting system. The MoGATU ontologies also allow the expression of time, both time instants and time periods. This allows the agent to specify when and how often to initiate an action plan. By reasoning over the ontology information available, the agent can take intelligent action based on the current state of its environment and the other agents (human or software) taking part in it.

CoDAMoS ontology

The CoDAMoS ontology[55], as presented by Preuveneers et. al., is another generic upper-level ontology which aims to provide a basis for the most important aspects of context information. It defines a User (which contains a user's preferences, profile and current activity), Environment (containing time and location information as well as environmental conditions such as lighting), Platform (a hardware and software description of a device) and Service (software which provides a service to a user).

User tasks can be broken down into the activities that the user performs in order to accomplish the task. As well as tasks, Users have Profiles which express static information about them, Roles which express the kind of actions they perform, and Mood stores extra more dynamic information such as current communication preferences.

The Platform section of the ontology provides a description of the software that is available on the device for the user and other services to interact with, and the hardware resources of the device. This includes elements such as the software installed on the device, the operating system used, the virtual machines available to execute software and so on. Each Platform is part of an Environment. The environment specifies physical properties such as its location and environmental condition (temperature, humidity, etc.)

Some less-cited models include SOCAM and CONON.

SOCAM

Gu et al. present a Service-Oriented Context-Aware Middleware (SOCAM)[34] based on a context model with person, location, activity and computational entity (such as a device, network, application, service, etc.) as basic context concepts.

CONON

The Context Ontology (CONON)[68] is an upper-level ontology for context which is designed to be extended with domain-specific ontologies for particular tasks. Location, user and activity are taken as the most fundamental elements of context. Computational entities are also considered as first-class elements of the top-level ontology.

The authors use data modelled with this ontology to reason about facts such as the location of users.

CoOL

Finally, the Context Ontology Language (CoOL)[65] is an ontology-based context modelling approach, rather than a particular model. It uses the Aspect-Scale-Context (ASC) model where each aspect (e.g. spatial distance) can have several scales (e.g. kilometer scale or mile scale) to express some context information (e.g. 20). Mapping functions exist to convert context information from one scale to another.

3.3 Evaluation of State of the Art

In order to evaluate how the state of the art meets the needs of context distribution, the systems examined in this chapter will be compared using a number of criteria. The first set of criteria is the requirements framework presented in Section 2.6. This comparison is presented in Section 3.3.1.

Feature	CoBrA	SCS	Rebeca	C. Spaces	Fulcrum	MoGATU
Context Specification	Y	Y	Y	Y	Y	Y
Comm. Abstraction	Y	Y	Y	Y	Y	Y
Resource Discovery	N	Y	Y	Y	Y	Y
Conversion	N	N	-	N	N	N
Network Comm./Avail.	Y	Y	Y	Y	Y	Y

Table 3.1: Evaluation of State of the Art Systems Against Requirements Framework

The second set of criteria is those which are derived from examination of the state of the art systems themselves. Issues identified by the authors of these systems are identified, in combination with other issues. The evaluation against this set of criteria is presented in Section 3.3.2.

3.3.1 State of the Art Evaluation Against Requirements Framework

This section summarises how the systems described in this chapter compare using the requirements framework described in Section 2.6. This summary is shown in the form of a table in Table 3.1.

Here we see that context specification and communication abstraction are provided by all the surveyed systems. Similarly, resource discovery is provided by all systems except CoBrA. CoBrA does not provide resource discovery as it is a centralised system, so all available context resources are stored on a central server.

None of the systems surveyed support conversion between different forms of context, except for Rebeca which supports a limited form of model conversion, converting between models in different Rebeca scopes, but not within scopes. All the surveyed systems offer network communication and availability, allowing constant access to available context.

This comparison clearly identifies support for heterogeneity (through conversion between models) as lacking in the current state of the art systems. This support for heterogeneity is crucial to the adoption of systems for context-awareness, as it allows independently-developed systems to be deployed in the environment without requiring costly rewriting of software to retrieve context information.

3.3.2 Additional Requirements

In addition to the evaluation requirements from the requirements framework, this section identifies additional important requirements which have become apparent from the state of the art survey.

The first of these requirements comes from the fact is that centralised systems such as CoBrA are not scalable as they require all context information to be centrally stored and managed. Distributed context storage avoids this problem, allowing each data source to manage its own repository of context. Systems which take the approach of storing context close to where it is being used, such as SCS, will provide even better performance over randomized distribution of the data. Exploiting the localized nature of context in this way improves performance by reducing the network resources required to transfer information.

Particular consideration must be given to the humans developing context-aware applications and managing context systems. An important factor for the developer is the format used for expressing context, and the context query language chosen. A combination of RDF and RDQL is a popular choice for context systems, however this requires data sources to express their context in RDF and applications to express their queries in RDQL. This requires developer effort as RDF/RDQL is not often used in industry and developers may be unfamiliar with it. The programming effort required to refactor existing applications to use this RDF data may be considerable, if a developer wishes to make the application context-aware.

For those managing such systems, it is important to minimize the additional management tasks that they must perform to keep the system running smoothly.

Tasks such as forming and managing scopes which is part of the operation of Rebeca increase the management overhead of maintaining a context system.

Another important requirement is the separation of the role of software developer, who writes context-aware applications, and the context system manager, referred to as the integrator in this work, who deploys these applications into an environment and is responsible for their correct operation. In the systems presented in this chapter, the developer and integrator were typically the same person or worked closely together. This simplifies application deployment as changes can be made to either the application or context system in order to allow them to interoperate, but this is very unrealistic scenario. It is much more likely that the application developer and context system manager will be separate people, and the system must be designed to account for this.

Chapter 4

Design

This chapter describes the design of the Cashua system. This design addresses the challenges of dynamism and heterogeneity in ubiquitous computing environments introduced in Section 1.1, and establishes the mechanisms for context dissemination and semantic modelling.

The overall architecture is set out in Section 4.1, including an introduction to the Context Service Node (CSN). The CSN is a service which is deployed in the environment and can be used by context-aware applications to retrieve context information. CSNs communicate with each other via a content-based network to route this information. The design of the routing mechanism is laid out in Section 4.3, which describes query and response routing.

In order to cater for heterogeneous applications in the environment, Cashua provides translation between the models of these applications. Section 4.2 describes Cashua's support for heterogeneity, including its ontology-based context modelling and mapping mechanisms. Cashua's support for dynamism is described in Section 4.4, explaining how Cashua's use of content-based networking manages the constantly changing set of mobile clients in the environment.

Each of the three sections describing routing, heterogeneity and dynamism in Cashua also include descriptions of the specific CSN components which provide this support in Cashua.

Section 4.5 describes the operation of the query engine, which orchestrates the CSN's resolution of queries through query and result translation, as well as communication with the content-based network. The overall design of the CSN is described in Section 4.6, and this is illustrated by describing how an example query is resolved in Section 4.7.

Section 4.8 describes the mapping discovery and distribution mechanism, before Section 4.9 describes additional tools which were designed to assist in the operation of the CSN—the context connector and the graphical CSN manager.

4.1 Architecture

As discussed in Section 3.1, a range of possible architectures exist to implement context services. In order to select a architecture for Cashua, features of these architectures were considered in order to establish their strengths and weaknesses.

The first approach considered was a centralised, blackboard approach where context would be registered at a central server. This server would be notified on change of information and could distribute the updated context to applications whenever it was queried. However, this architecture requires that all context information be transmitted to a central server for dissemination. In a ubiquitous computing environment, context information is produced at many different nodes which are distributed across the network. Transmitting all this information to a central server would remove the need for peers to communicate with each other to resolve queries, but would require large amounts of network bandwidth to keep the information up to date. This server would also require processing power to answer all context queries, and would become the central point of failure for the context system.

A second alternative was a middleware approach to context acquisition, where each context-aware application contains an embedded library which is responsible for retrieving context on the application's behalf. This approach was also rejected as the middleware approch is not easily managable in such a diverse and distributed environment, as any upgrades to how context is retrieved would require updating

the context retrieval library on all devices. Also, each application retrieving its own context must reside on a device which is capable of performing the required middleware computation. Low-power devices may not have sufficient computation power, and may therefore not be able to support the context service.

Instead an infrastructure model, using networked services, was chosen for Cashua. This architecture was proposed by Hong and Landay in [36]. In the infrastructure model, each context aware client communicates with a single service which is deployed as part of the context-awareness infrastructure of the environment. The service's responsibility is to provide the client with context, acting as an intermediary between the clients and sources, while all the raw context information continues to reside on the sources themselves. This removes the bottleneck of a single server approach to context, where all data must be collected and stored in one place.

This model also has advantages because clients need only discover the context service, not all context providers in the environment. This separates the need for complicated service discovery intelligence from the client, allowing it to make better use of its resources. Instead, the context service has responsibility for context discovery, and sending queries to where they can be answered. Context services can be deployed on different devices from applications, where more resources are available. Another advantage to this architecture is that the context system can be upgraded without requiring that software on each device be changed, as would be required in a middleware approach.

The disadvantage of this approach is that applications rely on support in the environment to resolve context, and are therefore not completely autonomous in their context-awareness. Instead, they are supplied with context by a service which is deployed in the environment and must be monitored and maintained like any other piece of software.

Because context-aware clients communicate with each other through the abstraction of this infrastructure, the context service can manage the variable availability of clients to present a stable interface to context. As long as this interface remains consistent, the context service can be upgraded without requiring changes to clients,

allowing for evolution of the system.

These context services form the basis of the Cashua system, and are called Context Service Nodes (CSNs). Figure 4.1 shows an example of a number of CSNs connected to a content-based network. There is no theoretical limit to the number of CSNs which could be present on the network, although practical limitations will apply, according to the pattern of traffic on the content-based network—the number and rate of quenches, subscriptions and notifications sent by the CSNs.

Each context-aware application runs on one of the devices shown to the side of the diagram. The applications each communicate with a single CSN, which communicates with others over the content-based network to resolve their queries. Applications can communicate with multiple CSNs if they wish, however as all CSNs communicate with the same content-based network and therefore have access to the same information there is no advantage in doing so. Applications can choose to connect to a new CSN in case of failure, providing a means of fault tolerance.

The CSNs exist to support applications running on devices in the environment, but are independent of these applications. CSNs continue to exist even if no context-aware applications are present in the environment. Having only one application served by each CSN would require an unnecessarily large number of CSNs, increasing the resources required to run the network, therefore the CSNs are designed to support as many applications as the resources of the host running the CSN will allow.

4.1.1 Context Service Node (CSN)

The fundamental unit of Cashua's design is the Context Service Node (CSN). The function of the CSN is to resolve context queries passed to it by a context-aware application. Each CSN is part of a network of these nodes, which communicate with each other to resolve queries posed to them. Each CSN is not dependent on other CSNs to operate, but make use of whatever CSNs are currently available on the network. This meets the requirement for network communication and availability

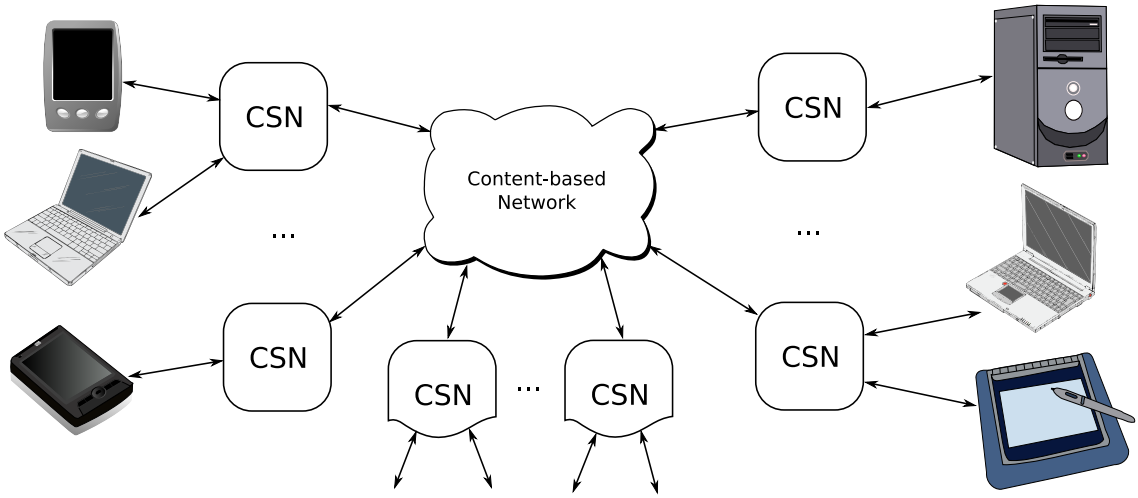

Figure 4.1: Cashua Network Architecture

outlined in the requirements framework in Section 2.6.

The design of the CSN required a number of elements at a high level of abstraction, and these are shown in Figure 4.2. The first of these was an **application interface** that applications could use to communicate with the CSN. This interface would be used for registration and query resolution. A **model repository** is used to store registered context model information.

These CSNs deal with two types of client: **local** and **remote**. **Local clients** are applications that register with the CSN in order to retrieve context information and also to make their own context information available. **Remote clients** are other CSNs who need access to context information from these clients. CSNs therefore handle both context querying and context sharing, communicating with each other to resolve queries.

The heart of the CSN's operation is the **query engine**, which is responsible for resolving queries on behalf of local and remote applications. The query engine uses a **network communication** module to manage its communication with other CSNs.

The CSN is more fully described in the following sections, which explain the choices made in the design of the CSN, and explain the functions of each of its components.

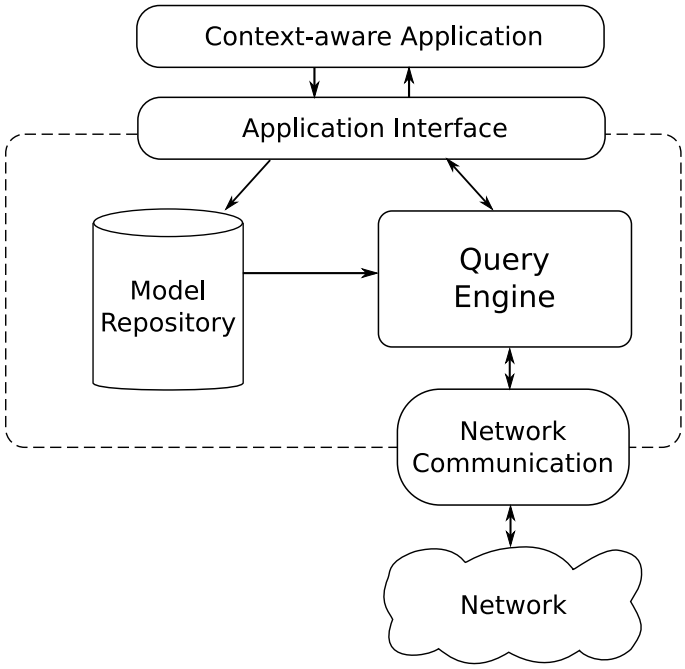

Figure 4.2: Context Service Node high level architecture

4.2 Support for Heterogeneous Context

Ubiquitous computing environments will naturally contain applications produced by different developers, which will nevertheless wish to communicate with each other to share context information. This provides challenges to the interoperability of these applications as described in Section 1.1.

Although standard data models for context such as those described in Section 3.2.2 have been proposed, none have gained widespread adoption, due to a number of factors. Firstly, the developer must decide which model best suits the purpose of the application, and this is a difficult task given the large number of available models. Two developers may select different models even though their applications use similar context information, and this leads to incompatibilities between applications.

Secondly, the large scope of what can be considered context information (explored at the beginning of Chapter 2) makes it difficult for any model to provide the features required by all applications. Models such as SOUPA[19] work around this issue by defining only an upper model, which can be extended as required. If developers choose to implement different extensions, they will not be able to utilize each other's

information without some means of relating their extended models to each other.

In order to support this heterogeneity and relationship construction, a context system must provide a mapping between application data models. This mapping is used when applications have chosen different models, or when they have extended a common upper model and wish to share information expressed in the extended parts of these models. This mapping allows each application to communicate with the context system using its own internal data model, providing independence from the environment in which it is deployed.

The selection of Cashua's modelling mechanism is discussed in Section 4.2.1 and the selection of a complementary mapping mechansim is discussed in Section 4.2.2.

4.2.1 Modelling Mechanisms

In order to select an expressive, standardized modelling mechanism, the current state of the art in the area was considered. Strang and Linnhoff-Popien's survey of context modelling approaches[64] compared a number of approaches and rated ontology approaches highly on a number of axes, particularly their formality and distributed composition. In addition to ontologies, topic maps were considered as an alternative context model representation.

Topic Maps

XML topic maps[5] are a knowledge representation technology which was investigated early in the design of Cashua. Although topic maps existed before the Web, XML topic maps were designed to apply the topic map paradigm to the Web. Topic maps attempt to connect pieces of data into a graph which represents the relationships between them. They do not attempt to enforce a rigid structure on the information they describe but rather they provide a lightweight way of navigating and accessing the information which exists in separately maintained data sources.

Topic maps provide a common framework for managing interconnected sets of

information objects. These information objects are the 'subjects' which are represented as 'topics' within the topic map. A subject can be pretty much anything that you can think or reason about: a chair, a webpage, a person, the sun, the concept of religion, or your favourite television character. A topic is created within the topic map to 'indicate' (or refer to) this subject. Information is then added to these topics by creating associations between them, thereby encoding additional knowledge. For example, the fact that a person has a webpage could be described by an association 'hasWebpage' between the topics representing the person and their webpage. The fact that a whale is a mammal is represented by an association 'isMammal' between the topics representing whales and mammals. The fact that a whale lives in the sea is described by an association 'livesIn', between the whale and sea topics; note that there is no restriction on the number of associations between topics.

While the lightweight nature of topic maps was attractive for modelling context, topic maps do not support more formal aspects of information modelling such as cardinality limitations and complex classes, which are required by other knowledge management applications. At the time of Cashua's design, topic maps also did not have support for expressing constraints. This has since been addressed and a draft proposal for expressing constraints on topic maps is currently under consideration with the ISO[1] for standardization.

Although Cashua does not directly make use of these more complex relations, their requirement for other knowledge management systems rules out topic maps for these systems, limiting the reuse of models expressed as topic maps. Another reason for the choice of ontologies is the lack of comprehensive tools for manipulating and reasoning over topic maps, while such tools already exist for ontologies.

Ontologies

Ontologies are used as a foundation for the Semantic Web[61], expressing knowledge in a formal way so that it can be reasoned about by computers. The Semantic Web is

[1] http://www.isotopicmaps.org/tmcl/

the vision of Tim Berners-Lee, and is a W3C-led initiative with the goal of providing technologies and standards for semantic markup of online information resources, enabling improved web navigation and supporting intelligent web services.

An ontology is an explicit specification of a conceptualization—a model of a particular domain. The Web Ontology Language (OWL)[48] is a W3C recommended language and has achieved wide popularity in the Semantic Web community. OWL is expressed in XML and has powerful expressions for making statements about classes, as well as instances and properties of those classes. OWL-DL, a popular subset of OWL, has Description Logic (DL)[3] equivalent semantics which provides decidable reasoning, guaranteeing the answer to certain queries in a finite amount of time.

The use of ontologies has the advantage that they are portable and reusable, as ontology languages have been standardized. Using ontologies to express context models in this way takes advantage of maturing work in the Semantic Web field. Tools exist to support the generation and maintenance of ontologies, notably Stanford University's Protégé[56] tool, which provides a graphical interface to ontology creation. Tools such as Pellet[63] and Racer[35] allow reasoning over ontologies, while the RDQL language[57] can be used to query ontologies to extract certain information.

A wide choice of ontologies is already freely available on the Internet, and the adoption of ontologies for knowledge representation has been growing rapidly[69], offering a wide choice to those wishing to reuse existing ontologies or use them as a starting point for their own needs.

Modelling Mechanism Selection

OWL ontologies were chosen to represent context models in the Cashua system. Initial work on topic maps for context representation[54] produced promising results, however although both ontologies and topic maps both provide lightweight means for expressing application data models, the formal and decidable semantics of OWL

ontologies allow more expressivity than topic maps. OWL's level of formality also allows for the checking of ontologies in order to determine their correctness. While not an absolute requirement for a context system, this checking can provide additional validation for application developers that their models are correct.

Although Cashua does not make direct use of all the features of OWL, the expressivity, formality and expressiveness of this language has promoted its use which provides opportunities for the reuse of a wide range of existing context models which are expressed as ontologies. Examples of some of these are given in Section 3.2.2. The availability of these models allows them to be adopted by application developers where possible, allowing for partial standardization of context models. Where this standardization occurs, applications can interoperate without the need for mappings.

Ontology Relations Used

Ontologies provide a very expressive set of features for describing characteristics of their domains of interest. Ontology languages can be used to express complex models using powerful relations such as cardinality restrictions, transitive and symmetric properties, and disjoint relations.

The aim of Cashua is not to provide a generalised framework for translation between arbitrary data models. As such, Cashua considers only concept hierarchies, composed of superclass/subclass relations, and equivalence relations between concepts. Of course, the other relations may be provided by application ontologies and will be checked by the ontology library for conflicting statements.

4.2.2 Mappings

The process of integration consists of providing mappings between the context models of applications. Having chosen ontologies as the means for expressing context models, it was natural to investigate the capability of ontologies for expressing mappings between models in order to achieve interoperability.

These mappings allow queries expressed using terms in one ontology to be translated into another. Software has been developed to analyze two ontologies and identify potential correspondences between them. This includes systems such as GLUE[29], ONION (ONtology compositION system)[45], OKMS (Ontology-based knowledge management system)[44] and OISIN[46]. Other systems are surveyed by Choi et al. in [22]. These tools can be used to greatly speed up the process of integration, and also to identify potential mismatches where a more complex mapping may be required to bridge the heterogeneity between the two domains.

Many of these systems produce mappings in a proprietary format. For example, the OntoMerge system[30] uses bridging axioms written in first order logic language to express the translation rules between the concepts in the ontologies. Another example is the MAFRA system[43], which uses Semantic Bridges based on an ontology called the Semantic Bridging Ontology (SBO). No clear standard has yet emerged to provide a format for expressing ontology mappings, so Cashua takes the approach of expressing its mappings using the *equivalentClass* mechanism which is provided by OWL.

OWL's standardization and direct support for equivalence relations made it the ideal choice for use in Cashua. If Cashua were to support more complex mapping types, another mapping language might be chosen. However the choice of mapping language must be considered carefully in order to maximize interoperability and mapping reusability. The equivalence relations provided by OWL were sufficient to demonstrate the operation of model mapping for the purposes of this work. Future work might include the investigation of more feature-rich semantic and schematic model mappings.

Mapping specification tools such as those described above can be used to partially automate the process of mapping between two ontologies. Typically they present a side-by-side hierarchical structure of concepts, and use a variety of techniques such as lexical analysis of concept names and structural analysis of the ontology to identify candidate matches between concepts. These candidates are presented to the user for verification, after which the user can choose to add additional matches

to complete the mapping.

Integrators can use these mappings to integrate applications into the context system. These mappings can be specified by an integrator who has no access to the application source code, only to the applications' ontologies. These mappings are therefore the functional unit of interoperability—they provide the bridge between heterogeneous applications. As applications are updated, and provide updated ontologies to express their context knowledge, these mappings can be updated accordingly without requiring source code modification to either the context service or to other applications.

Another advantage of encapsulating the mappings as ontology mappings (in an application-independent form) is that other context services can make use of them to discover mappings not previously known about. This reduces the workload on the integrator, as mappings need not be supplied for every application to be integrated, but rather can be discovered automatically.

4.2.3 Cashua Components Supporting Heterogeneity

Given the choice of ontologies for model and mapping specification, the following components of the Context Service Node (CSN) were designed in order to support the heterogeneity of context models within the system. The ontology repository and ontology mapping repository provide the *model repository* from Figure 4.2, while the ontology registration interface and mapping registration interface provide part of the *application interface* functionality.

Ontology repository The ontology repository stores ontologies supplied to the context service by its local client applications. The terms used in these ontologies compose the domain of knowledge that the context service can resolve queries about and it will advertise these terms to other context services, allowing appropriate queries to be routed to it.

Ontology mapping repository The ontology mapping repository stores mappings provided by applications or discovered during the course of querying. These mapping terms are also understood by the context service, as if we understand term A, and have a mapping that converts B→A, we can say that we understand B also. Publishing that they are understood allows queries to be routed to the CSN that will understand them.

Ontology Registration Interface The registration interface allows applications to register their context models with the CSN. During registration the application provides a context ontology, which defines the domain of context information that the application will query. These ontologies are stored in the ontology repository where they can be referenced by querying applications when they submit queries via the local query interface.

Mapping Registration Interface The mapping registration interface allows integrators to register mappings with the CSN. Integrators provide these mappings between the application's model and the model of data sources available on the content-based network. These mappings are stored in the mapping repository where they can be retrieved by the query engine, for query and response translation. Mappings are also retrieved so that they can be sent across the content-based network when requested using mapping discovery.

Query Engine The query engine (discussed in Section 4.5) supports heterogeneity by translating queries and results between models. It retrieves models and mappings from the repositories and uses this information to perform translation of queries sent to it. These queries can come from local applications or from the network.

4.3 Query and Response Routing

The decision to use a networked infrastructure model for context brought a requirement for a routing mechanism to allow the CSNs to communicate with each other to route context queries and responses, intelligently routing queries to where they could be answered. The routing mechanism must deliver queries to only those nodes which are capable of resolving them.

4.3.1 Routing Mechaisms

A number of choices for a routing mechanism were considered. For routing on the network level, IP multicast was considered. Overlay routing mechanisms such as peer-to-peer networking and publish-subscribe networking were considered before content-based networking was chosen.

IP Multicast IP multicast was considered as a choice for routing context information to a particular subset of hosts which express an interest in such information. This interest might be expressed by joining an IP multicast group, which would then be used to deliver context information on a particular topic to the application. This approach is attractive as it would limit traffic required on the network, as multicast routers would selectively forward information based on the current subscribers to a particular multicast group.

However, this would require support for IP multicast in all routing components on the network, which is unrealistic in the current environment as IP multicast adoption has been slow and it is not widely supported[28]. While it is likely to be more widely supported in future, this lack of low-level support could delay the deployment of context-aware applications if IP multicast was required. In addition, clients would need to be a part of a multicast group for each type of context they were interested in, with no method for filtering out unwanted information using additional conditions. This would require extra network resources as unwanted information was forwarded to members of the multicast groups.

Peer-to-Peer Networking Peer-to-peer (P2P) networking is often used in such distributed network environments, to transfer information between peers. However, P2P networking treats all peers as equal and connected to all the other peers. This is not the case in a ubiquitous computing scenario, where peers will wish to be very closely connected with nodes which are physically close to them and therefore likely to have relevant context information available. In such a scenario, the fully-connected nature of P2P networking is a disadvantage, as P2P networks will not favour information from nodes which are physically nearby.

Publish-Subscribe Networking Another networking architecture that was considered was topic-based publish-subscribe networking. Here the clients in the network register subscriptions for certain topics of interest with a server, and are sent messages which match those topics when the server receives them. However, this topic-based paradigm does not allow for publishing decisions to be made on all of the content of the messages sent through the server, only on the topic information.

The investigation of topic-based publish/subscribe led to the discovery of content-based networking, which is an extension to the topic-based paradigm. In content-based networking, messages are delivered to clients based on a subscription which can match on all the information present in the message, and route messages appropriately based on the available subscriptions. Content-based networking is discussed in the next section.

4.3.2 Content-based Networking

Content-based networking is an architecture where the distribution of network traffic is not restricted to the narrow definition of a combination of source and destination IP addresses and port numbers. Clients advertise a predicate, or subscription, which describes the type of messages they are interested in. Messages (called notifications here) containing data are therefore implicitly addressed to a range of hosts by the content they contain, which matches these predicates. Carzaniga presents an introduction to content-based networking in [11]. Content-based networks operate as

overlay networks, providing them with the flexibility to run on top of any underlying network protocol such as the Internet Protocol (IP). Several other content-based networking implementations exist, among the most cited are Elvin[60], Siena[11] and Gryphon[67].

The use of content-based networks does not prescribe a particular form of message, however attibute-value messages are typically used. These messages are often called **notifications** as they are used to notify a client of a particular event. An example notification might be as follows:

```
stock: AAPL
price: 93.96
```

When a notification is sent to a content-based network, it is routed according to available routing tables and distributed to nodes which have expressed an interest in being sent these notifications. Likewise, clients publish a predicate which describes the type of messages they are interested in. This predicate, also called a subscription, gives hosts their virtual 'addresses' on the content-based network. A predicate is an arbitrarily complex set of conditions against which each message can be evaluated. If the message evaluates true against the predicate it is delivered to subscribers who have registered that predicate, otherwise it is discarded. This **subscription** information is used to build routing tables on each content-based router, which describe to which neighbour routers messages should be sent. An example subscription might look like:

```
stock == "AAPL" && price > 90
```

Content-based networks use various mechanisms to maintain their routing tables. Elvin[2] uses federated servers to exchange subscription information, while a mechanism using Receiver advertisements (RAs) and Sender Requests/Update Replies (SRs/URs) is described in [12]. XRoute[14] provides a subscription update method with XPath subscriptions over XML data. Effort is made to reduce the amount of control traffic required to maintain these routing tables through mechanisms such as caching. When one subscription is more general than another, and so matches all messages which it matches, it is said to cover the other

subscription. Aggregating subscriptions which cover each other allows routing traffic to be minimized, and efficient routes to be generated.

Comparisons are made between content-based networking and current (though limited) approaches to information routing such as IP multicast in [13]. This document also evaluates approaches that minimize the time taken to make a routing decision, and produces concrete performance evaluations of these forwarding algorithms.

Quenches are an Elvin[60] mechanism which extends the subscription mechanism. Quenches allow clients to be aware of what subscriptions are present on the network. This is typically used to allow clients to evaluate subscriptions in advance and cease producing notifications when no matching subscriptions are present. When a client registers a quench on a particular topic, it is then notified via a quench notification when a subscription matching that topic is sent to the network.

4.3.3 Cashua components supporting Routing

Cashua uses a content-based networking library to provide the *network communication* component shown in Figure 4.2. Cashua uses the content-based networking messages as the basis for communication between CSNs.

Quenches are used to register a list of concepts for which the CSN has available context information. The CSN analyses the models and mappings which have been registered by applications, and publishes the extracted list of concepts via a quench.

When a CSN receives a query from an application it registers the query on the network using a subscription. This subscription consists of the query itself, along with a description of the concepts used in the query. This concept description is what is used to route the queries to the correct CSNs, which have registered quench information matching the query.

Once the remote CSN has resolved the query, it constructs a notification in order to return the result to the CSN which originally sent the query. This notification is designed to match the subscription which was used to publish the query to the

network. The content-based network can use this subscription which still exists on the network to route the notification containing the result. When the querying CSN receives this notification, it extracts the result and returns it to the application. The CSN then has the option of removing the subscription, or leaving it on the network as a long-standing query.

The most flexible mechanism is to allow the client to specify whether it wishes the query to persist on the network, or return as soon as the first result is retrieved. Cashua currently supports both of these modes of operation. An extension to this might be to have the CSN weight the results according to some set of criteria such as the reliability of the source, and select a result to return based on this information. Long-standing queries which persist on the network are useful to applications, but will naturally consume resources on the CSN and the content-based routers to be maintained, so application developers should be encouraged to only register long-lived queries for specific information which interests them.

Use of Quenches

Cashua does not require any changes to content-based networking software to operate. Using standard message types for communication means that Cashua could make use of any content-based network software. While quenches are not universally available, without them Cashua could be configured to use subscriptions to publish descriptions of available context and notifications to send queries.

The disadvantage of this approach is that the system would then require a separate communication mechanism to return the results. Queries would attach a 'return address', which would give details of where the query result could be returned. The remote CSN would then send query results directly to this address instead of routing them through the content-based network.

While quenches were developed for the Elvin content-based network, they have since been adopted by other popular systems such as Siena[12]. It is reasonable to assume that a quenching mechanism would exist for the content-based network in

use, however the above approach could be taken in cases where quenches were not available.

4.4 Supporting Dynamism

The dynamic nature of the ubiquitous computing environment also poses challenges to a context system as discussed in Section 1.1. In traditional environments, clients receive information from sources such as databases and web services which have static network locations where they can reliably be found to answer queries. Clients are given a DNS name which can be resolved to the IP address of the data source. When the source changes location and is assigned a new IP address, an update is made to the DNS to reflect this. These environments also have a relatively stable population of clients which querying data sources.

In contrast, ubiquitous computing environments will contain mobile, dynamic sources of context which will not be consistently available at a particular network location. A context management system must be designed in order to support the mobility and dynamism of its clients. Section 4.4.1 describes the alternative communication mechanisms which were considered in order to support dynamism. Content-based networking was selected to provide this support, and Section 4.4.2 describes how this is applied in Cashua, while Section 4.4.3 describes the CSN components which provide this support.

4.4.1 Design Choices

Initially, a centralized directory of context was considered for managing the dynamic nature of the network. This approach was attractive because all information could be retrieved from this central directory, requiring no reconfiguration of clients when they changed network locations. Similarly, context could be published to this location by all context sources. However, this approach was dismissed as it would limit the scalability of the architecture as one central server could not process context from

a large network of hosts.

Using IP multicast[24] was another potential candidate for managing dynamism, however this requires support at the IP networking layer in all routers on the network. While this might be realistic in future, support for IP multicast is not currently widespread, and may not be universally available in future. Designing an architecture to depend on this technology is therefore not advisable. Additionally, IP multicast works by clients subscribing to multicast groups, which causes routers to send messages addressed to those groups to all members of the group. However, there is no mechanism to filter the messages that will be received any further than membership of the group.

Also considered was the use of P2P networking to support the dynamic addition and removal of hosts in the network. P2P networks typically use a mechanism such as distributed hash tables (DHTs) to manage the distribution of information across a highly dynamic population of nodes. These methods do not account for the locality of information, and distribute it randomly rather than close to where it will be used. In ubiquitous computing environments, context information is likely to be located close to the clients who wish to use it. Therefore, an architecture which keeps this information where it will be used will provide better performance.

4.4.2 Content-Based Networking

Content-based networking is ideally suited for supporting the dynamic nature of ubiquitous computing environments. Here we study how the content-based networking solution can be applied to context service queries.

Cashua's design uses a network of CSNs which are members of the content-based network. Queries are sent using the subscription mechanism, which sends a message to the network with the contents of the query as well as the concepts used in it. Each CSN understands a set of concepts, registered with it by applications, and it publishes a description of these to the content-based network. This description is used to route queries through the network. Each router maintains a routing table

which describes the predicates (concepts, in this scenario) that its neighbours are interested in. As this published list of concepts is updated when applications move around the network and register their context models with CSNs, these routing tables are constantly updated to route queries to their destinations.

If none of its neighbours advertise that they are interested in a particular concept (ie. that they will understand a query referencing that concept), the query will not be forwarded through the network. This assists in reducing the amount of traffic generated by queries.

Queries are injected by other context services into the CBN, and routed hop by hop to the context services that will be able to resolve them (ie. understand the terms in the query). This approach also has the advantage that context services can perform long-lived queries that will be disseminated and will provide periodic responses whenever relevant information is available.

4.4.3 Cashua Components Supporting Dynamism

Cashua's local query interface forms part of the *application interface* from Figure 4.2 for context-aware applications, and supports dynamism by continuing to provide this interface to all applications which connect to it as they move around the network. Its content-based network library forms the *network communication* element of the design.

Local Query Interface The context service provides a query interface to which the application can forward queries. This query interface is used by applications (and the context connector) looking to retrieve context information. This interface takes in queries and passes them to the query engine for resolution.

Every time an application passes a query to the interface, it also specifies the ontology against which the query is expressed. This allows the query engine to determine exactly which concept is being referenced by each term in the query.

Content-Based Network Library The content-based networking component of the CSN is responsible for taking information extracted from models and queries by the query engine, and sending messages across the network based on this information. This component consists of the content-based networking library which is driven by the Query Engine to communicate with the content-based network. This component handles quench and subscription registration, as well as notification delivery on behalf of the query engine.

4.5 Query Engine Design

The query engine has three main functions. It helps to provide support for routing, heterogeneity and dynamism and so is discussed independently of the previous sections.

The first function of the query engine is to publish a description of the available context models registered by applications. Whenever new ontology information is received by the CSN, it consults the ontology and mapping repositories in order to determine the list of concepts registered with it, and then registers this list of concepts with the content-based network.

The second function of the query engine is to receive queries from registered clients and to resolve these queries by sending them to the content-based network. Each query passed to the query engine is expressed against a particular ontology, which the query engine can retrieve from the ontology repository. The query engine analyses the query in order to determine which concepts it references. It extracts the list of these concepts so that they can be used by the content-based network in resolving the query. The query engine can then pass the query including the concept information to the content-based network, which will then route the query appropriately.

The final function of the query engine is to take queries received from the content-based network, resolve them and return the result. These queries are received by the remote query interface, and are analysed in order to determine which concepts they reference. Each concept is then matched a concept from a

locally-registered ontology. If necessary, the query engine translates the concept using mapping information from the mapping repository, before sending it to a local data source. Once it receives the result from the data source, the query engine translates the result back to the data model of the querying application and returns it across the content-based network.

4.5.1 User Roles

As described in Section 3.3.2, the application developer and integrator are likely to be separate people. The developer writes their application to be independent of the environment in which it will be deployed, and this deployment is carried out by the integrator. These roles are defined as follows for the purpose of this work.

Application developer is the person who writes the context-aware application. This person will have intimate knowledge of the application, its data model and the external context information it wishes to make use of. However, they need not know where the external information is on the network, or the data model of this external information.

Integrator The integrator will have knowledge of the local environment and the available context sources. The integrator must take the application and deploy it into the environment, so that it can access the information from these sources.

The separation of the roles of developer and integrator allows these functions to be carried out by separate people. While a developer writes the application and provides it for use, the integrator can take this application and use it in her own environment. The developer should not need to know any particular information about the environment in which it will be deployed, and the integrator should not need to modify the application's code to allow it to work.

Figure 4.3 shows the interaction of the developer and integrator with the CSN. The context-aware application (written by the developer) registers its context model

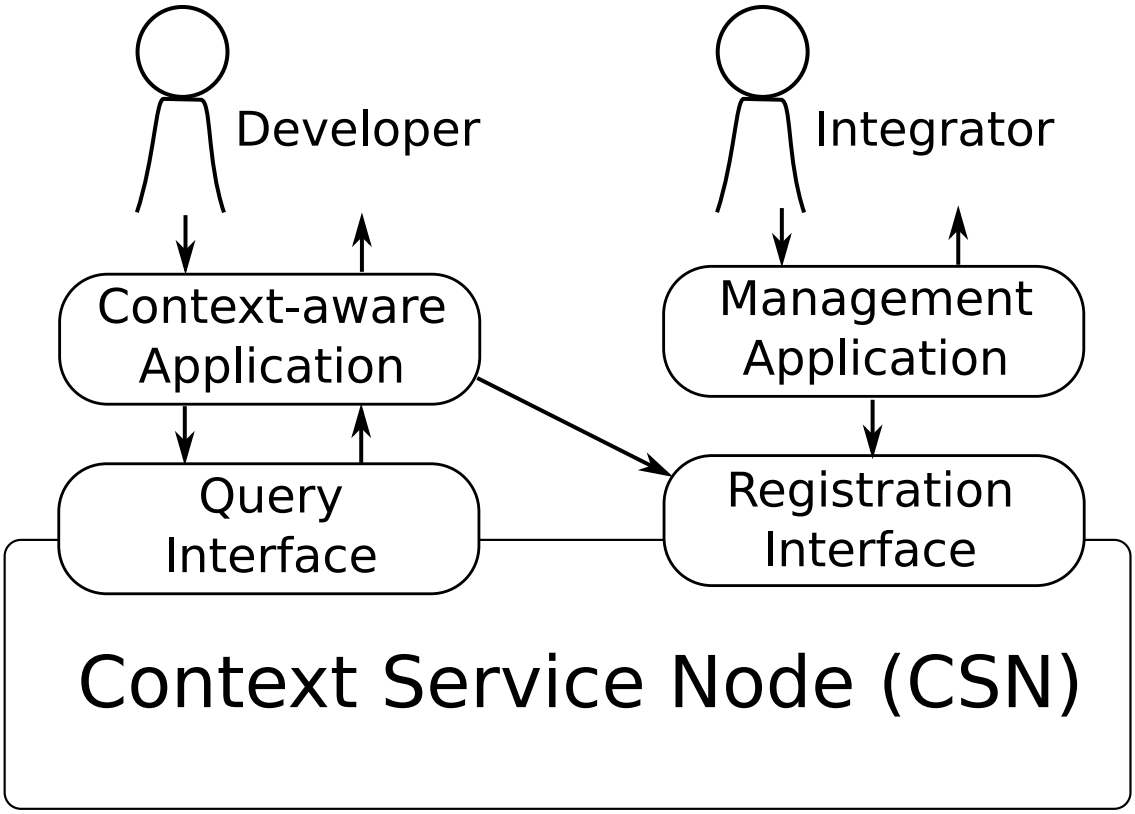

Figure 4.3: Interaction with the CSN

using the registration interface, and this registration is represented by the diagonal arrow in the diagram. The integrator uses the management application to register any necessary mappings. The application can then pose queries via the query interface of the CSN, and retrieve results.

4.6 Overall CSN Design

The overall design of the CSN is shown in Figure 4.4. The modules outside the dashed line are external to the CSN itself, but are drawn to show their interaction with the CSN. The modules that intersect with the dashed line are those that interface with the external modules.

Each of the CSN components has been introduced previously, however their combined use will be summarised here. This will be further expanded with a full example of how a query is resolved in Section 4.7. When it connects to the CSN, the

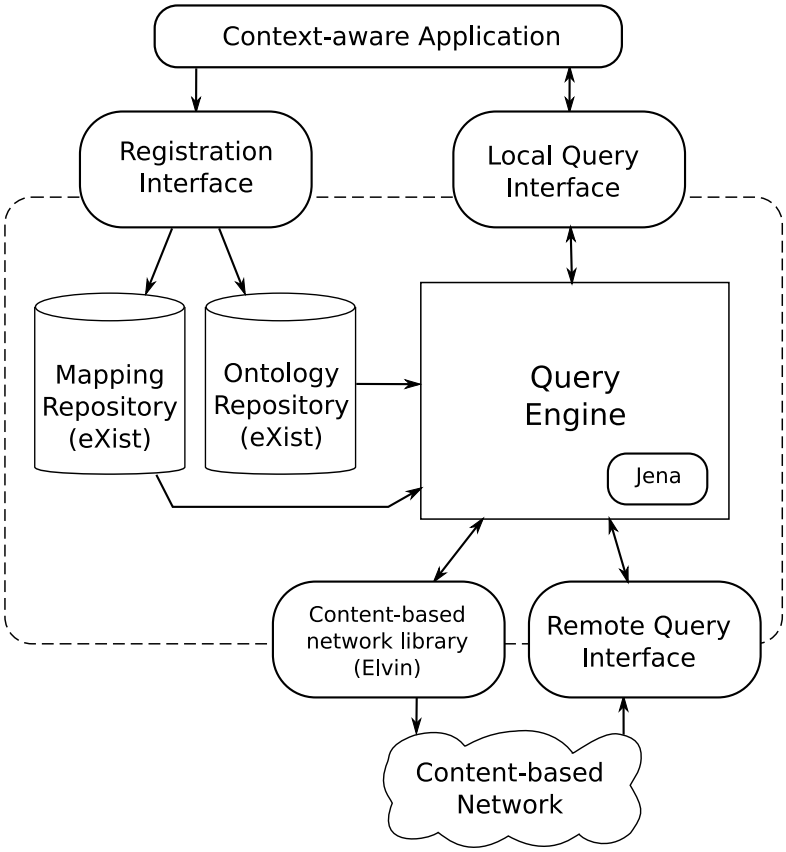

Figure 4.4: Context Service Node architecture

context-aware application submits its context model via the registration interface, where it can also optionally submit any mappings it knows. These models and mappings are stored in the ontology repository and mapping repository respectively.

These models are loaded by the query engine to form its concept model. The query engine uses this model to process queries from applications and determine the concepts used in the queries, using the content-based networking library to communicate with other CSNs and resolve queries. The query engine spawns a remote query interface to process queries from the network, translating them where necessary and passing them to a local database for resolution.

4.7 Query Resolution Example

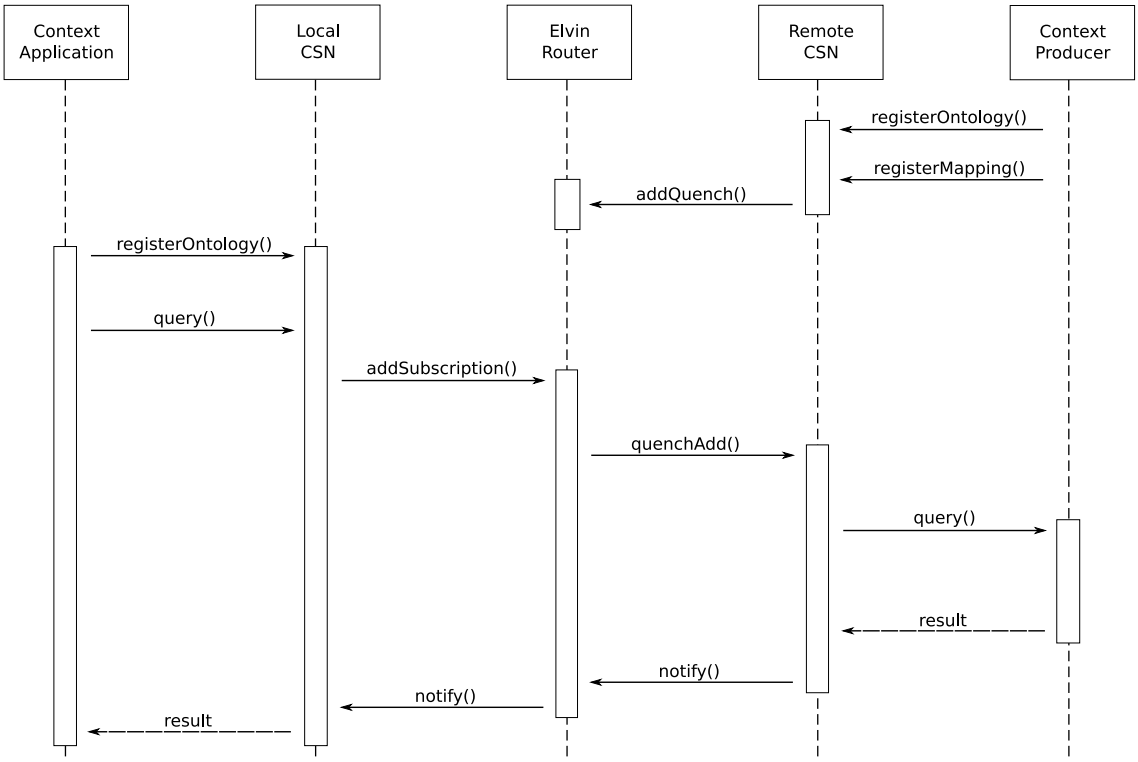

Figure 4.5: Query Resolution Process

The query resolution process is shown in Figure 4.5. The diagram shows a context-aware application on the left, who wishes to query for information that is available on the context producer on the right. It is therefore acting as a context

consumer. Both the producer and consumer have registered their context models with their CSNs, and the integrator has registered a mapping with the context producer's CSN.

The producer's CSN analyzes the provided models and mappings and extracts the list of concepts from the models. It then publishes a quench with those concepts to the content-based router.

Once these models have been loaded and the quench has been registered, the context-aware application can send its query to the local CSN. The local CSN analyzes this query to extract the list of concepts that it uses. It then publishes a subscription onto the content-based network, using the addSubscription() method to add the subscription to the Elvin router. This subscription contains two pieces of information. The first of these is the query itself, and the second is a list of the query's concepts.

Notifications are attribute-value based, and its quenches are designed to match subscriptions that contain certain attributes. Cashua therefore composes its subscriptions so that they contain the concept URIs as attributes, simply specifying that the value they match is 'true'. Once the content-based router receives this subscription, it is matched against the concepts in the quench and a 'quenchAdd' message is sent to the producer's CSN with the content of the subscription.

The producer's CSN extracts the query and list of concepts from the subscription, and uses its available ontology and mapping information to translate the concepts in the query into those understood by the context producer. Once this translated query has been extracted, it is passed on to the context producer such as a database to be resolved. When the result is returned it is translated back into the context model of the consumer by the query engine of the producer's CSN. This is the result that will be received by the consumer.

The result is sent across the content-based network as a notification. The notification is designed to match the subscription that was registered by the consumer's CSN, so it contains the subscribed concepts as attributes and also includes the query

text string. When the notification is sent to the router, it is matched against this subscription and sent to the consumer's CSN. The result portion of the notification is extracted and sent to the consumer.

4.8 Mapping discovery

The use of content-based networking also allows for the dissemination of ontology mappings. Mappings can be discovered because they are also published on the CBN. This allows some applications to be integrated automatically by discovering mappings that relate the needed terms. This is a welcome feature as it reduces the workload on the integrator.

Cashua's mapping discovery operates by CSNs registering quenches for concepts that they are interested in. In this respect, it is the same mechanism as for regular querying. Other CSNs receive these quenches and return addresses for where mappings can be retrieved. The CSN can then retrieve the mapping and use it in its future operation.

4.9 Additional tools

This section describes additional tools that were developed as part of Cashua, but which are not part of the core design. The context connector, described in Section 4.9.1, offers a traditional database interface to applications while conditionally retrieving some information through Cashua. The graphical CSN manager provides a GUI interface to the management of a CSN, its ontology repositories and querying, and is described in Section 4.9.2.

4.9.1 Context Connector

Some applications, particularly those not developed for use with Cashua, may not have support for registering model information and querying a CSN directly. For

these applications which would prefer to query a regular database to retrieve context, a context connector can perform these queries on its behalf.

The context connector provides a conventional database interface against which the application can perform queries. When the connector receives these queries, it can make a decision about whether they relate to context information. If they are context queries, the connector can filter them out and pass them on to the CSN. Otherwise, they can be passed to a conventional database.

This filtering of queries is based on the context data model provided to the CSN when the application is registered with it. Only queries over this model will be passed to the CSN for resolution, allowing the application to maintain its own information which is available as context information to other CSNs (and hence other applications). Queries against information which is not part of context will be passed unchanged through the context connector.

Using the context connector means that an application can query the connector instead of a traditional database, while the connector encapsulates all the needed knowledge about the CSN and access to context information. Hence, the application need not know whether or not its queries are context-related as it simply performs queries against the connector.

This allows the applications to query virtual views of context information through the connector, while these queries are actually transmitted to other context services to be resolved.

4.9.2 Graphical CSN Manager

Also designed was a graphical CSN manager, which could be used by both developers and integrators to test the operation of Cashua independently of any other code. This software interfaces with the CSN interfaces to manage the ontology and mapping repositories and perform queries.

Chapter 5

Implementation

This chapter describes the implementation of the Context Service Node (CSN), whose architecture is shown in Figure 5.1 for reference. The CSN is implemented in Java, and runs as a standalone application. The CSN makes use of external libraries which provide functionality that is not specific to Cashua. These libraries include the Elvin client library, the eXist embedded XML database, and the XML-RPC remote procedure call library.

The implementation of Cashua took place over a period of approximately six months. During this period the external components such as eXist and XML-RPC were integrated into the codebase and the CSN-specific code was also written. The CSN code itself consisted of about 1,500 lines of Java, while external applications such as the context connector and the graphical CSN manager amounted to approximately another 1,000 lines.

Each component of the CSN is discussed in the following subsections. The interfaces presented to context-aware applications are described in Section 5.1. These interfaces are used by applications to register the ontologies representing their context models, as well as mappings between models. These models and mappings are stored in repositories which are described in Section 5.2. Section 5.3 describes the query engine which is used to process queries. The engine uses a local query thread to resolve queries from local applications, and a remote query thread to

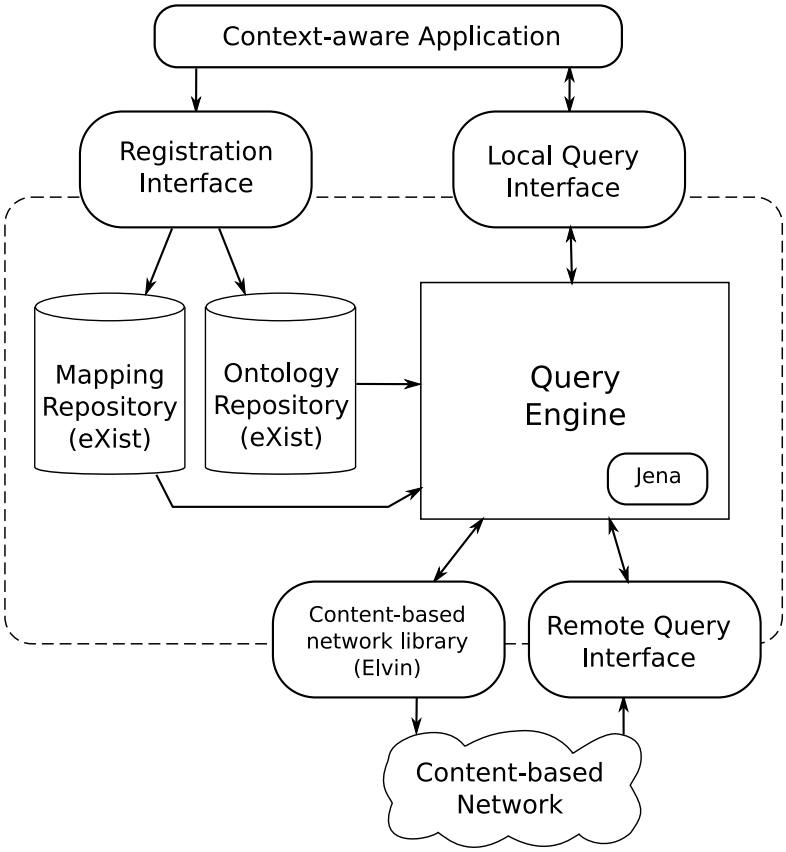

Figure 5.1: Context Service Node architecture

resolve queries from other CSNs across the content-based network.

5.1 Application Interfaces

XML-RPC[75] was chosen for communication between context-aware applications (both producers and consumers) and their CSN. It was chosen for its lightweight nature, and availability of free implementations. The XML-RPC implementation used in Cashua is the Apache project's Java XML-RPC[1]. This implementation automatically extracts method definitions from Java classes and exposes them via the XML-RPC interface.

Cashua uses the *WebServer* implementation which functions as a miniature embedded HTTP server. When the CSN starts up, a new *WebServer* instance is created which reads public methods from a class definition which is registered with the server under a particular prefix. These methods are then registered as the available XML-RPC methods of the server. Clients can execute methods on a particular object by calling the name of the public method, using the appropriate prefix in front of it.

Apache XML-RPC's automatic extraction of method definitions allows the CSN's public interface to be extended without needing to modify a complex protocol or change the code of the CSN's clients, as long as the methods they were using remain unchanged. The XML encoding of the protocol also allows CSN communication to be examined as it passes over the network for debugging purposes.

5.1.1 Registration interface

The registration interface is used to store an ontology model in the ontology repository. The registerOntology() method stores the model and then calls the query engine to update the CSN's quench with the new list of available concepts.

```
public String registerOntology(String ontologyText,
                        String ontologyName, String lang);
```

Similarly, the registerMapping() method stores the mapping in the mapping repository and then calls updateQuench() on the query engine.

```
public String registerMapping(String map, String mapName,
                        String lang);
```

In order to display which ontologies and mappings have been registered with the CSN, the listOntologies() and listMappings() methods are supported. These methods return a newline-separated list of ontologies and mappings respectively.

```
public String listOntologies();
public String listMappings();
```

Any of these ontologies and mappings can be removed from the CSN using the name returned by the list methods as a parameter to removeOntology() or removeMapping().

```
public String removeOntology(String ont);
public String removeMapping(String map);
```

5.1.2 Local Query Interface

The local query interface is used by applications to send their queries to the CSN for resolution. The query() method takes three parameters: the query string itself, the name of the ontology model used by the application, and the base namespace of that model.

```
public String query(String query, String ontologyName, String baseNS);
```

The query is an XPath[76] expression which selects XML elements from an XML database. The choice was made to use XPath queries over XML-structured data for a number of reasons. Firstly, this better supports existing applications which are not context or ontology-aware. These applications can use their existing XPath queries and parse results without modification to their source code, demonstrating the ease with which pre-existing applications can be integrated with Cashua. This was investigated using experimental work to evaluate the developer effort required to integrate with Cashua.

Secondly, application developers can use the well-known XPath query language, reducing development time when compared to introducing them to RDQL[57] or another ontology query language. Thirdly, Cashua's approach is demonstrated by the integration of content-based networking and ontologies to deliver context, and is therefore independent of the query language used and the storage format of context information. The choice of XPath queries over XML data allows clear demonstration of this approach without the added unnecessary detail of more complex data models or query types.

5.2 Ontology/mapping repositories

The ontology and mapping repositories are implemented using an eXist XML database. While eXist is capable of running as a standalone server or in a servlet context, the CSN uses an embedded version of the database. This allows the CSN to have full access to the database without starting network listeners, as the database does not need to be accessed by any external application. This saves resources and also simplifies the administration of the CSN.

The embedded database is created using the XML:DB[74] API. The database is stored on the CSN's local filesystem, allowing ontologies and mappings to persist between restarts of the CSN. The CSN's storage and retrieval is also done via the XML:DB API, which gives the opportunity to easily substitute the eXist database for any other database which implements this API.

The choice of an XML database to store the ontologies and mappings allows the CSN to perform queries directly where needed, bypassing the processing of the ontologies through the ontology library. For example, the query engine can retrieve the base namespace of an ontology through an XPath query. Also, the registration interfaces can use the eXist resource storage API to store ontologies and mappings, removing the need for the CSN to reimplement file-handling logic to store models on the CSN's local filesystem.

The eXist database is a full-featured native XML database and naturally some of its

more advanced features aren't utilised by Cashua. Implementing local file-handling would allow models to be stored on the CSN's local filesystem, reducing the resources required to run the current version of the CSN. However it was felt that the features of using such a database outweighed the performance gains which might be achieved by reimplementing the model storage functionality it provides.

5.3 Query Engine

The query engine makes use of both the Jena library, which is described in Section E.3, and the Elvin library, discussed in Section 4.3.

5.3.1 Quench Registration

Once the query engine is started, it initializes the Elvin library and creates connections to the ontology and mapping repositories. These connections allow the engine to retrieve the registered models, in order to extract the list of available concepts. The query engine loads each model and mapping in the repositories to form the current CSN's current concept model. The ontologies are loaded using the Jena library, and this final concept model contains all the information in the ontologies that compose it.

Once the concept model has been formed, the classes used in this model are extracted and used to form the quench list, which is registered with the content-based network using a Quencher object. This quench is also updated when the available models change, via a registration or removal.

As the quench is formed, the query engine registers itself as a callback object for when a quench update message is received, as it implements the QuenchListener interface. The quench update method is called when subscription is registered on the network which matches the quench. When this happens, the query engine invokes a remote query thread to process the query.

5.3.2 Local Query Thread

When a query is received from an application on the local query interface, the query engine spawns a local query thread to resolve it. The thread is initialised with required information such as the concept model from the query engine and then its query method is invoked with the supplied query and ontology URI.

On receiving this query, the query thread first analyzes it to extract a concept list. This is done by analyzing the XPath query and comparing it against the concept model. The query is parsed using the eXist XPath parser in order to generate an Abstract Syntax Tree (AST) of the query. The nodes in this tree represent the tokens in the query, such as '/' (SLASH) and the names of XML elements (QNAME, which stands for qualified name). The local query thread navigates this tree to find the qualified names which correspond to ontology concept identifiers.

Once this list of concepts has been determined, the local query thread generates a subscription containing both the query and the list of concepts. The subscription contains the URIs of the concepts extracted from the query as its attributes, with the value of the attribute set to 'true'. This is because of how quenching works, matching on attribute names for which there are registered subscription.

The local query thread registers this subscription using the Elvin library and registers itself as a notification listener the subscription. When a notification for the subscription is returned, the notificationAction() method of the thread is invoked, which it implements as part of the NotificationListener interface. When this method is called by the Elvin library, the thread is notified that a result has been received. The thread can then extract the result and return it to the client. It then has the choice of either terminating or waiting for additional results from other CSNs, which it can return to the client.

5.3.3 Remote Query Thread

The remote query interface is implemented by a RemoteQueryThread which is created by the query engine to handle the query which has been received from the network. Remote query threads are created when a quench notification message is received from the content-based network. When another CSN registers a subscription which matches the concepts contained in the CSN's quench, this causes the Elvin library to call the quenchAdd() method on the query engine.

From the QuenchEvent message which represents the quench notification, the query engine extracts the list of concepts in the quench. It then analyzes the concept model in order to find equivalent concepts, which are present because of mapping information. It then initializes the remote query thread with this information.

The remote query thread analyzes the XPath query similarly to how the local query thread, identifying the concepts. However, as it analyzes the AST it also converts the concepts found to the equivalent concepts which are understood by this CSN. As it performs this conversion, it builds up a translated query. Once the AST has been fully traversed, this query is sent to the local data source. In the experiments carried out for evaluation, this was implemented as an eXist database which was accessed over a network connection.

Once the query result has been returned, the remote query thread translates the XML in reverse, to the concept names used in the original query. It then constructs a notification which contains the query result as the value of an attribute, and has the concepts of the original query as the other attribute names. This notification is designed to match the subscription registered by the local CSN. When the remote query thread sends this notification, it is routed by the content-based network back to the local query thread which registered the subscription, and it is returned to the client as described in the local query thread section.

5.4 Query Resolution Example

To further explain the query resolution process, an example from one of the evaluation experiments is presented. Here the Adaptive Engine (AE), a software application written in Java, wishes to access remote context.

The AE has registered its context model with the CSN which is as follows:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns="http://yossarian.cs.tcd.ie/ruaidhri/ae.owl#"
  xml:base="http://yossarian.cs.tcd.ie/ruaidhri/ae.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="aeidentifier">
      <rdfs:subClassOf>
          <owl:Class rdf:ID="learner"/>
      </rdfs:subClassOf>
      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >general/identifier</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="knowledge">
      <rdfs:subClassOf rdf:resource="#learner"/>
      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >educational/adaptivity/adaptivitytype/set/candidate/langstring</rdfs:comment>
  </owl:Class>
</rdf:RDF>
```

This model consists of three concepts (classes), which are learner, aeidentifier and knowledge. The aeidentifier and knowledge classes are subclasses of the learner class, and *rdfs:comment* Strings attached to the classes mark the XPath paths for each of these concepts in the AE's data model. Each class can be prefixed with the value of 'xmlns' above to get the full URL of the concept.

A remote XML database contains learner information in LIP format (different from the AE's format). It has registered its context model with the CSN, which is as follows:

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://yossarian.cs.tcd.ie/ruaidhri/lip.owl#"
  xml:base="http://yossarian.cs.tcd.ie/ruaidhri/lip.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="name">
      <rdfs:subClassOf>
          <owl:Class rdf:ID="learnerinformation"/>
      </rdfs:subClassOf>
      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >identification/formname/contentype/referential/sourcedid/id</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="ability">
    <rdfs:subClassOf rdf:resource="#learnerinformation"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >competency/contentype/referential/indexid</rdfs:comment>
  </owl:Class>
</rdf:RDF>
```

This model defines the learnerinformation, name and ability classes. The name and ability classes are subclasses of learnerinformation, and as in the previous example *rdfs:comment* Strings on these classes mark the XPath paths for these concept within the LIP data model.

The remote CSN also has a mapping registered by an integrator between the AE and LIP models:

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://yossarian.cs.tcd.ie/ruaidhri/ae.owl#"
  xml:base="http://yossarian.cs.tcd.ie/ruaidhri/ae.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="aeidentifier">
      <rdfs:subClassOf>
          <owl:Class rdf:ID="learner">
              <owl:equivalentClass rdf:resource=
        "http://yossarian.cs.tcd.ie/ruaidhri/lip.owl#learnerinformation"/>
          </owl:Class>
      </rdfs:subClassOf>
      <owl:equivalentClass rdf:resource=
        "http://yossarian.cs.tcd.ie/ruaidhri/lip.owl#name"/>
  </owl:Class>
```

```
  <owl:Class rdf:ID="knowledge">
      <rdfs:subClassOf rdf:resource="#learner"/>
      <owl:equivalentClass
                rdf:resource="http://yossarian.cs.tcd.ie/ruaidhri/lip.owl#ability"/>
  </owl:Class>
</rdf:RDF>
```

This ontology describes a mapping from the AE model to the LIP model. For each of the concepts in the AE hierarchy, an *owl:equivalentClass* statement provides the full URL of the equivalent concepts in the LIP model.

The remote CSN's query engine reads its ontology and mapping repositories, and generates its concept model from the LIP model and the mapping. This model contains the combined list of concepts it understands, which the CSN publishes the following list of concepts as a quench:

```
http://yossarian.cs.tcd.ie/ruaidhri/ae.owl#learner
http://yossarian.cs.tcd.ie/ruaidhri/ae.owl#aeidentifier
http://yossarian.cs.tcd.ie/ruaidhri/ae.owl#knowledge
http://yossarian.cs.tcd.ie/ruaidhri/lip.owl#learnerinformation
http://yossarian.cs.tcd.ie/ruaidhri/lip.owl#name
http://yossarian.cs.tcd.ie/ruaidhri/lip.owl#ability
```

In this query, the Adaptive Engine (AE) wishes to retrieve the competencies of a particular learner. It does this by selecting a 'learner' element from the XML, which has an 'identifier' (typically a username) containing a particular string. From the learner element matching this user it extracts a set of candidate identifiers, which describe topics that the learner has a competency in. The query it sends to the CSN is as follows:

```
/learner[general/identifier='user']/educational/adaptivity
        /adaptivitytype/set/candidate/langstring
```

The query is sent to the local CSN, which analyzes it to extract the list of concepts by navigating through the AST of the query. The local query thread then creates a subscription which looks like:

```
query == ("/learner[general/identifier='user']/educational
        /adaptivity/adaptivitytype/set/candidate/langstring")
&& require(http://yossarian.cs.tcd.ie/ruaidhri/ae.owl#learner)
&& require(http://yossarian.cs.tcd.ie/ruaidhri/ae.owl#aeidentifier)
```

```
&& require(http://yossarian.cs.tcd.ie/ruaidhri/ae.owl#knowledge)
```

This subscription contains the text of the query, while the 'require' strings list the concepts referred to in the query. The subscription is sent to the content-based network, where it is matched against the quench published by the remote CSN. The remote CSN is notified of this subscription due to the quench, and spawns a remote query thread to process the query. It extracts the query text from the subscription, parses the query and derives the AST. It then navigates the AST and uses the information in the concept model to compose the translated query.

```
/learnerinformation[identification/formname/contentype/referential
    /sourcedid/id='name']/competency/contentype/referential/indexid
```

This query is sent to the local XML data source containing learner information, which responds with the result:

```
<indexid>db.concepts.introduction</indexid>
<indexid>db.concepts.relational_model</indexid>
```

The remote query thread then translates the result back to the model of the querying CSN, using the concept matches it previously made to translate the XML so that it appears as if it is the result of the original query, including the names of the matching XML tags:

```
<langstring>db.concepts.introduction</langstring>
<langstring>db.concepts.relational_model</langstring>
```

This result is sent back to the local CSN in the form of a notification:

```
result: <indexid>db.concepts.introduction</indexid>
        <indexid>db.concepts.relational_model</indexid>
query: /learner[general/identifier='user']/educational/adaptivity
        /adaptivitytype/set/candidate/langstring
http://yossarian.cs.tcd.ie/ruaidhri/ae.owl#learner: true
http://yossarian.cs.tcd.ie/ruaidhri/ae.owl#aeidentifier: true
http://yossarian.cs.tcd.ie/ruaidhri/ae.owl#knowledge: true
```

This notification matches the subscription which was registered by the local CSN, so the notification is delivered to it. The result is extracted from the notification and returned to the client.

5.5 Experimental Platform

These experiments were conducted on a Xen[73] virtual machine running Debian Linux, allocated 256MB of RAM on a 3.0GHz Pentium 4 server. Tests were also carried out with CSNs on a 2.0GHz Pentium 4 desktop with 512MB of RAM, in order to test Cashua's networking functionality.

5.6 Additional tools

5.6.1 Context Connector

The context connector acts as an interface between an application and its data store. The function of the context connector is to function as a traditional data source from the point of view of the application, while selectively forwarding queries to the CSN.

The context connector implements an XML-RPC interface which mimics the interface offered by the eXist database. Applications configured to query eXist can communicate with the context connector, which will make a decision to forward their queries to the CSN or to another data source.

This allows legacy applications to run without being updated to be aware of the CSN, or whether the information they need is available as context or not. Instead, they can simply query a traditional database while the CSN performs these tasks on their behalf. Currently the context connector makes this determination based on the XPath prefix of queries sent to it, however this could be extended in the future as required.

Chapter 6

Evaluation

This chapter describes the evaluation of the Cashua system that was performed. The aim of this evaluation was to assess the work undertaken to achieve the first two objectives described in Section 1.2. In order to do this, this chapter does the following:

(a) evaluates the mechanisms established to enable context delivery in heterogeneous and dynamic ubiquitous computing environments.

(b) evaluates the design of Cashua with respect to the requirements framework detailed in Section 2.6 and the current state of the art.

The evaluation method for (a) consisted of experiments to give quantifiable information in order to demonstrate the performance of Cashua (described in Section 6.2) along with the operation of the mapping distribution mechanism (described in Section 6.3), and also case studies which examine the application developer's experience using Cashua in order to demonstrate its feasibility (described in Section 6.4).

The evaluation method for (b) was to conduct a review of Cashua with respect to the requirements framework (described in Section 6.6.1), and with respect to additional requirements from state of the art systems (described in Section 6.6.2).

6.1 Evaluation Overview

Firstly, experiments were conducted to assess key performance indicators of Cashua, in order to evaluate the routing and heterogeneity features of Cashua. Current applications directly query their data sources, which is efficient but very limited and inflexible in the information that can be received. Using an infrastructure-based context system means that instead of communicating directly with databases, sensors or other information sources, the application must communicate through the context system to retrieve the context information it needs. This evaluation, presented in Section 6.2, examines both the latency experienced by the application and the resources consumed by the Cashua system during operation in order to assess the extent of the performance impact this causes.

Secondly, the distribution of mappings was considered in order to evaluate this important part of Cashua's support for heterogeneity and dynamism. A design feature of Cashua is its ability to cope with this heterogeneity through the use of ontology mappings. The automatic distribution of these mappings provides a mechanism for the CSN to discover information needed to translate between context models, allowing for applications to move between different CSNs without requiring human intervention to provide the mapping information. This is important for providing context information in ubiquitous computing environments, where large numbers of devices and their dynamically changing location make manual configuration impractical. The second evaluation objective was to assess the correct operation of this mechanism for distributing mappings, by evaluating the correct distribution of mappings. This experiment is described in Section 6.3.

The next part of the evaluation was to assess the feasibility of using the system for application developers. In order to encourage the use of context information in applications, the process by which developers make their applications context aware must be as easy as possible. Two case studies were performed, integrating two software projects with Cashua and using questionnaires to assess the experience of application developers. The results of these questionnaires, which are presented

in Section 6.4, indicate the feasibility of the approach.

This chapter focuses on some of the objectives discussed above. In order to provide a fuller discussion and put this work in context, Section 6.6.1 compares Cashua with the current state of the art systems, along the axes of the requirements framework described in Section 2.6. Finally, Section 6.6.2 evaluates the performance of Cashua along a number of additional axes not covered in the state of the art.

6.2 Performance Experiment

6.2.1 Overview

A context system must provide results for context queries as fast as possible, with a minimum use of resources. Although constant advances in computer hardware will improve the performance of all software by the time context-aware applications are commonly deployed, a context system which performs well on current hardware will be even more scalable in the future, as these advances reach the marketplace.

A scalable design allows for more applications to be supported by providing timely query results, while maintaining relatively low resource consumption. The advantages of the Cashua system as discussed in Section 4.1 must be balanced against the resources used to provide these results. The performance experiment was designed to measure how well the Cashua system performs using a number of metrics.

The aim of the performance experiment was to evaluate what performance Cashua would provide. In order to do this, a series of experiment parts were designed in order to test the design characteristics of Cashua.

6.2.2 Design

The experimental method in evaluating Cashua's performance was divided into a number of stages. The first stage was to define a set of measurable performance

metrics. The next stage was to provide a number of context models with varying characteristics for use by test applications. These models consist of superclass/subclass hierarchies, as these are the model features used by Cashua and any additional information in the model will not have an impact on the operation of Cashua once the model has been processed by the Jena library. The final stage was to automatically generate a set of queries over each model, which could be used by applications.

CSNs were then set up in two different configurations, and their performance measured. The first configuration focused on the overall performance of Cashua, using CSNs connected to each other via a content-based network. These experiments are described in Section 6.2.3.

Another configuration evaluated the performance of a CSN in isolation, independent from the content-based network, and this evaluation is described in Section 6.2.4.

Performance Metrics

The metrics that were used to evaluate the performance of the CSN in resolving queries are listed below.

Throughput The number of queries per second that can be processed by the CSN. Measuring the query throughput of the CSN allows the calculation of an estimate of how many applications each CSN could support, if the average query rate of clients is known.

Latency The response time (latency) for a query impacts directly on the end-user's experience of a context-aware application. If the query latency is too high, the user may get frustrated with the application. The experiment is designed to validate whether the distribution of query latency matches the end-user's requirements. The ideal result here is a small number of high-latency queries or a small average query time.

Human-computer interaction (HCI) research has shown that application response times of 0.1 seconds give the user to impression that the application

is responding immediately, while a limit of 1.0 seconds is the threshold to maintain user's attention on the task at hand[10][62]. Context-aware applications must be able to resolve queries quickly, in order to maintain the user's attention.

Required Resources This metric identifies the resources required by the CSN to resolve queries posed to it. These resources include CPU time, memory and network bandwidth consumed. By measuring the CSN's use of these resources as it resolves queries, we can evaluate how its performance will scale with different query sets, and also determine how small a resource footprint a CSN might consume.

This metric is important in order to determine whether resource-constrained devices such as PDAs can host CSNs.

These metrics were used as part of this experiment to determine the performance characteristics of the Cashua system.

Design of Context Models

Context models are used by the CSN to identify the concepts used by applications in their queries, and also to translate queries and results between models. These models are processed by the query engine and are navigated according to the terms used in the query. The models used in this experiment therefore test the operation of the query engine, as it will operate differently depending on the context models which it must process in order to resolve the queries.

Testing the operation of CSNs using a range of context models shows their performance using models with different characteristics. The context models used in this experiment were generated by a Java utility which produces an ontology hierarchy with certain characteristics. These characteristics can be provided as input to the utility, and it will produce a context ontology model. This model can then be loaded directly into a CSN for querying.

Using programmatically generated models allows for the CSN to be tested in specific ways by varying these characteristics. This approach was chosen rather than use general-purpose ontologies written by others, because it allows examination of how the system performs as these characteristics are varied. Using general-purpose ontologies would add more variables to this equation, making it more difficult to determine how Cashua's performance varies as these characteristics change.

Two characteristics of the context models were considered: their depth and breadth. Deep ontologies contain deep subclass hierarchies, while broad (or 'bushy') ontologies contain a large number of concepts not arranged into hierarchies. Real-world examples of bushy, shallow ontologies include personal identity ontologies such as FOAF[7], calendaring and geospatial ontologies, as well as ontologies describing flattened relational databases. Deep ontologies are found in more complex domains, such as science, engineering or specialist domains. The Wine ontology[72] is a good example of an ontology which is both deep and bushy.

The model generation utility was used to generate models that were deep but thin, that were bushy but shallow, and that were both deep and bushy. These models were labelled "deepThin", "shallowBushy" and "deepBushy" respectively. Models that were shallow and thin did not contain enough concepts to fully test the query engine's operation, and were therefore not considered.

The deep and thin ontologies contain 20 levels of sub-concepts, as shown in Figure 6.2. Shallow, bushy ontologies contain 20 top-level concepts, as shown in Figure 6.1. Deep and bushy ontologies combine these two features to have 20 trees, each 20 concepts in depth for a total of 400 concepts. The structure of the deep and bushy ontology is shown in Figure 6.3.

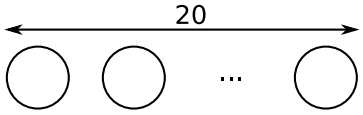

Figure 6.1: Shallow, bushy ontology structure

These models test the CSN by using various superclass/subclass type hierarchies in order to test the CSN's concept matching. These models are not designed to be

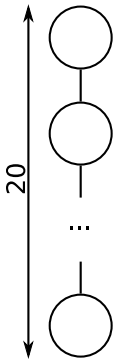

Figure 6.2: Deep, thin ontology structure

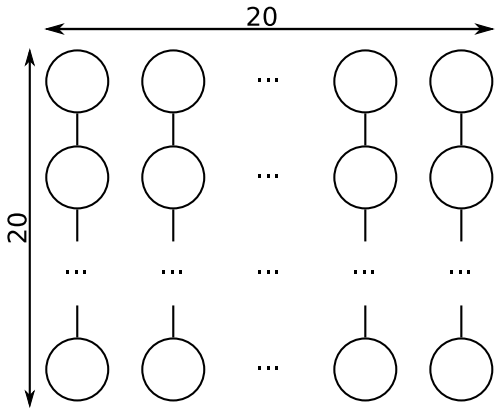

Figure 6.3: Deep, bushy ontology structure

fully representative of all real-world ontology models, which may be more complex and contain additional relations such as constraints which are not covered here. However, the CSN only considers these type hierarchies when extracting concepts, and does not process any additional model information.

This means that once the model has been loaded by the query engine using the Jena library, no additional processing needs to be done on this additional information. Cashua only reloads this model when new information is registered. For each query, any additional model information will have minimal impact on the time taken by Jena to search for the required concepts, as the extra information is ignored. Searching the model may take slightly longer as more statements are present, however this search time will scale linearly with the number of statements in the model, irrespective of the type of those statements.

Design of Queries

A variety of queries over these context models were used, to evaluate the performance of the CSN as these queries are processed. A query generation utility was written to generate query sets for each context model. This utility was used to simulate the context-aware application submitting queries to the CSN and receiving results.

Because the generation of the queries can be automated, new sets of test queries can be easily generated against a particular context model. For each model, the query generator chooses concepts from the deepest part of the context model hierarchy. The query generator produces a query selecting this concept as shown below, where the N in class$N$ refers to the level in the hierarchy.

/class1/class2/.../class19/class20

This is the worst-case scenario for the query engine, as it requires searching to the deepest part of the concept tree to find the relevant concept. Any queries which reference higher-level concepts in the tree are resolved more quickly because as the CSN navigates down the concept hierarchy it can terminate the search as soon as the concept is identified, ignoring information lower down in the concept tree. Choosing the lowest-level concepts in these trees requires the most computation from the CSN for the deep ontologies.

Design of performance experiment parts

The performance experiment consisted of two parts. The first part was to measure the CSN's performance as queries and responses were sent between CSNs connected over a content-based network. This was designed to test the Cashua in the most realistic way possible, by using it to query remote databases connected to other CSNs over the content-based network. This part is described in Section 6.2.3.

The CSN was also tested in isolation, as described in Section 6.2.4. This provided information about how the CSN itself performs, without the added variable of the content-based network performance.

Experimental hardware

The performance experiments were conducted on a Xen[73] virtual machine, allocated 256MB of RAM on a 3.0GHz Pentium 4 server. Where additional tests using multiple servers were carried out, these were performed using a 2.0GHz Pentium 4 desktop with 512MB of RAM.

6.2.3 Elvin Part

Design

This part of the experiment was designed to test queries being resolved by CSNs over the content-based network. The experiment was configured as shown in Figure 6.4. In the first run, the Elvin content-based networking server and the two CSNs were deployed on the same physical computer, to eliminate the possibility of network latency being a limiting factor in performance. On subsequent runs, each of the CSNs and the Elvin service ran on separate physical computers.

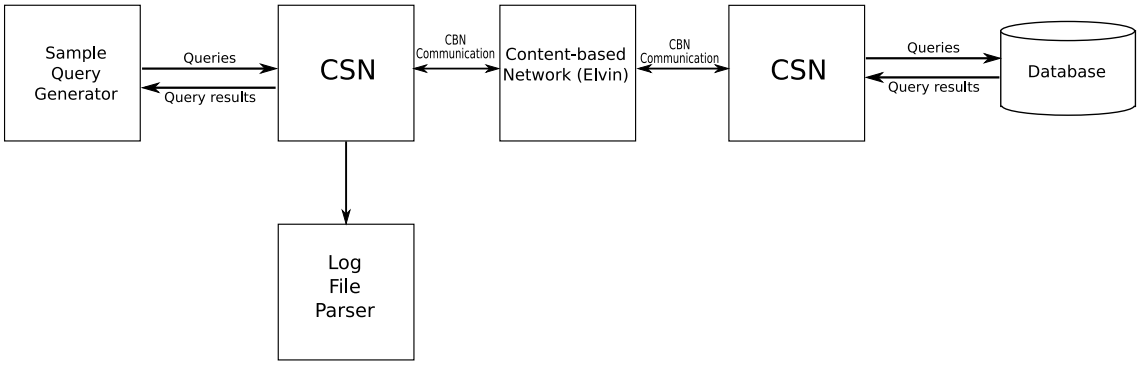

Figure 6.4: Configuration for the Elvin part of the performance experiment

The queries were sent to the CSN, and the time taken to return the result was measured. In addition to the total response time, the time spent by the CSN on each stage of the query resolution process was recorded into its log file. For a 'local' CSN, which receives the query directly from the client, the query stages that were timed were:

- parse the query (parseQuery)

- find the concepts in the query (findConcept)
- compose the subscription (subs)
- wait for a result (waiting)
- remove the query subscription (remove)

For a 'remote' CSN, which receives this query, resolves it and returns the result, the stages involved were translating the received query (transQuery), sending the query to its local eXist database and receiving the result (eXist), translating the result (transResult) and returning it via a notification (notif).

The experimental client repeatedly generated queries over the context models listed, using two thousand queries on each run, and the times taken were recorded. Once the experiment had been performed with a single client sending queries one after another, multiple clients sending simultaneous queries were used to further stress-test the CSN, in order to determine the maximum query rate it could sustain.

Results

Over a run of two thousand queries, the average time for each stage of the query was recorded, as well as the total query time. This was done for both the local and remote CSN.

The times for the local CSN are shown in Table 6.1, and graphed in Figure 6.5. The average query time for a round-trip query through Cashua was 119.4ms. Analysing the individual stages of the query shows that the query parsing (parseQuery) stage took the most time on the deep queries, which contained a large number of terms. Concept extraction took less than 1ms on most queries, including querying over the deepBushy ontology with 400 concepts.

The remainder of the processing time taken by the local CSN is dominated by adding a subscription (subs) and removing it (remove). These are Elvin operations provided by the Elvin library, and are therefore not part of the Cashua core code.

Model type	parseQuery	concept	subs	waiting	remove	total
deepThin	4.6	0.8	38.1	34.5	39.9	118.4
(Std. dev.)	6.3	0.8	3.0	5.8	0.7	8.88
deepBushy	4.5	1.0	36.3	35.6	40.8	122.1
(Std. dev.)	1.0	0.7	5.73	3.4	0.5	5.2
shallowBushy	0.9	0.1	37.7	38.3	39.9	117.6
(Std. dev.)	0.8	0.4	1.6	1.0	0.4	1.0
Averages	3.3	0.63	37.4	36.1	40.2	119.4

Table 6.1: Average local CSN query stage times in ms.

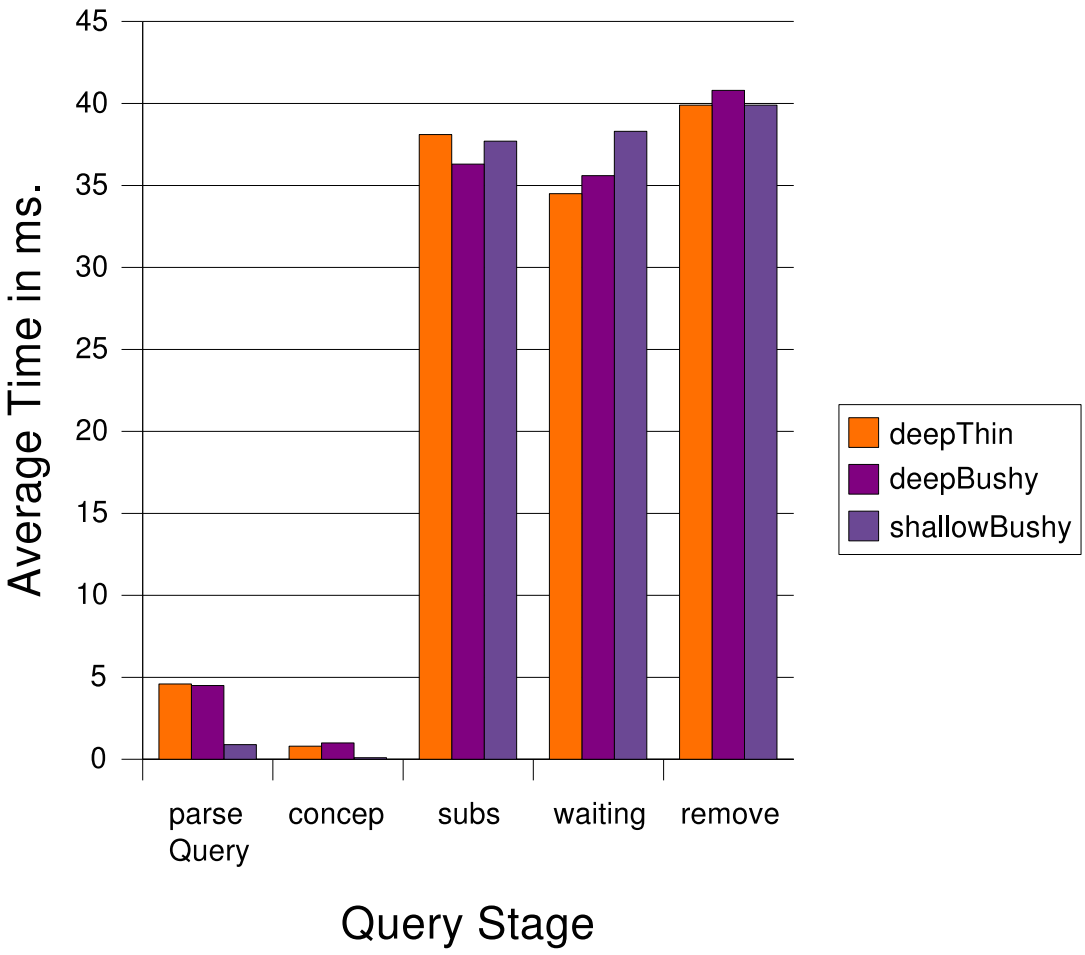

Figure 6.5: Local CSN query time graph

Model type	transQuery	eXist	transResult	notif	total
deepThin	1.4	7.1	0.4	0.4	13.9
(Std. dev.)	0.9	1.2	0.6	0.6	1.9
deepBushy	2.4	7.2	0.4	0.4	15.5
(Std. dev.)	0.8	1.1	0.5	0.6	1.7
shallowBushy	0.7	6.4	0.5	0.1	9.0
(Std. dev.)	.5	1.2	0.6	0.4	1.4
Averages	1.5	6.9	0.4	0.3	12.8

Table 6.2: Average remote CSN query stage times in ms.

The final element of the local CSN's operation is waiting for a notification to be received containing the response to the query. This waiting period includes time taken to route subscription information to the remote CSN, to extract and translate the query, perform the query, translate the result and return it using a notification.

The times taken during the operation of the remote CSN are shown in Table 6.2, and these figures are graphed in Figure 6.6.

Here we see that query translation times take less than 3ms, even on the deepBushy ontology containing 400 concepts. Sending the translated query to eXist (eXist) takes a consistent 7ms, while translating the result back to the model of the querying CSN takes a maximum of 0.5ms. Constructing and sending the notification back to the content-based network (notif) takes less than 0.5ms.

In total, the average time spent processing by the remote CSN amounts to 12.8ms. Comparing this to the average waiting time of the local CSN, which is 36.1ms, we can see that once the initial subscription has been made, the time taken for the content-based network communication (the sending of the query and the result) totals just 23.3ms.

A total of 20 simultaneous clients were used to investigate the subscription addition and removal process. Each of these clients continually queried a single CSN, in order to investigate performance with a large number of subscriptions operations being performed simultaneously. This experiment showed that the throughput rate of the system is controlled by the rate at which the CSN can perform subscribe, notify and remove subscription operations on the Elvin server. With the test Elvin

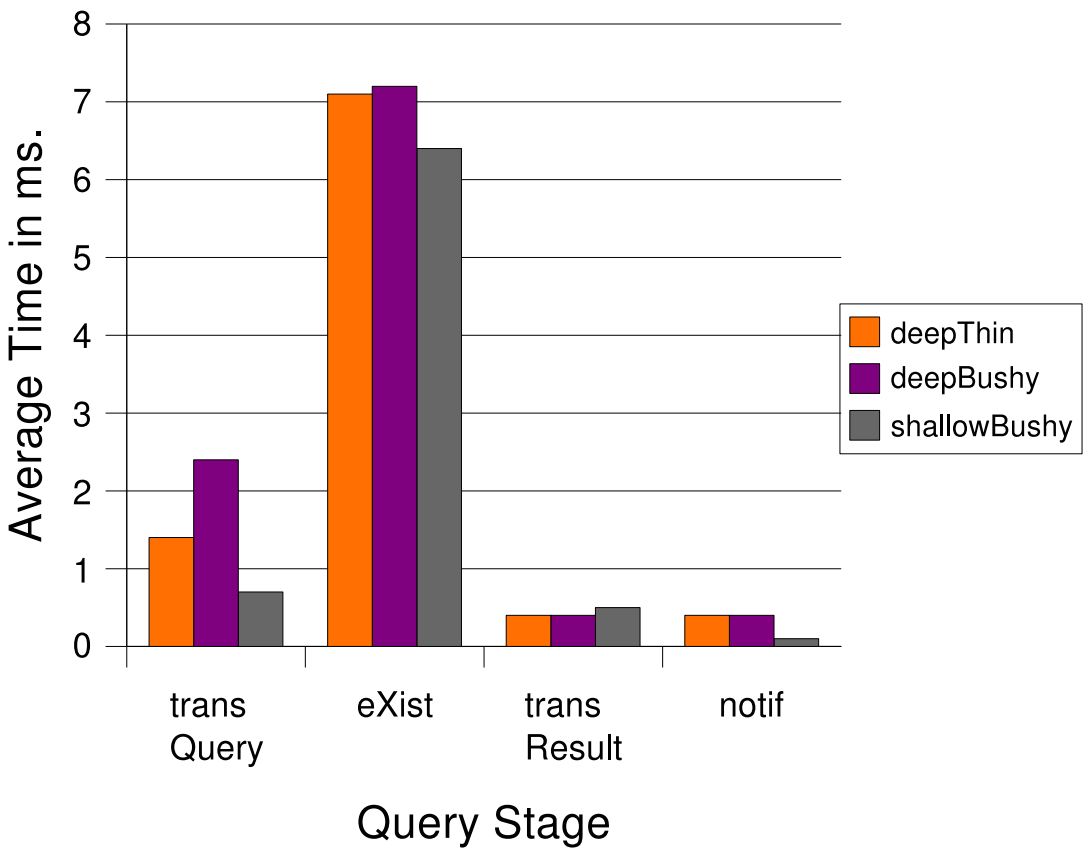

Figure 6.6: Remote CSN query time graph

server running on a Linux virtual machine, a maximum rate of between 16 and 18 subscribe/notify/remove cycles per second was the limiting factor.

Elvin Performance Experiment Findings

Looking at the results for the local CSN, we see the time taken to parse the query into an abstract syntax tree (AST) depends on the number of elements in the tree, as the time taken decreases dramatically for shallow ontology queries which have fewer elements. Concept extraction is also a lightweight operation, taking less than 1ms for queries over the generated context models.

Most of the time the local CSN spends in its query resolution cycle is taken up with Elvin operations—adding and removing a subscription. The time taken waiting for a response is actually only about one third of the total query time, with Elvin operations taking up the vast majority of the remaining time.

Efforts were made to test the Elvin server to see if it could be pushed past the cycle of 18 subscribe/notify/unsubscribe cycles per second, however this proved difficult. This is most likely due to the Elvin designers not considering that frequent subscriptions and unsubscriptions might be necessary for some applications, instead providing acceptable performance for the standard case of long-lived subscriptions matching notifications.

Although the current version of Elvin has this limitation on the rate of registering and removing subscriptions, an updated version of Elvin or alternate content-based networking software may remove this limitation. The current implementation is designed to work with Elvin's quench feature, so other content-based networks were not investigated.

Overall performance of the queries tested shows that Cashua can resolve these queries in approximately 120 ms, which is only slightly above the response time required in an application providing immediate response to the user. The current implementation would therefore be suitable for all applications except those which are highly time-sensitive, requiring immediate response to the user based on context

information.

This short query time allows a CSN to simultaneously handle a large number of clients without requiring large computing resources in order to provide timely responses. This will allow CSNs to be deployed on any device with a reasonably modern CPU. The main restriction on CSN deployment would be the presence of sufficient memory in order to support the current Java-based implementation. More memory is required to process larger ontology models and to support more simultaneous queries, as each query takes up a small piece of memory as its thread is running.

During the performance experiment 256MB of memory allocated to the development machine was sufficient to run two CSNs, the Elvin server, the test application and the remote database, showing that memory consumption is not excessive. Given that it is not unusual for current high-end mobile phones to have 64MB of RAM, it is conceivable that the current CSN implementation could run on a mobile phone to provide context to the phone's applications. If the CSN was rewritten to use a more memory-efficient language such as C, it would be even better able to take advantage of advances in device technology as they happen—allowing the CSN to be deployed on more and more devices as available resources are increased.

6.2.4 Isolated CSN Performance Experiment

Design

This part of the performance evaluation examined the case of the isolated CSN, processing queries against a service which simulates a fast content-based network. The CSN was completely separated from the content-based network in order to evaluate its performance in isolation from other factors. In order to do this, the CSN was connected to a service that simulated a content-based network, returning pre-prepared results immediately rather than waiting for the query to be resolved over a real network. In order to do this, the CSN was configured as shown in Figure 6.7.

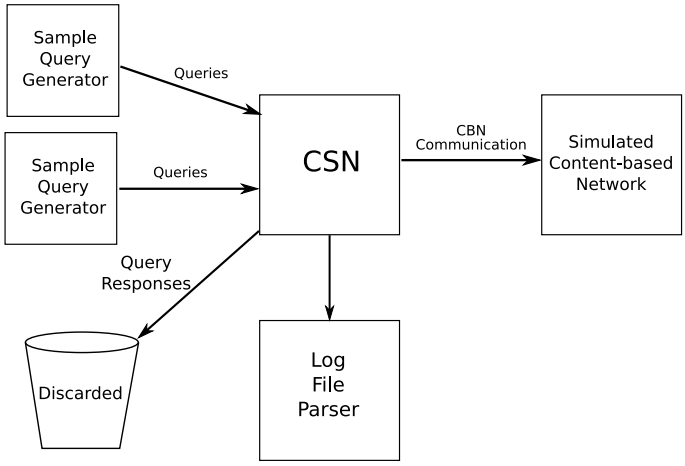

Figure 6.7: Isolated CSN test setup

The performance metrics described in Section 6.2.2 were collected from these experiments and were used to evaluate the CSN's query response time independently of the content-based network.

Despite the lack of a large-scale network of context-aware applications to query a CSN to test its scalability, an experiment was carried out to provide an indication of this scalability, in order to give an idea of the number of simultaneous applications which a CSN could support. This experiment consisted of 20 query generator applications which were configured to repeatedly query the CSN, sending their next query as soon as a query result was returned. These query generators each sent two thousand queries, and the average query response time, and also the query throughput was measured. While this experiment will not describe the CSN's performance when supporting massive numbers of clients, within the constraints of practicality it was considered a worthwhile test as it would indicate the CSN's scalability to this number of clients.

Results

Table 6.3 shows the average time taken for each stage of the two thousand queries in a test run with a single client, and this information is graphed in Figure 6.8. The stages here are the same as for the Elvin part of the performance experiment,

Model type	parseQuery	concept	subs	total
deepThin	4.87	0.8	0.24	6.19
Std. dev.	1.0	0.8	0.5	6.2
deepBushy	4.96	0.93	0.37	9.99
Std. dev.	6.0	0.8	3.4	8.9
shallowBushy	1.0	0.17	0.03	1.75
Std. dev.	3.6	0.5	0.2	1.7
Averages	3.61	0.63	0.21	6.0

Table 6.3: Isolated CSN local query times

except that here the subscription time (subs) refers to the time taken to compose the subscription, not to register it with the network. This is because the application simulating the content-based network returns immediately rather than taking time to register the subscription as a true content-based network server would.

These results show that query parsing (parseQuery) times are approximately the same as in the Elvin case, as are the concept extraction (concept) times. Generating the subscription takes less than half a millisecond, and in this experiment the subscription does not need to be registered with the content-based network which reduces the 'subs' time in the result.

The average total time across the three ontology types was 6ms. This shows that in a scenario where a fast content-based network exists, and subscription operations would occur almost instantaneously, a CSN could support a query rate of approximately 160 queries per second.

When the number of simultaneous queries was increased by using 20 query generators, each querying simultaneously, the average query time rose to just 8ms. The CSN processed these queries at a combined rate of 120 per second.

Analysis

This part of the experiment shows that the majority of the time taken during query resolution is consumed by subscription management on the Elvin server. When the simulated content-based network is configured to immediately process subscriptions and subscription removals, the average total time taken for queries goes from 119ms

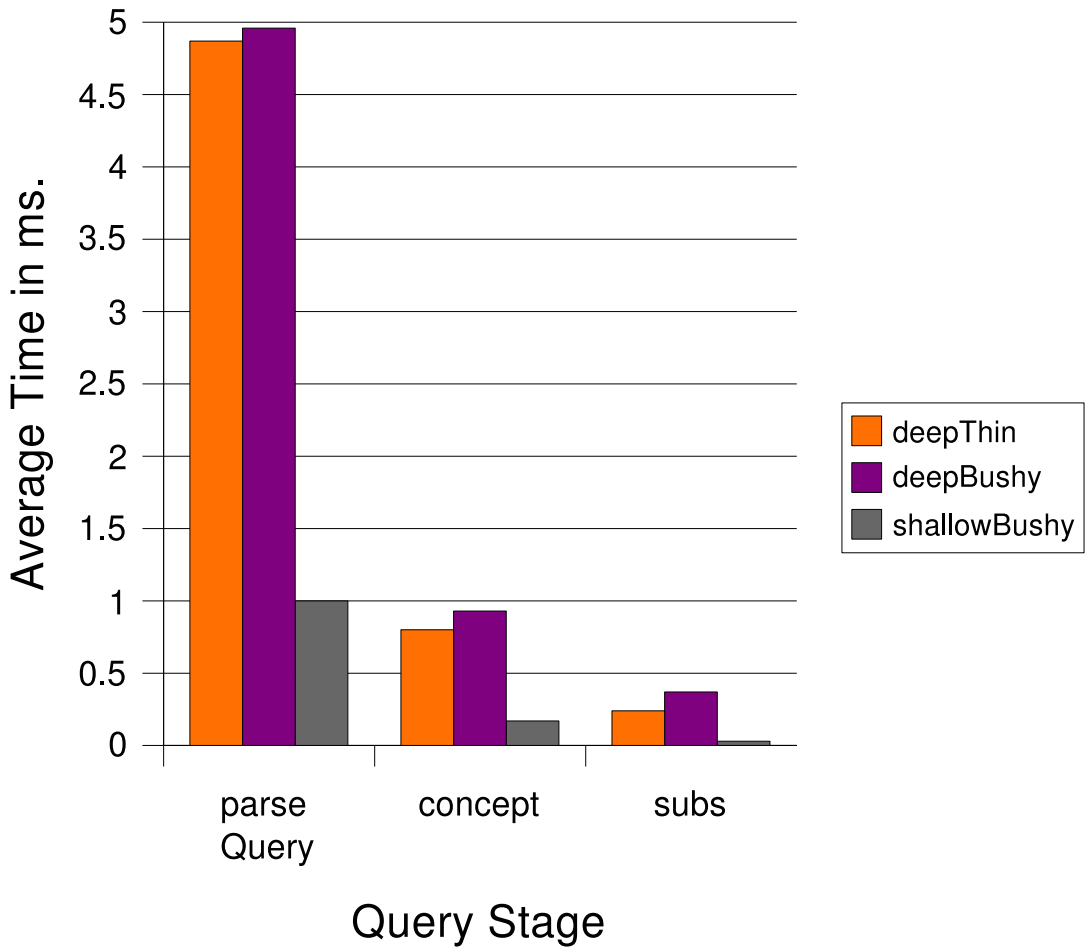

Figure 6.8: Isolated local CSN query time graph

down to 6ms.

With each query taking an average just 6ms to process on the CSN, we see that the processing requirements for the CSN, excluding the Elvin operations, are quite small. The CSN is capable of resolving approximately 160 queries per second when the queries are sent serially. The experiment using 20 query generators acting as clients showed a large degree of parallelism which is possible in the CSN, allowing it to scale to support more applications without dramatically affecting the speed at which it resolves queries. While not a large-scale experiment, a combined rate of 120 queries per second on the isolated CSN experiment when using 20 query generators demonstrates that the CSN scales to at least this number of clients.

This high rate of query resolution would allow the CSN to support many clients simultaneously, where it is deployed on hardware with sufficient memory to manage the registered context models and simultaneous queries. This support for many clients allows for fewer CSNs to be deployed in environments where more powerful devices are common. Depending on the query rate of clients, a small number of CSNs may be capable of serving context to a large environment. However, care must be taken not to reintroduce the weaknesses of a centralised environment by relying on a small number of CSNs on powerful hardware, which are a point of failure in the system.

6.2.5 Overall Performance Experiment Analysis

The performance experiment shows that the local and remote CSNs can quickly resolve a large number of queries without consuming large amounts of resources. The time taken by the CSN to extract concepts, translate queries, and so on was a small fraction of the time spent performing CSN operations, such as registering and removing subscriptions.

Although the use of Elvin limits the rate of subscription addition and removal, improvements to content-based networking technology will help to remove this limitation in future. Research is already underway in managing content-based

networks with highly dynamic subscription sets, such as in [47] which uses a probabilistic method to quickly determine covering relations over large numbers of subscriptions.

The isolated CSN experiment showed that the low amount of processing required for each query will make it possible to deploy CSNs on resource-constrained devices, such as mobile devices. Although these devices themselves have little processing power, if they can support a CSN they can hand the task of query and response routing to the content-based network. The content-based network should be considered part of the infrastructure of the environment, and run on more powerful hardware. Using this hardware to speed up the processing of subscription addition and removal would allow queries to be resolved more quickly.

One issue not directly measured in the performance experiment is the performance of Cashua with large ontologies which contain many relations between concepts which are not used by Cashua. As described in Section 4.1, Cashua uses superclass/subclass relations between concepts in order to navigate the concept hierarchy, however additional relations such as constraints and properties are not utilized by Cashua. These relations would add to the CSN's processing time when loading a particular ontology model, which is done as the CSN is started up or a new model is registered. However, these additional relations would not have a large impact at query resolution time as the concept model is pre-computed and does not need to be recomputed for each query. Search time for a query would scale according to the ontology processing library's search time for a concept inside the ontology model.

6.3 Mapping Distribution Experiment

6.3.1 Overview

The ontology mappings that allow translation between two context models are a vital tool in overcoming the heterogeneity that exists between producers and consumers

of context. The automatic distribution of these mappings allows CSNs to make use of mappings without requiring a human to manually register them at the CSN. The objective of this experiment is to evaluate the correct function of the mechanism for distributing mappings between CSNs when they are needed. The performance of the mapping distribution was not considered, as mapping distribution is a much less frequent operation than query resolution and will therefore have little impact on the overall performance of the system.

The methodology used in this experiment is a practical deployment of Cashua, using a configuration of CSNs, applications, models and mappings as described in Section 6.3.2. In the experiment's initial state, mapping information is available to some but not all CSNs. Through the use of a query on the content-based network, a CSN without this mapping information can express interest in it. Once remote CSNs are aware of this interest, they publish the address where the required mapping information can be found. Cashua's mapping distribution was evaluated by confirming its correct operation in transferring mappings to where they were needed.

6.3.2 Design

The mapping distribution experiment consists of multiple CSNs connected via a content-based network. Connected to these CSNs are context producers and consumers. Here the context consumer is a context-aware application labelled A, accessing information from a database B. The context models used by A and B are different, and are labelled X and Y respectively. The initial state of the experiment is outlined in Figure 6.9.

At this stage, A and B are connected via CSNs communicating over the content-based network indicated by the dashed line. Consumer A uses context model X, so it registers this with its CSN and the concepts in X are published on the network, as indicated by the X below the CSN. Meanwhile, producer B has registered model Y with its CSN, but also has access to a mapping between the

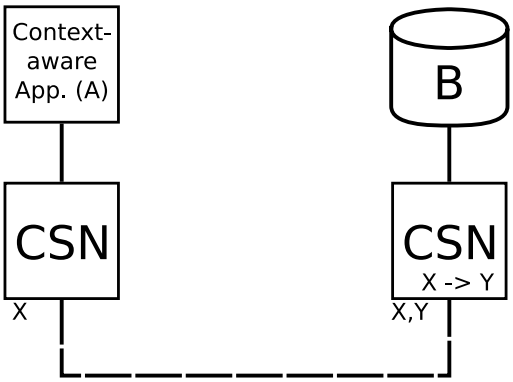

Figure 6.9: Initial state of mapping experiment

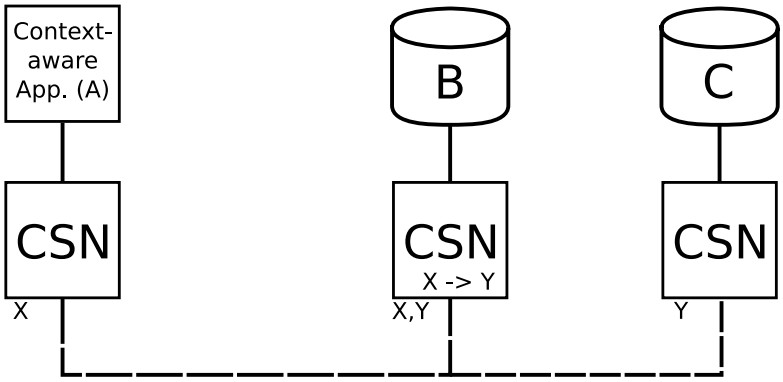

Figure 6.10: Producer C is added to the network

concepts of X and Y. This mapping is represented by X→Y in the figure.

The first part of the experiment investigates transferring mappings to a CSN newly joining the network. Context producer C is added to the network, as shown in Figure 6.10. This producer also uses model Y, and has registered this with its CSN. However, the CSN does not have access to the mapping between X and Y, so this is not registered here.

The CSN queries the network for any mappings that match the concepts in model Y, using a mapping discovery query. This mapping discovery query matches the concepts published by producer B's CSN as it has the X→Y mapping, so it sends the mapping encapsulated in a notification. The mapping is now available on C's CSN, as showing in Figure 6.11.

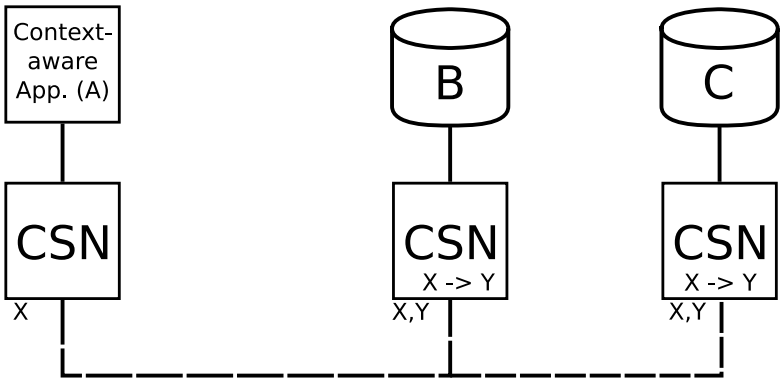

Figure 6.11: Mapping is transferred to producer C

For the second part of the experiment, we examine the mapping transfer to consumer A's CSN for later use. This CSN is already present on the network, and the mapping was not available when it did its initial mapping discovery query. When consumer A sends a query to the network which is translated by the remote CSN, it has the option of also asking the CSN which translated it to return the URL of the mapping used to perform the translation. If the CSN does not have this mapping stored in its database, it can retrieve the mapping via a mapping discovery process.

6.3.3 Mapping Distribution Results

In this experiment two types of mapping discovery query were considered—a discovery as the CSN was attached to the network, and later discovery and exchange of mappings so they can be cached for future use. Querying for mappings as the CSN is connected to the content-based network allows the CSN to make use of mappings already present in the network, saving on effort of manually distributing mappings.

The experiment showed that only one discovery query was required per CSN joining the network, and the size of this query was proportional to the number of concepts registered with the CSN. Mapping URLs returned over the content-based network were correctly downloaded and saved to the mapping database. Smaller mappings were distributed via the content-based network, rather than downloaded via a URL, however the use of a URL for download removes more traffic from the network,

allowing for more efficient querying.

6.3.4 Mapping Distribution Findings

The mapping distribution mechanism allows for mappings to be automatically distributed, removing the need for mappings to be loaded manually at all CSNs where they can be used.

The mapping distribution mechanism complements standard query distribution, by making use of previously published quenches to route the mapping discovery queries. Mapping discovery both as the CSN connects to the network and via translated queries allows mappings to be transmitted to the nodes that need them, without burdening the network with large volumes of discovery queries. The use of the standard quench mechanism to distribute the mapping discovery queries means that no additional network traffic is required until a new CSN is connected. Allowing the URLs of mappings to be transmitted instead of the mapping itself also reduces the amount of network traffic used, and allows CSNs to retrieve these mappings for later use.

The trustworthiness of these discovered mappings is not considered, however the CSN could be extended to treat mappings differently based on their source, or the presence of a digital signature. For example, a CSN might choose to use untrusted mapping information only when no other mappings are available, in order to protect the user from spurious mappings which might be injected into the network.

The use of mapping distribution mechanisms is important as it reduces the manual burden on integrators of distributing mapping information to where it is needed on the network. Automatic mapping distribution allows integrators to supply the information once to a single CSN, yet have it automatically redistributed to where it is needed. This is a necessary mechanism for networks of mobile hosts, which can reregister their model information at a new CSN when they move location, but need to use mapping information which may be present on the network at another location.

This automatic distribution feature will reduce the workload on integrators, as Cashua does not require that mapping information be supplied at each CSN where it is to be used. Additionally, it removes the need for application developers to discover mapping information for use by their context-aware application.

6.4 Developer Case Studies

One of the objectives of the evaluation was to evaluate the feasibility of this approach from the application developer's point of view. The creation of context-aware applications must be made as simple as possible in order to promote the use of context in ubiquitous computing. If making an application context-aware is too difficult or time-consuming, developers may choose not to make use of context. These case studies indicate that developers found Cashua easy to use and were able to integrate their applications in a reasonable time, without needing to modify their source code.

6.4.1 Overview

The usability requirements of this system centre around two features. The first is the time and effort required to let an application access context information. The second is the insulation of the developer from the environment to which the application will be deployed. A developer must not be required to tailor the software for a particular environment, as this limits its reusability. Instead, the developer should design their software and context model to be independent of the environment in which it will be run. The goal of this evaluation is to assess how Cashua functions in practice to provide these features.

6.4.2 Design

This experiment consisted of making two large software projects context-aware—the Adaptive Engine and the Interlocutor. These projects are described in Section 6.4.2

and Appendix D respectively.

Adaptive Engine

The Adaptive Engine[23], developed within our own research group, adaptively delivers online courses to learners based on their features such as the learner's prior knowledge and preferred learning styles.

This experiment allows the Adaptive Engine (AE) and the SQL course to make use of context information from a Context Service Node (CSN). The standard AE is not context-aware, and its information must be explicitly specified by the user. The learner model was identified as the best candidate for being retrieved from context servers.

The Adaptive Engine makes use of three main sources of information, from which it retrieves the narrative, the learner model and the content model. These could be implemented as three separate databases, but in practice they all come from the same XML database.

The learner model is composed by taking a pre-test which identifies the concepts that the user wishes to learn. From this the learner model is composed. This personal information is filled in by the user at the start of the course, though may not be available. Because this information must be manually entered, it is limited in its scope. When the Adaptive Engine reads in the narrative, learner model and content model, it makes its decisions based on what it finds in the learner model for this particular learner.

Allowing the AE to retrieve its information from a context service greatly benefits the learner for a number of reasons. Firstly, the pre-test might not be required, allowing the user to commence the course immediately with their preferences already in place. Secondly, the AE could make use of more detailed information about the learner (such as competencies learned and preferences) from other Adaptive Engine based courses distributed across the network.

If this context information is easy to obtain, the AE might also query for extra

information that it would never consider worth asking the learner for in a pre-test. This would allow for a richer, more detailed learner model and hence the AE has the option of performing more intelligent adaptations based on this information.

A key feature is that the AE will not need any knowledge of where the learner model information is coming from. It is the task of integration (separate from the software design of the AE) that establishes the location of context information, allowing the CSN to provide that information to querying applications, in this case the AE.

In the case of a context-aware AE, the person performing the integration process will take the data model and ontology provided by the AE designers, and establish mappings to other learner models distributed around the network. In this experiment the learner models will be largely homogeneous, but more complex mappings could be composed to deal with independently-developed models.

6.4.3 Methodology

The methodology used for the developer questionnaires was as follows. Firstly, the developers were asked to integrate their applications with the CSN by performing the technical integration. This consisted of passing their application's queries to the XML-RPC interface of the CSN, instead of to a local database.

Once this integration had been performed, the application developer was given the developer questionnaire. This questionnaire had a number of aims. Firstly, it collected information about the developer's application, its data model and the queries it poses. Next, it ascertained the developer's experience with XML technologies and ontology languages.

Once this background information was collected, the questionnaire instructed the developer on how to create an ontology representing their application's data model using Protégé, and deploy this ontology to a CSN. It then asked questions to determine the software development time required, and whether any changes were required to the application.

6.4.4 Results

The results of the developer case studies are shown in the appendix. Firstly, the blank questionnaire, as it was presented to the developers, is shown in Appendix B. This includes the instructions to developers and ontology primer. The completed questionnaire for the Adaptive Engine is presented in Appendix C, and for the Interlocutor in Appendix D. These do not repeat the instructions to developers, for clarity.

6.4.5 Findings

The developer case studies showed that with very straighforward instructions, developers could take existing software and make it context-aware through Cashua. These developers had good general knowledge of programming and XML technologies, but were not experts in ontologies. Even for a complicated data model, averaging 4KB to 8KB in size for an XML instance of the model, it was possible for the developer to identify the concepts he wished to retrieve using Cashua. The developer used these concepts to create a concept model which was passed to the CSN, where it was used to resolve queries. The Adaptive Engine (AE) developer was able to perform these tasks though he had not previously developed his own complete ontologies:

> I have a good conceptual understanding of Ontologies and have used them as part of some simple systems, however I have not developed my own ontologies. (1 year)

The Interlocutor developer was able to perform this task without any previous direct experience of ontologies:

> No Direct experience, good comfort level, I have read and attend several presentations on this topic.

Allowing the developer to express a portion of his data model as a concept model, rather than requiring the complete data model to be expressed, allows the developer the flexibility to decide what information can be retrieved from context. This allows him to start with a fixed, small part of the model for which he knows valuable information exists on the network, and extend this model as his understanding of the available context information expands, and more context sources become available.

The technical instruction for developers to integrate with Cashua is quite straightforward, requiring less than half a page of text. The developer can replace existing database queries (which may already happen over the network) with a query to a CSN, while maintaining the same query text. This allows integration with a reasonably small amount effort outside the modelling tasks already outlined.

The integration with the Adaptive Engine demonstrated the use of queries to retrieve the learner model information of students. An Adaptive Engine course was configured to retrieve students' prior knowledge and learning style information through Cashua. This demonstrated a use case for querying the network for available information about the student, in order to tailor the course to the student's needs. The Interlocutor integration demonstrated the use of a small amount of mood information, which was required in a timely manner by an instant messaging (IM) application. This experiment also utilized non-blocking queries which returned results to the interlocutor as soon as changes were made to the information, removing the need to poll for updates.

In both the Adaptive Engine and Interlocutor integrations, the time taken to perform the technical integration—adding an interface to their software to communicate with Cashua, via XML-RPC or a SOAP wrapper provided as an alternative—was longer than the time required to perform the context modelling for the application. While this is partly due to the fact that these case studies only retrieved a small part of their data model through Cashua, it does demonstrate that the fundamentals of the integration process are straightforward and do not require intimate knowledge of how Cashua or ontologies work. Larger context models would have more concepts, taking more time to produce, however they are formed using exactly the same method as

smaller context models.

Both developers were typical in their knowledge of XML-based technologies, however due to their position within the research group they also had some introductory knowledge of the purpose of ontologies. This knowledge was not particularly deep however, and neither developer had previously developed a complete ontology. Establishing the concept hierarchy and using Protégé to express this model as an ontology took between 5 and 15 minutes once the aim of the exercise had been communicated via the questionnaire.

These case studies demonstrate that this approach is feasible, and does not put large programming burdens on developers. They demonstrate that developers can choose to create a context model using a subset of their application's data model, allowing them to get up and running quickly with using context. This feature will have impact on the adoption of context-aware applications, supporting rapid prototyping and dynamic evolution of available context information.

The application integrations described in this section were sufficient to evaluate the usability of Cashua for the purposes of this thesis. However if more time and resources were available, it would be beneficial to extend these experiments to evaluate integrating larger applications with more complex context models and diverse queries. These larger experiments could provide more information about how to improve the integration process.

6.5 Evaluation of the Elvin platform

From a programming perspective, the Elvin API was quite simple to understand and program against once the appropriate client libraries were in place. Within a short amount of time it was possible to have an application query against the Elivn server. The server itself was easy to use, worked reliably and provided low-latency delivery of notifications to clients in the experiments.

One limitation found during the course of this work was the rate at which

subscriptions could be registered and deregistered on the server. This is most likely because the Elvin developers understandably did not consider that subscriptions might be registered and removed at a high rate, so did not particularly optimize the subscription registration code for performance, but rather concentrated on a typical usage pattern where the number of notifications far exceeds the number of registered subscriptions.

As Elvin is a closed source system, optimizing the code was not possible however future work might consider subscription performance in other content-based networking systems.

6.6 Related Work

6.6.1 Evaluation of Cashua Against Requirements

In order to compare Cashua with the systems presented in the state of the art chapter, the requirements framework presented in Section 2.6 is used. Table 6.4 shows how Cashua compares with the systems described in the state of the art chapter, using this framework.

In this table the hyphen in Rebeca's 'Conversion' row indicates that it supports a limited form of model conversion, namely converting between models in different Rebeca scopes, but not within scopes.

Feature	CoBrA	SCS	Rebeca	C. Spaces	Fulcrum	MoGATU	Cashua
Context Specification	Y	Y	Y	Y	Y	Y	Y
Comm. Abstraction	Y	Y	Y	Y	Y	Y	Y
Resource Discovery	N	Y	Y	Y	Y	Y	Y
Conversion	N	N	-	N	N	N	Y
Network Comm./Avail.	Y	Y	Y	Y	Y	Y	Y

Table 6.4: Comparison of Cashua with State of the Art Systems

This table shows visually that Cashua meets all the requirements as outlined in the framework in Section 2.6. In contrast, none of the other systems described offer any form of conversion between context models, which is a fundamental part of Cashua's coping with heterogeneity, and a vital feature for context systems. Cashua is now analysed along these axes.

Context specification Cashua provides a formal method of context specification, through the use of ontology models of context. These models, provided by the developer, are used by Cashua to extract the information needed to correctly route queries and results. Models registered by context sources are processed to extract a description of the concepts that the source can answer queries about, and these concepts are published on the network. Application queries are also processed to determine which concepts from these models they refer to, and this concept information is used in the query routing.

The use of ontology models for context specification is vital to the operation of Cashua, providing a formal mechanism for specifying context models. These models are machine-understandable, allowing the CSN and its underlying content-based network to perform query and result routing without human intervention once the models have been provided.

The use of ontologies exploits established work by the semantic web community in domain description. Ontology concepts and the relationships between them allow Cashua to deliver relevant context information to applications.

Communication abstraction Cashua's infrastructure model provides a communication abstraction for the retrieval of context. Each application and data source only needs to communicate with its local CSN, which is responsible for communicating with other CSNs to resolve queries.

This infrastructure model facilitates reuse and maintainability within the system, as applications are not responsible for communicating directly with their data sources. They therefore do not need to know the communication protocols, network location,

data model or any other information about their sources. This abstraction is provided by the CSN, effectively hiding these details from applications.

The most important aspect of this abstraction is that applications and data sources can be upgraded independently of each other, provided that they maintain the same interface to the CSN.

Resource discovery The dynamic routing of queries to data sources based on published context information provides a method for applications to discover what context is available on the network. This resource discovery is built in to the query mechanism, so queries can be directed and resolved without needing to perform a separate resource discovery step.

This integrated mechanism also allows resources to be dynamically added and removed from the network without needing periodic rediscovery to make them available. Once they register with their CSN, the CSN publishes their availability on the content-based network automatically.

The Adaptive Engine experiment most clearly demonstrated the benefits of resource discovery, as new learner information was discovered (by routing of queries) and that information was used by the Adaptive Engine without it needing to be aware of its source.

Conversion and Interpretation One of the main functions of the CSN is to convert context between different models used by the producer and consumer, thereby overcoming the semantic differences between them. The CSN performs this conversion using mappings between models which are provided to it by an integrator.

Cashua's conversion between ontology models does not support full conversion between models of arbitrarily different structure, as full conversion between arbitrary models is a research topic in itself, and beyond the scope of this work. Cashua does however use OWL equivalence expressions in order to express mappings between individual concepts in the model, and can use additional relations inferred from superclass and subclass relationships to query at different levels of abstraction.

More complex interpretation mechanisms, tailored to ubiquitous computing environments, could be added. These might include proximity between entities based on their coordinates, or adding fuzziness to context results when queried by untrusted applications. However, the interpretation supported by Cashua is sufficient to demonstrate the concept, and more complex interpretation mechanisms are left as future work.

This mapping approach removes the need for context to be expressed using a global schema, as mappings between individual models are supported. Cashua mappings are expressed in ontology format to allow them to be imported from other sources and also reused when possible, making most effective use of human effort in composing these mappings.

Network communication and availability Cashua's content-based network communication layer is independent of the CSNs registered with it. Cashua handles all inter-CSN communication on behalf of the application, allowing queries to be transparently routed to multiple providers. All providers can then process the query and return their results, providing the most information possible to the application.

This configuration also allows queries to continue functioning when CSNs are removed from the network. When this removal happens the content-based network routes are updated automatically, allowing Cashua to continue to operate without needing context producers or consumers to be modified.

6.6.2 State of the Art Evaluation

This section evaluates Cashua with respect to the state of the art, along axes other than those requirements addressed in the requirements framework in Section 2.6.

Separation of Developer and Integrator

The design of Cashua is based on the premise that context-aware software will be written once and deployed many times, as is currently the case with conventional

software. With this fact in mind, Cashua was designed so that context models and mappings between them are be expressed separately, and can be deployed by different people.

The issues of integration are avoided with systems that require applications to commit to shared ontologies to communicate, such as CoBrA, Fulcrum, Context Spaces and MoGATU, where all agents share a common ontology with the context service. However using a common ontology does not permit heterogeneity in the system, as developers must perform integration themselves, at the design time of the application.

Some heterogeneity in context models is permitted by SCS, where developers may extend an upper-level ontology with their own domain-specific ontology. The best system in this regard is Rebeca, where a 'scope administrator' can manage conversion between applications in different scopes, and register mappings between ontologies.

Because of Cashua's support of mappings, an application developer can write their application and its associated context model (as an ontology), and distribute this as a self-contained package. An integrator can then take the ontology model and construct a mapping between this model and the model of the context source and register the mapping with the CSN. Once the context model and mapping are registered with the CSN, it can begin resolving queries.

The advantage of this separation is that the integrator does not need to collaborate with the developer in order to integrate the application into the environment. The integrator can compose the mapping by inspection of the context models of producer and consumer, rather than needing to customize the data model of either one to match the other. This customization would be very expensive, as it would require the developer to individually tailor the application to each deployment environment. Instead, mappings are used to bridge the semantic differences between data models.

This separation was demonstrated in the Adaptive Engine and Interlocutor experiments, as these existing complex pieces of software were integrated with remote data sources without requiring their developers to interact with an integrator

to perform the task. Instead, the integrator could simply take relate two context models without needing to know the in-depth operation of the software in question.

Intelligent Routing

The use of content-based networking allows for intelligent context routing based on available information in the network. Cashua intelligently routes queries to those hosts which have registered interest in particular concepts. This allows queries to return the most information possible, without flooding the network by sending queries to all hosts. Cashua's intelligent, distributed routing based on available information is a contribution to the current state of the art.

Centralized systems such as CoBrA obviously don't have this routing requirement, as they rely on a central server at the expense of the performance issues this can cause. Other systems divide peers into categories and route messages based on the categories the peers are part of—SCS routes messages between 'spaces', while Rebeca routes according to the 'scope' of a message. These classifications are coarse-grained, and rely on the correct selection of a scope for the peer and messages destined for the members of the scope.

Other systems use existing peer-to-peer messaging such as Context Spaces, which uses the JXTA[38] to communicate between peers. MoGATU uses its own custom message routing, where advertisements and subscriptions are forwarded to the one-hop neighbours of InforMAs in order to route messages. The most powerful routing mechanism of the systems surveyed is used by Fulcrum. Fulcrum peers use active subscriptions on the Siena content-based network to subscribe to events of interest to them.

A feature of Cashua's use of content-based networking is that queries which can be resolved locally are only distributed locally. This intelligent routing helps to reduce the amount of traffic on the network, and thereby make the system more scalable.

Cashua's routing is also dynamic in nature, as the system reconfigures itself based on available context model information and active queries. This intelligent routing

allows administrators, developers and integrators to configure the system's routing simply by updating model information. Configuring Cashua at the level of models and mappings allows developers and integrators to consider the system from a more abstract information model level, while the low-level routing configuration is done automatically based on these models.

Automatic Mapping Distribution

Cashua's automatic distribution of mappings allows for CSNs to make use of mappings that have not been registered directly with them, but have been discovered on the network. This mapping distribution is an extension of the resource discovery requirement, as discovering mappings permits heterogeneity in the network while reducing the amount of human effort required to maintain the connections between context producers and consumers.

The systems surveyed have no automatic mapping distribution mechanism. Those which only support a single ontology (CoBrA, Context Spaces, Fulcrum and MoGATU) obviously have no mappings. SCS supports mappings between upper and lower ontologies only, not between domain-specific ontologies. Rebeca supports mappings, but has no method for distributing them automatically to where they are needed.

Familiar Query Language

Cashua uses XPath, an XML query language which is already familiar to developers and does not require a change in how applications express their data model. This is naturally the case for applications which already use XML data, however converting other data models such as relational databases to XML format is possible.

CoBrA and SCS use RDQL as their query language, while MoGATU uses an RDF-based query specification language, which expresses additional constraints such as cardinality and temporal constraints on the query. Rebeca and Fulcrum are event-based systems so instead of queries, clients register to receive events of a

particular type. In Context Spaces, XPath is used to select context elements from a global ontology.

XML is already the language of choice for inter-machine communication, and the choice of XML ontology models and XPath queries over data expressed in those models is the natural choice.

6.6.3 Summary

The evaluation presented in this chapter considers the performance of Cashua in providing context to applications, the feasibility of this approach for software developers, its comparison against the requirements framework, and also the features it offers with respect to other state of the art systems.

The performance experiments demonstrate that Cashua can provide context in a timely manner to applications, without requiring these applications to have knowledge of the location or format of the context information they are retrieving. Cashua's design allows it to manage these details on behalf of the applications, allowing it to provide a valuable service to applications that wish to avail of context information without requiring them to manage access to this information themselves.

The two case studies undertaken demonstrate the feasibility of the approach from the point of view of an application developer. The results of the questionnaires show that a developer can integrate with the Cashua system in a short time, with only a basic knowledge of ontology concepts and languages.

Evaluation of Cashua shows that it provides all the features identified in the requirements framework in Section 2.6. It also offers several features in addition to these requirements. Its design allows for the separation of the roles of application developer and integrator, allowing software to be written once and deployed many times into different environments.

Cashua provides intelligent routing of context information, routing queries to where they can be answered automatically based on available descriptions of context. This intelligent routing allows Cashua's design to be applied to dynamic networks,

where statically routed systems are impractical. Cashua also supports automatic distribution of ontology mappings, providing a means for the context system to integrate heterogeneous applications. Finally, Cashua's use of the XPath query language provides a familiar query language for application developers, promoting the adoption of the system by not requiring that applications refactor their applications or the query language used.

Chapter 7

Conclusion

7.1 Objectives

The objectives of this work are introduced in Section 1.2.

- The first objective was to conduct a survey on existing approaches to context modelling and the routing of context information in context systems, to establish the state of the art in the area. This survey identified lessons learned by others when building context management systems.

- The second objective was to establish mechanisms to be used by a context management system to address the challenges of dynamism and heterogeneity in ubiquitous computing environments. A survey was undertaken to evaluate technologies for addressing these challenges. Content-based networking and ontology modelling mechanisms were selected to form the basis of the design of the Cashua system, which was then implemented.

- The final objective was to evaluate the integration of ontology and content-based networking through a set of experiments. These experiments would examine key performance metrics and investigate the feasibility of the approach.

The work presented in this thesis has met these objectives, by describing a context management system which provides a novel mechanism for distributing context in a ubiquitous computing environment. By surveying background systems, a requirements framework was developed which was then used to evaluate state of the art systems. The framework assisted this evaluation, identifying a gap in the state of the art for a system which provided support for both dynamism and heterogeneity.

Technology selection was carried out along these axes, selecting content-based networking and ontologies as the mechanisms for providing these features. While the list of technologies considered was not exhaustive, it considered key state of the art approaches to providing these features. The technologies were integrated to form the design for the Cashua system.

A number of features of this integration were evaluated, demonstrating both the practical performance of Cashua in distributing context and the feasibility of its approach from the point of view of an application developer. Practical constraints limited the scale to which the performance could be tested, however the performance evaluation shows acceptable performance in terms of latency and query rate with up to 20 simultaneous clients. The two case studies carried out indicate that this approach is usable for developers, and demonstrate real applications which have been integrated with Cashua. The experience of the integrator is not considered, however this role consists of providing mappings between ontologies, which already an established field of research with tools available to assist in this task. Additional features were evaluated through reference to state of the art approaches to this area.

7.1.1 Objective 1

The background systems presented in Chapter 2 were pioneer systems in the field of context-aware systems, and typically concentrated on a small number of issues in context-aware computing. Combining the lessons learned by these systems into the requirements framework allowed their research to be aggregated in order to describe features that should be present in a modern context management system.

Given that Cashua would support heterogeneous applications, a survey was conducted of context modelling approaches, in order to investigate the types of context models that might be used in the system. This survey discovered a wide range of existing context models at different levels of expressivity, and tailored for different needs. This confirmed the belief that these heterogeneous models must be supported rather than assuming or enforcing homogeneity in the network.

State of the art context systems were also compared using the requirements framework, which indicated that modern systems elected to focus on either the issues of dynamism or heterogeneity, but did not provide an integrated solution to both these issues, instead ruling them out of scope. These systems typically chose to focus on dynamism rather than heterogeneity.

Additional requirements beyond those identified in the literature were also identified, and added to Cashua's design requirements. Distributed context storage, familiar languages for expressing and querying context and the separation of the roles of developer and integrator provide useful features for a modern context system.

7.1.2 Objective 2

The second objective was to establish mechanisms for dealing with the dynamism and heterogeneity. A number of possible solutions for these issues were surveyed, including Topic Maps for heterogeneity and traditional peer-to-peer approaches for dynamic networks. From these, a content-based networking and ontology models for context were selected as appropriate mechanisms.

Ontologies are a basis for interoperation, providing a formal method of expressing the context models of applications. These models are used by Cashua to identify where context about particular concepts exists in the environment, and also as part of mappings that specify equivalences between concepts. Expressing context models and mappings as ontologies has advantages in that these ontologies can be reused. This applies to developers who wish to avoid writing their own context model, to integrators who wish to reuse existing mappings rather than writing their own and

also to Cashua, which distributes mapping information automatically when it is needed in the network. The use of ontologies also exploits a wide range of existing tools in the area, supporting ontology creation and management, and also mapping.

Using ontologies does add an additional step to the process of writing a context-aware application, requiring developers to either compose their own ontology or find an ontology which they can reuse. Integrators must also provide mapping information in order to deploy an application which has no existing mapping to the model of a data source. There is naturally also some processing overhead to the use of ontologies, as investigated in the performance experiment. However, the use of ontologies allows Cashua to support heterogeneity which could not otherwise be considered, and also supports the routing of context information in the system.

Cashua's use of content-based networking provides routing of context information, which exploits the ability of content-based networks to route on the content of messages to route queries to where they can be answered, and return results accordingly. The content-based network manages the update of routing information as new applications join and leave the network and their CSN publishes its updated information to the network. The publish-subscribe nature of content-based networks is also well suited to providing long-lived queries, which return results over time as information is updated.

Using a content-based network for the distribution of context information requires that this network exists in the environment, where it must be maintained by a local administrator. However, Cashua does not require exclusive access and this network can also be used by other applications at the same time. The use of quenches limits the current implementation to content-based networks which support this feature, however alternative mechanisms exist to provide similar functionality.

7.1.3 Objective 3

The evaluation objective was met through gathering quantifiable performance information, through developer case studies and through discursive evaluation

against the design requirements of the system.

The performance experiments used multiple context models to test how the performance results varied with these models. These models were automatically generated in order to control their characteristics. This gave insight into how these characteristics affected the system's performance, however a more comprehensive investigation of these models might take other 'real world' models and measure their performance. This would require correlating which characteristics have an impact on performance, while isolating those characteristics which have no impact. This is more difficult when using model where the characteristics cannot be changed programmatically.

The performance experiment also considered the operation of the CSN independent from the content-based network, in order to isolate its performance from the performance of the content-based server. An issue with the rate at which subscriptions could be added and removed was discovered with the current choice of content-based server, suggesting the investigation of other servers as possible future work.

The developer case studies determined the technical effort required to integrate with Cashua, and indicated that the system was relatively easy to use for developers who were not experts in the area of ontologies. These developers demonstrated the feasibility of the approach by integrating existing applications into the system, without requiring changes in how their applications operated. The number of case studies could be expanded in future, to gather more experiences from other developers and to suggest directions for future growth.

Cashua was also reviewed with respect to the requirements framework, in order to establish that it met its design requirements. Cashua met all of these requirements, however these requirements may be added to by future systems which would require evolving Cashua's design to meet them.

7.2 Contribution

7.2.1 First contribution

Cashua's novel integration of content-based networking for context dissemination and ontology modelling of context to deal with heterogeneity provides support for two of the major requirements of context sytems—dynamism and heterogeneity.

These two technologies are a good fit for use in a context service. Ontologies are used as the basis for interoperation, and describe the structure of available context information. These ontologies are also passed to the application's CSN which publishes their information to the network. Ontologies have been used in other information management applications to describe information models, however this work expands on their utility by using ontology definitions of available context information to route queries to where they can be answered.

Content-based networking provides a service for the routing of context information, exploiting existing research in the field of publish-subscribe systems in order to distribute information to a large number of clients. By using existing content-based networking messages, the Cashua system allows CSNs to communicate with each other over the network to resolve their clients' context queries.

This novel integration also allows Cashua to meet the additional design requirements for a context system, providing a feasible approach to distributing context in a ubiquitous computing environment.

7.2.2 Second contribution

Cashua's use of content-based networking to distribute ontology mappings allows for the evolution of the system without requiring the services of an integrator. This mechanism reduces the effort required in order to allow heterogeneous applications to use the network, as they can retrieve mappings as necessary in order to allow translation between the models of different applications.

Because this system is automatic, applications may be also move around the network and register with new CSNs without providing any mapping information. Instead, this mapping information is transmitted to their new location as needed, without the intervention of a human. This mechanism has particular importance for mobile devices, which will frequently change location. Allowing applications on these devices to communicate with new CSNs without also providing mapping information (or requiring an integrator to provide it) increases the usability of the system.

7.3 Future Work

The first step in extending the usefulness of Cashua is to take more applications in the ubiquitous computing space and integrate them with Cashua in order to expand experience gained with the practicality of the system and its performance.

Using larger, more complex applications with richer context queries will allow Cashua to be improved, in two ways. Firstly, more complex applications will have more complex context models that must be integrated with those of their data sources. These larger integrations will exercise the role of the Integrator more fully, allowing insight into any difficulties they might experience. For example with large models it may not be possible to completely map all concepts between the models in question, so not all queries sent by the application would be answerable. A more in-depth study would focus on how applications should deal with this scenario.

A related element of future work could examine the 'fuzzy' nature of context information, ie. the fact that context information is often incomplete and imperfect. Some information sources may contain outdated or inaccurate information, and may not return this information in a timely manner. Future work could examine best practices for dealing with these scenarios. One possible approach would be to allow applications to decide whether they want to be returned all the (possibly contradictory) results from data sources in the environment or whether they wish the CSN to make the decision on what result to return, which could possibly be annotated with a confidence metric to indicate how certain the CSN is that the

result is accurate.

To tailor the system more closely to the need of applications, Cashua could be extended to support model mappings which are not part of the present system. As well as the current equivalent and inheritance mappings, the CSN should support transformations such as mathematical transformations which can be used to provide useful translations. Some transformations such as conversion of location information are particularly relevant to ubiquitous computing environments, which would allow Cashua-enabled applications to determine if a particular user or device was in a particular location, for example.

Cashua's implementation may also be rewritten to be more processor and memory efficient, providing greater performance and allowing CSNs to run on more resource constrained devices. Other content-based networking software may also be investigated, in order to assess changes in performance that this would bring.

Finally, Cashua's method of analyzing and translating XPath queries should be re-examined to ensure its correctness and generality. The current implementation uses the Abstract Syntax Tree (AST) generated by the XPath parser, modifying this tree in order to understand and translate queries. While this approach worked correctly for the class of queries used in the experiments described in this thesis, a more formal approach to query translation would likely be required to deal with more complex queries. A more full-featured XML query language such as XQuery should also be investigated, and support added to Cashua.

7.4 Final remarks

Cashua's integration of ontologies and content-based networking provides support for some of the challenges posed by delivering useful, timely context information to applications in ubiquitous computing environments. While these environments have not yet come to fruition, current trends in computing where users deal with an increasing number of devices in their everyday lives indicate that these environments are fast approaching. These devices will be mobile and heterogeneous, and will

require systems to be put in place to allow them to communicate with each other, enhancing their operation and providing useful context information to users.

Bibliography

[1] Apache xml-rpc, http://ws.apache.org/xmlrpc/.

[2] David Arnold, Bill Segall, Julian Boot, Andy Bond, Melfyn Lloyd, and Simon Kaplan. Discourse with disposable computers: How and why you will talk to your tomatoes. In *In Proceedings of the Usenix Workshop on Embedded Systems*, pages 9–22, March 1999.

[3] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, New York, NY, USA, 2003.

[4] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2004.

[5] Michel Biezunski and Steven R. Newcomb. Xml topic maps: Finding aids for the web. *IEEE MultiMedia*, 8(2):104–108, 2001.

[6] Robert Boyer and William Griswold. Fulcrum - an open-implementation approach to internet-scale context-aware publish/subscribe. *International Conference on System Sciences (HICSS)*, 09:275a, 2005.

[7] Dan Brickley and Libby Miller. Foaf vocabulary specification, http://xmlns.com/foaf/0.1/, January 2006.

[8] Peter Brown. The stick-e document: A framework for creating context-aware applications. In *Proceedings of Electronic Publishing 1996 (EP'96)*, pages 259–272, January 1996.

[9] Barry Brumitt, Brian Meyers, John Krumm, Amanda Kern, and Steven A. Shafer. Easyliving: Technologies for intelligent environments. In *Handheld and Ubiquitous Computing*, pages 12–29, September 2000.

[10] Stuart Card, George Robertson, and Jock Mackinlay. The information visualizer, an information workspace. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 181–186, New York, NY, USA, 1991. ACM Press.

[11] Antonio Carzaniga, David Rosenblum, and Alexander Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[12] Antonio Carzaniga, Matthew Rutherford, and Alexander Wolf. A routing scheme for content-based networking. In *Proceedings of IEEE INFOCOM 2004*, Hong Kong, China, March 2004.

[13] Antonio Carzaniga and Alexander L. Wolf. Forwarding in a content-based network. In *Proceedings of ACM SIGCOMM 2003*, pages 163–174, Karlsruhe, Germany, August 2003.

[14] R. Chand and P.A. Felber. A scalable protocol for content-based routing in overlay networks. Technical report, Institut EURECOM, Technical Report RR-03-074, February 2003.

[15] Harry Chen. *An Intelligent Broker Architecture for Pervasive Context-Aware Systems.* PhD thesis, University of Maryland, Baltimore County, USA, December 2004.

[16] Harry Chen, Tim Finin, and Anupam Joshi. An ontology for context-aware pervasive computing environments. *Knowledge and Engineering Review, Special Issue on Ontologies for Distributed Systems*, 2003.

[17] Harry Chen, Tim Finin, and Anupam Joshi. Using owl in a pervasive computing broker. In *Proceedings of Workshop on Ontologies in Agent Systems, AAMAS-2003*, July 2003.

[18] Harry Chen, Tim Finin, and Anupam Joshi. An ontology for context aware pervasive computing environments. *Knowledge Engineering Review - Special Issue on Ontologies for Distributed Systems*, 2004.

[19] Harry Chen, Tim Finin, and Anupam Joshi. *The SOUPA Ontology for Pervasive Computing*, pages 233–258. Whitestein Series in Software Agent Technologies. Springer, July 2005.

[20] Harry Chen, Filip Perich, Dipanjan Chakraborty, Tim Finin, and Anupam Joshi. Intelligent agents meet semantic web in a smart meeting room. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 854–861, Washington, DC, USA, 2004. IEEE Computer Society.

[21] Harry Chen, Filip Perich, Tim Finin, and Anupam Joshi. SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. In *International Conference on Mobile and Ubiquitous Systems: Networking and Services*, Boston, MA, August 2004.

[22] Namyoun Choi, Il-Yeol Song, and Hyoil Han. A survey on ontology mapping. *SIGMOD Rec.*, 35(3):34–41, 2006.

[23] Owen Conlan, Ian O'Keeffe, Aoife Brady, and Vincent Wade. Enabling adaptive semantic interoperability for pervasive computing. In *Seventh IEEE International Conference onAdvanced Learning Technologies, 2007. ICALT 2007*, July 2007.

[24] Stephen E. Deering and David R. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Trans. Comput. Syst.*, 8(2):85–110, 1990.

[25] Anind Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, College of Computing, Georgia Institute of Technology, December 2000.

[26] Anind K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7, February 2001.

[27] Anind K. Dey, Gregory D. Abowd, Mike Pinkerton, and Andrew Wood. Cyberdesk: A framework for providing self-integrating context-aware services. In *ACM Symposium on User Interface Software and Technology*, pages 75–76, 1997.

[28] Christophe Diot, Brian Neil Levine, Bryan Lyles, Hassan Kassem, and Doug Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network*, 14(1):78–88, 2000.

[29] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Learning to map between ontologies on the semantic web. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 662–673, New York, NY, USA, 2002. ACM Press.

[30] Dejing Dou. *Ontology Translation by Ontology Merging and Automated Reasoning*. PhD thesis, Yale University, New Haven, CT, USA, 2004.

[31] Ludger Fiege, Mariano Cilia, Gero Muhl, and Alejandro Buchmann. Publish-subscribe grows up: support for management, visibility control, and heterogeneity. *IEEE Internet Computing*, 10(1):48–55, January 2006.

[32] Florian Fuchs, Iris Hochstatter, Michael Krause, and Michael Berger. A metamodel approach to context information. In *PERCOMW '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 8–14. IEEE Computer Society, 2005.

[33] Tao Gu, Hung Keng Pung, and Daqing Zhang. A peer-to-peer overlay for context information search. In *Proceedings of 14th International Conference on*

Computer Communications and Networks 2005. ICCCN 2005, pages 395–400, October 2005.

[34] Tao Gu, Xiao Hang Wang, Hung Keng Pung, and Da Qing Zhang. An ontology-based context model in intelligent environments. In *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference, San Diego, California, USA*, January 2004.

[35] Volker Haarslev and Ralf Möller. RACER system description. *Lecture Notes in Computer Science*, 2083:701–??, 2001.

[36] Jason Hong and James Landay. An infrastructure approach to context-aware computing. *Human-Computer Interaction*, 16(2, 3 & 4):287–303, 2001.

[37] Jabber.org technology overview, http://www.jabber.org/jabber-for-geeks/technology-overvie

[38] Jxta technology overview, http://www.sun.com/software/jxta/.

[39] Cory Kidd, Robert Orr, Gregory Abowd, Christopher Atkeson, Irfan Essa, Blair MacIntyre, Elizabeth Mynatt, Thad Starner, and Wendy Newstetter. The aware home: A living laboratory for ubiquitous computing research. In *Proceedings of the Second International Workshop on Cooperative Buildings - CoBuild'99*, October 1999.

[40] Tim Kindberg, John Barton, Jeff Morgan, Gene Becker, Debbie Caswell, Philippe Debaty, Gita Gopal, Marcos Frid, Venky Krishnan, Howard Morris, John Schettino, Bill Serra, and Mirjana Spasojevic. People, places, things: Web presence for the real world. *Mobile Networks and Applications*, 7(5):365–376, October 2002.

[41] Sanjeev Kumar, Philip R. Cohen, , and Hector J. Levesque. The adaptive agent architecture: Achieving fault-tolerance using persistent broker teams. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS 2000)*, pages 159–166, Boston, MA, 2000.

[42] Douglas Lenat and R. V. Guha. *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Addison-Wesley, February 1990.

[43] Alexander Maedche, Boris Motik, Nuno Silva, and Raphael Volz. Mafra – a mapping framework for distributed ontologies. In *Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web: 13th International Conference, EKAW 2002, Siguenza, Spain*, page 235, 2002.

[44] Alexander Maedche, Boris Motik, Ljiljana Stojanovic, Rudi Studer, and Raphael Volz. Ontologies for enterprise knowledge management. *IEEE Intelligent Systems*, 18(2):26–33, 2003.

[45] P. Mitra and G. Wiederhold. Resolving terminological heterogeneity in ontologies. In *Proceedings of ECAI-02 Workshop, CEUR-WS 64, Ontologies and Semantic Interoperability*, 2002.

[46] Declan O'Sullivan, David Lewis, and Vincent Wade. Enabling adaptive semantic interoperability for pervasive computing. In *Proceedings of the 2nd International Workshop on Managing Ubiquitous Communications and Services*, December 2004.

[47] Aris M. Ouksel, Oana Jurca, Ivana Podnar, and Karl Aberer. Efficient Probabilistic Subsumption Checking for Content-Based Publish/Subscribe Systems. In *7th International Middleware Conference (Middleware 2006)*, 2006.

[48] W3c web ontology language overview, http://www.w3.org/tr/owl-features/.

[49] Filip Perich. Mogatu bdi ontology, http://mogatu.umbc.edu/bdi/, January 2004.

[50] Filip Perich, Sasikanth Avancha, Anupam Joshi, and Yelena Yesha. Query routing and processing in mobile ad-hoc environments. Technical report, University of Maryland, Baltimore County, October 2001.

[51] Filip Perich, Anupam Joshi, Tim Finin, and Yelena Yesha. On Data Management in Pervasive Computing Environments. *IEEE Transactions on Knowledge and Data Engineering*, May 2004.

[52] Michael Pinkerton. Ubiquitous computing: Extending access to mobile data. Master's thesis, Georgia Institute of Technology, June 1997.

[53] David Poole. Compiling a default reasoning system into prolog. *New Generation Computing*, 9(1):3–38, 1991.

[54] Ruaidhri Power. Topic maps for context management. In *Proceedings of the 1st International Symposium on Information and Communication Technologies*, page 235, 2002.

[55] Davy Preuveneers, Jan Van den Bergh, Dennis Wagelaar, Andy Georges, Peter Rigole, Tim Clerckx, Yolande Berbers, Karin Coninx, Viviane Jonckers, and Koen De Bosschere. Towards an extensible context ontology for ambient intelligence. *Lecture Notes in Computer Science: Second European Symposium on Ambient Intelligence.*, 3295:148–159, 2004.

[56] Protege website, http://protege.stanford.edu/.

[57] Rdf query survey, http://www.w3.org/2001/11/13-rdf-query-rules/.

[58] Bill Schilit and Marvin Theimer. Disseminating active map information to mobile hosts. *IEEE Network*, 8(5):22–32, 1994.

[59] Albrecht Schmidt, Kofi Asante Aidoo, Antti Takaluoma, Urpo Tuomela, Kristof Van Laerhoven, and Walter Van de Velde. Advanced interaction in context. In *HUC '99: Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, pages 89–101, London, UK, 1999. Springer-Verlag.

[60] Bill Segall, David Arnold, Julian Boot, Michael Henderson, and Ted Phelps. Content-based routing with elvin4. In *Proceedings of AUUG2K, Canberra*, 2000.

[61] W3c semantic web activity, http://www.w3.org/2001/sw/.

[62] Ben Shneiderman. Response time and display rate in human performance with computers. *ACM Computing Surveys*, 16(3):265–285, 1984.

[63] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. Technical report, UMIACS, 2005.

[64] Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *Proceedings of Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp 2004*, September 2004.

[65] Thomas Strang, Claudia Linnhoff-Popien, and Korbinian Frank. CoOL: A Context Ontology Language to enable Contextual Interoperability. In Jean-Bernard Stefani, Isabelle Dameure, and Daniel Hagimont, editors, *LNCS 2893: Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003)*, volume 2893 of *Lecture Notes in Computer Science (LNCS)*, pages 236–247, Paris/France, November 2003. Springer Verlag.

[66] Anthony Tarlano and Wolfgang Kellerer. Context spaces architectural framework. In *SAINT-W '04: Proceedings of the 2004 Symposium on Applications and the Internet-Workshops (SAINT 2004 Workshops)*, page 702, Washington, DC, USA, 2004. IEEE Computer Society.

[67] The Gryphon Team. Achieving scalability and throughput in a publish/subscribe system. Technical report, IBM Research Report RC23103, Feb 2004.

[68] Xiao Hang Wang, Tao Gu, Da Qing Zhang, and Hung Keng Pung. Ontology-based context modeling and reasoning using owl. In *Proceedings of Modeling and Reasoning Workshop at PerCom 2004*, 2004.

[69] Robert H. Warren and Christopher O. J. Baker. Ontologies: Where are we at?, poster at knowledge-based bioinformatics

workshop, montreal, quebec, canada, september 2005. http://junobeach.cs.uwaterloo.ca/~warren/publications/warren:bio:2005/bio_wo%rks

[70] Mark Weiser. The computer for the twenty-first century. *Scientific American*, 265(3):94–104, September 1991.

[71] Mark Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):75–84, 1993.

[72] Wine ontology, http://www.w3.org/tr/owl-guide/wine.rdf.

[73] Xen hypervisor, http://xen.org/.

[74] Xml:db initiative for xml databases, http://xmldb-org.sourceforge.net/.

[75] Xml-rpc website, http://www.xmlrpc.com/.

[76] Xpath 2.0 specification, http://www.w3.org/tr/xpath20/.

The following appendixes list additional information. Appendix A lists peer-reviewed publications relating to this work. Appendix B lists the questionnaire given to application developers to investigate their experience with Cashua, while Appendix C and D list the completed questionnaires from the Adaptive Engine and Interlocutor developers respectively. Appendix E describes technologies used in Cashua, while Appendix F lists additional results from evaluation.

Appendix A

Published Work

D. Lewis, D. O'Sullivan, K. Feeney, J. Keeney, R. Power, Ontology-based Engineering for Self-Managing Communications, 1st IEEE International Workshop on Modelling Autonomic Communications Environments (MACE 2006), Dublin, Ireland, 25-26 October 2006, edited by W. Donnelly, R. Popescu-Zeletin, J. Strassner, B. Jennings, S. van der Meer , multicon verlag, 2006, pp81 - 100.

D. Lewis, D. O'Sullivan, R. Power, J. Keeney, Semantic Interoperability for an Autonomic Knowledge Delivery Service, 2nd IFIP WG6.6 International Workshop on Autonomic Communication - Autonomic Communication Principles (WAC 2005), Athens, Greece, 3-5 October 2005, LNCS 3854, Springer, 2006, pp129 - 140.

R. Power, D. O'Sullivan, Cashua: A Context Awareness Service for Heterogeneous Ubicomp Applications, 3rd International Workshop on Managing Ubiquitous Computing and Services (MUCS), Co-located with Pervasive 2006, Cork, Ireland, 2006

R. Power, D. O'Sullivan, D. Lewis, O. Conlan, V. Wade, A Context Information Service using Ontology-Based Queries, First International Workshop on Advanced Context Modelling, Reasoning And Management, UbiComp 2004, Nottingham, England, September 7 - 10, 2004.

Power, R., O'Sullivan, D., Conlan, O., Lewis, D., Wade, V., Utilizing context in adaptive information services for pervasive computing environments, Workshop

on Applying Adaptive Hypermedia Techniques to Service Oriented Environments: Pervasive Adaptive Web Services and Context Aware Computing, Eindhoven, The Netherlands, August 2004, 2004.

Ruaidhri Power. Topic maps for context management. In Proceedings of the 1st International Symposium on Information and Communication Technologies, page 235. Dublin, Ireland. September 2003.

Appendix B

Developer Questionnaire

B.1 Application Overview

Describe the application in general terms. Include details about the application such as:

Function what task is the application designed to perform?

End users does your application primarily or exclusively cater for any particular class of end user? If so, please provide information about this class of user such as their age, gender and particularly their technical knowledge.

Queries what information does the application query? Who provides this information, and where is it retrieved from? Who designed the format of this information? How often does this information change?

B.2 Application Data Model

B.2.1 Model complexity

The following questions aim to characterize the application's data model. Here for example, if your application was a library inventory management system, the

homogeneous entities it manages might be books, with each instance of the data model representing a single book.

What is the average file size of an instance of the model?

Given that the model is expressed in XML, what is the average number of unique XML tags in the model?

How many homogeneous entities are represented in instances of the data model? These 'homogeneous entities' are as defined above. If this number varies according to the number of users of your application, please indicate how these numbers are related.

Any further comments...

B.2.2 Application Queries

How often does your application query the model?

Describe the queries your application makes on the model. An exhaustive list of query types is preferable, with a short description of what each query does. Eg: `/book[author='Shakespeare']/title` will display the titles of books by Shakespeare.

Any further comments...

B.3 Developer Experience

In order to ascertain your experience in relevant technologies, please elaborate on your experience in the following areas:

For each of the following, please provide the following information:

- Your comfort with the subject matter in this area.
- Number of years you have worked with this technology.

- An estimation of the number and relative size of projects you have contributed to using this technology.

Information modelling such as UML, Relational databases, Object-oriented programming.

XML-based technologies—XML markup, XSL, XSLT, etc.

XML Query languages—XPath and XQuery.

Ontology modelling concepts

Ontology languages—DAML+OIL, OWL, etc.

B.4 Ontology Concepts Introduction

If your knowledge of ontologies and ontology languages is limited, please study the Ontology 101 primer document, available on the web at:

```
http://protege.stanford.edu/publications/ontology_development/
                ontology101-noy-mcguinness.html
```

Following this, you should be familiar with the terms 'class' and 'property' (also called 'slots' in the guide) as they apply to ontologies and should be comfortable with creating your own ontology with Protégé.

B.5 Context Modeling Process

B.5.1 Instructions

Studying the existing data model of the application, identify the classes within it that you wish to be able to query through the CSN. As a running example, we will assume that your application retrieves context about people, each person having a name and address. Your data model might then look something like this:

```
<person>
```

```
 <name>Joe Bloggs</name>
 <address>1 Infinite Loop, Cupertino, CA</address>
</person>
```

This would be represented by an ontology with person as a class, each person having properties of name and address.

- When creating a class that represents a particular XML tag within the model, such as $< person >$, give the class the same name.
- Otherwise, put a relative XPath expression that is used to select these elements as a comment on the new class. The class can then be called whatever you wish.
- Using Protégé, create an OWL ontology of these concepts. At the Protégé loading screen, choose "Create New Project ...".
- Choose "OWL/RDF Files" from the menu. And click "Next".
- Enter the full URI where this ontology will reside.

  And click "Finish".
- The main Protégé window will appear, with the metadata tab visible
- Click on the "OWLClasses" tab, so you can construct the class hierarchy.
- Select "owl:Thing" from the subclass explorer pane, and click on the "Create subclass" button to create your first subclass.
- Type the name of the first class into the text box in the "Class editor" pane.
- Repeat this process for each subclass you wish to add to the model.
- To create the first datatype property for your ontology, switch to the "Properties" tab.
- Click on the "Datatype" tab in the Property browser, and then click on the "Create datatype property" button.

- Name your property, in the same manner as you named your classes.
- Specialise the domain of this property to a particular class by clicking the "Specialise domain" button.
- And choose the class this property applies to in the window that appears.
- Upload the created ontology to the web so that it's accessible at that URL.

B.6 Deployment

- Taking the OWL ontology developed using Protégé, load it into the CSN using the CSN Manager application. To do this, start the Manager application and on the 'Ontologies' tab
- Redirect XPath queries that were previously sent to a database to go to the CSN's web service instead. The query function takes two arguments: the XPath string (exactly as it was previously), and the name of the ontology you loaded. The web service methods are as follows:

```
String blockingQuery(String query, String ontologyName);
String nonBlockingQuery(String query, String ontologyName);
```

  The blockingQuery method doesn't return a result until one has been retrieved, and nonBlockingQuery takes the query and will execute a callback whenever results are received. This non-blocking query can be used to retrieve results over a period of time.

What software development time was required to redirect the application's queries to use the CSN?

Does your application use the non-blocking query, that returns results over time?

If so, explain how it uses this query. If not, do you feel your application could be enhanced by adding this functionality to it?

Did you notice any functional change in how your application works, or have to make any changes to the source code apart from redirecting queries?

Appendix C

Adaptive Engine Questionnaire Results

C.1 Application Overview

Describe the application in general terms. Include details about the application such as:

Function what task is the application designed to perform?

The system adaptively composes personalised courseware, based on a predefined pedagogical strategy. The system presents the composition to the user through a web interface by dynamically generating the link structure and pulling in the appropriate content resources. The adaptive behaviour is manifested through the presentation of different resources to the user based on their prior knowledge of the subject domain (Database theory and SQL) as well as their preferred learning style (based on Honey and Mumford).

End users does your application primarily or exclusively cater for any particular class of end user? If so, please provide information about this class of user such as their age, gender and particularly their technical knowledge.

The particular system implementation in question is used by Sophister students in the Computer Science/Engineering domain (ages 20-24 approx.)

Queries what information does the application query? Who provides this information, and where is it retrieved from? Who designed the format of this information? How often does this information change?

The system queries a remote XML database for attributes of the current user (each user has a personal profile stored in the database). The format of the user model is of a proprietary nature and was designed by the original developers or the adaptive courseware (SQL course). It consists of user information broken up into sections which correspond to the types of adaptation that can be applied in an adaptive system such as the one in question (in this case prior knowledge and learning style). The user profile is populated by the user indirectly through instruments (online form/questionnaire) which they must complete prior to first using the system. These instruments are available to the user at all times so that they can later 'rebuild' their profile.

The system queries the database for the user profile whenever it is known that the profile has changed (i.e. Whenever the user has 'rebuilt' their profile). This process occurs variable number of times depending on the individual user but can be considered to be relatively static over short periods of time.

C.2 Application Data Model

C.2.1 Model complexity

The following questions aim to characterize the application's data model. Here for example, if your application was a library inventory management system, the homogeneous entities it manages might be books, with each instance of the data model representing a single book.

What is the average file size of an instance of the model?

Models which simply contain the prior knowledge information are approx. 4kb in size while models containing both prior knowledge and learning style information are approx. 8kb in size.

Given that the model is expressed in XML, what is the average number of unique XML tags in the model?

13 and approx 28 respectively

How many homogeneous entities are represented in instances of the data model? These 'homogeneous entities' are as defined above. If this number varies according to the number of users of your application, please indicate how these numbers are related.

As each XML resource in the database refers to an individual user, only one homogeneous entity is represented per model

Any further comments...

C.2.2 Application Queries

How often does your application query the model?

In the case of the prior knowledge query, for 34 users over a 8 month period there were 333 rebuilds, that averages 9.7 rebuilds per user. For the learning styles set of queries, a set of 31 users averaged 4.7 rebuilds over the same period of time.

Describe the queries your application makes on the model. An exhaustive list of query types is preferable, with a short description of what each query does. Eg: `/book[author='Shakespeare']/title` will display the titles of books by Shakespeare.

```
/learner[general/identifier='username']/educational/adaptivity/adaptivitytype/set/candidate/langst
```

Retrieves a list of all of the concepts that a course should cover

```
/learner[general/identifier='username']/accessibility/preference/ext_preference/activist
/learner[general/identifier='username']/accessibility/preference/ext_preference/reflector
/learner[general/identifier='username']/accessibility/preference/ext_preference/theorist
/learner[general/identifier='username']/accessibility/preference/ext_preference/pragmatist
```

Retrieves values for each learning style from the learner model

Any further comments...

C.3 Developer Experience

In order to ascertain your experience in relevant technologies, please elaborate on your experience in the following areas:

For each of the following, please provide the following information:

- Your comfort with the subject matter in this area.
- Number of years you have worked with this technology.
- An estimation of the number and relative size of projects you have contributed to using this technology.

Information modelling such as UML, Relational databases, Object-oriented programming.

I have used UML as part of large integrated projects in order to capture/convey system design with other developers. WRT OO programming, I have a lot of experience with several OO languages, e.g. C++, Java, C# (7 years experience)

XML-based technologies—XML markup, XSL, XSLT, etc.

I have a strong knowledge of these topics as they are fundamental to the software that I have developed. The systems I have developed read/write XML files and store their DOM structure in memory using available libraries as well as transforming raw XML models using XSLT in order to generate hypertexts. (3 years experience)

XML Query languages—XPath and XQuery.

As in the previous point, the systems I have developed are heavily based on XPath however my use of XQuery to this point has been relatively limited although I am

aware of what it can do in theory. (3 years experience)

Ontology modelling concepts

I have a good conceptual understanding of Ontologies and have used them as part of some simple systems, however I have not developed my own ontologies. (1 year)

Ontology languages—DAML+OIL, OWL, etc.

I have a limited knowledge of the syntax, etc. of ontology modeling languages. I generally use tools to work with ontologies.

C.4 Context Modeling Process

How many classes did you identify in the context modeling process?

8 classes

Did you have any difficulty establishing the hierarchy of classes? If so, give specific examples of the difficulties and the decisions you finally made.

No

How long did the entire process take?

10-15 minutes

Did you need to use any tools apart from Protégé in the creation of the ontology?

No

Any further comments...

C.5 Deployment

- Taking the OWL ontology developed using Protégé, load it into the CSN using the CSN Manager application. To do this, start the Manager application and on the 'Ontologies' tab
- Redirect XPath queries that were previously sent to a database to go to the

CSN's web service instead. The query function takes two arguments: the XPath string (exactly as it was previously), and the name of the ontology you loaded. The web service methods are as follows:

```
String blockingQuery(String query, String ontologyName);
String nonBlockingQuery(String query, String ontologyName);
```

The blockingQuery method doesn't return a result until one has been retrieved, and nonBlockingQuery takes the query and will execute a callback whenever results are received. This non-blocking query can be used to retrieve results over a period of time.

What software development time was required to redirect the application's queries to use the CSN?

The amount of time it takes to edit a script so that the URL to the database is modified (5 minutes)

Does your application use the non-blocking query, that returns results over time?

No, this application carries out inference on the data retrieved from the CSN in order to present results to the user. Therefore it cannot carry out any other work while waiting for the result.

If so, explain how it uses this query. If not, do you feel your application could be enhanced by adding this functionality to it?

N/A

Did you notice any functional change in how your application works, or have to make any changes to the source code apart from redirecting queries?

There were not functional changes to the application/system.

The only changes necessary were to redirect queries to the CSN.

Appendix D

Interlocutor Questionnaire Results

D.1 Application Overview

Describe the application in general terms. Include details about the application such as: Function—what task is the application designed to perform?

Pervasive ad-hoc environments have the characteristics of many services interoperating with one another. This is beginning to force a rethink with respect to the architectures of traditional collaboration applications (e.g. IM, email) that will need to continue to operate in such environments. This paper describes a novel mechanism called Interlocutor that has been implemented to cater for the seamless migration of certain functionality of applications based on centralised architectures to distributed services. In addition Interlocutor allows these previously centralised applications to take advantage of newly available distributed services. The implementation of an enhanced Instant Messaging application is used to illustrate the use of the mechanism. The Interlocutor mechanism itself is decentralised.

For this research an application was chosen as a use case to demonstrate and evaluate Interlocutor. Instant Messaging (IM) application was chosen because it was a good exemplar of a popular application based on centralised architecture. The interceptor mechanism was evaluated and analysed by integrating different researchers work (services) into the IM application, which we will term the enhanced IM application in

this paper. The following services were accessed through the intercepting mechanism:

- *The Pudecas Simulator provided a location information service. The Pudecas Simulator , is a 3D virtual environment which was configured to emit user location information as users navigate the 3D world.*
- *The Community Based Policy Management System (CBPMS), was used to formulate and manage communities of IM users.*
- *The Trust Based Access Control (TBAC) service was used to emit access control decisions for location information on an individual user basis.*
- *Cashua service was added and offers a means for dynamically routing user mood queries in the enhanced IM application.*

End users—does your application primarily or exclusively cater for any particular class of end user? If so, please provide information about this class of user such as their age, gender and particularly their technical knowledge.

No

Queries—what information does the application query? Who provides this information, and where is it retrieved from? Who designed the format of this information? How often does this information change?

Mood Information—The users, stored in an database, I did, as often as the users change it.

D.2 Application Data Model

D.2.1 Model complexity

The following questions aim to characterize the application's data model. Here for example, if your application was a library inventory management system, the

homogeneous entities it manages might be books, with each instance of the data model representing a single book.

What is the average file size of an instance of the model?

10 KB

Given that the model is expressed in XML, what is the average number of unique XML tags in the model?

One

How many homogeneous entities are represented in instances of the data model? These 'homogeneous entities' are as defined above. If this number varies according to the number of users of your application, please indicate how these numbers are related.

Any further comments...

D.2.2 Application Queries

How often does your application query the model?

As often as the users does.

Describe the queries your application makes on the model. An exhaustive list of query types is preferable, with a short description of what each query does. Eg: `/book[author='Shakespeare']/title`—display the titles of books by Shakespeare.

Query used is: `/mood[user='username']`

Any further comments...

D.3 Developer Experience

In order to ascertain your experience in relevant technologies, please elaborate on your experience in the following areas:

For each of the following, please provide the following information:

- Your comfort with the subject matter in this area.
- Number of years you have worked with this technology.
- An estimation of the number and relative size of projects you have contributed to using this technology.
- Information modelling such as UML, Relational databases, Object-oriented programming.

  I have some experience modeling my software with UML, I have a lot experience in both Relational Databases, mysql and over two years experience working with IBM DB2, Java I have 6 years experience.
- XML-based technologies—XML markup, XSL, XSLT, etc.

  I have used XMPP—XML streaming protocol and I have excellent experience using XML very good comfort level, 5 years. 10 projects spread between Industry and Academia. No experience in XSL and in XSLT.
- XML Query languages - XPath and XQuery.

  Good amount of Experience - 6 months, good comfort level 2 projects in Academia
- Ontology modelling concepts

  No Direct experience, good comfort level, I have read and attend several presentations on this topic.
- Ontology languages—DAML+OIL, OWL, etc.

  No direct experience.

D.4 Context Modeling Process

How many classes and properties did you identify in the context modeling process?

Three classes

Did you have any difficulty establishing the hierarchy of classes? If so, give specific examples of the difficulties and the decisions you finally made.

N/A

How long did the entire process take?

5 minutes

Did you need to use any tools apart from Protégé in the creation of the ontology?

No

Any further comments...

Very easy!!

D.5 Deployment

What software development time was required to redirect the application's queries to use the CSN?

One day in linking of the two services. 4hrs the programming in place. 3 hrs test establishing connections between entities components elements

Please note: Three days. There was a small degree of learning to handle the soap implementation

Does your application use the non-blocking query, that returns results over time?

Yes

If so, explain how it uses this query. If not, do you feel your application could be enhanced by adding this functionality to it?

Provided the application with asynchronous notification changes of mood data. The enhanced IM application implemented gives you access to location information and mood information. A user can choose a member of their IM roster list and request either their location or mood information. The location information is provided by the Pudecas Simulator and the mood information is supplied by the Cashua service.

The enhanced IM application invisibly uses Cashua for the delivery of context information , in this case the dissemination of mood information to and from enhanced IM clients. The Cashua service was added to the network as seen in figure 23 point four. The mood service is explored further section 4.2.4.4.

The MC Service allows IM Clients to assign a mood to how they are presently feeling and to request the moods of other individuals in their roster list. The MC Service has two types of message that are generated by the IM client a MOOD-UPDATE (MU) and MOOD-REQUEST (MR). A MOOD-UPDATE is a message which is sent from the client updating his/her mood whether it is sad, happy or angry. The function of a MOOD-REQUEST message is to request the mood from a member of the clients IM roster list. Figure 39 illustrates the Mood Service in operation with Intra-loc. If a client performs a MR, a message is sent for the target (IM roster buddy) client to the Message handler as seen in the figure at point one. The configuration is established from the Configuration handler and because it is a CACHE-REQUEST the message is processed by the Cache handler.

After the cache request has been cached and the Content Handler translates the message from a standard XML message to a SOAP message, it is sent to the external web service monitoring for MR as seen in the figure at point five. The Cashua service receives the request and sends the response to the Web Component SOAP Service connection in the Service handler as seen in the figure at point seven. If it is a MU message the configuration outlines that it is to be delivered to Elvin. MU messages are configured not to be cached and access control is also not required as seen in figure 40. Therefore the message is syntactically translated and delivered to the Elvin protocol as seen in the figure at point six. If there are any subscriptions to a clients mood Cashua sends this change in mood to the Service handler via the SOAP web component.

Did you notice any functional change in how your application works, or have to make any changes to the source code apart from redirecting queries?

No changes were required to the existing application apart from the addition of a query for mood information.

Appendix E

Technologies Used in Cashua

E.1 XML-RPC

XML-RPC[75] is a remote procedure call method which encodes its calls in XML, which it then sends over a HTTP protocol. XML-RPC is a very simple protocol to implement, although it only supports a limited number of data types. The standard for XML-RPC was initially implemented by Dave Winer when he was working with Microsoft, and was eventually extended to form the SOAP protocol. It is still widely used due to its simplicity, minimalism and ease of use. XML-RPC supports method calls using primitive data types such as booleans, integers, doubles, etc. It also supports arrays and strings, but not complex user-defined types although language-specific complex types can be encoded and sent across the protocol.

E.2 eXist

E.2.1 Overview

Cashua uses the eXist open source native XML database for model and mapping storage. It is released under the terms of the GNU LGPL[1].

[1] http://www.gnu.org/copyleft/lesser.html

Can be deployed as a standalone database server, as an embedded library inside an application, or in a servlet engine. The engine stores XML documents in collections which are similar to a filesystem hierarchy, and supports both XPath and XQuery to access this data. These collections are indexed in order to provide better query performance.

The server supports multiple mechanisms for providing access to data. From a simple REST style HTTP protocol, eXist also supports XML-RPC and SOAP (in servlet mode) to access all database functions. The document hierarchy is available via eXist's WebDAV support.

- Integrated XML databases for ontology and mapping repositories
- XQuery Abstract Syntax Tree (AST) extraction

E.2.2 API Overview

Once the database driver has been initialised, a reference to a collection can be retrieved using its URL:

```
Collection col = DatabaseManager.getCollection("xmldb:exist://localhost:8080/exist/xmlrpc/db");
```

This collection can be used to retrieve an XPathQueryService object for the collection.

```
XQueryService service = (XQueryService) col.getService("XQueryService", "1.0");
```

This query service can be used to perform queries using the query() method as follows:

```
ResourceSet result = service.query(queryString);
```

A Java Iterator object can be retrieved from this ResourceSet, and used to iterate over the resources in the result. Each of these resources is a single XML document or document fragment selected by the XPath expression.

The API can also be used to create new XML resources inside a collection. The content of the document can be filled in from a File object, and stored as a Resource

in the database.

```
XMLResource document = (XMLResource)col.createResource(null, "XMLResource");
File f = new File(fileName);
document.setContent(f);
col.storeResource(document);
```

E.3 Jena

E.3.1 Overview

Jena is a Java framework for building Semantic Web[61] applications, and allows processing and manipulation of RDF, RDFS and OWL ontologies. It also includes an inference engine using a variety of reasoners and SPARQL query support. Jena was developed as part of HP Labs Semantic Web Research[2], and was released under an open source license in order to encourage its adoption.

Cashua makes extensive use of Jena's ontology API, which is language-neutral, using calls which are not specific to particular ontology languages. This allows Jena applications to concentrate on the manipulation of the ontology, rather than the specifics of the ontology language used. Jena provides a consistent programming interface to the developer, independent of which ontology language is being used. The features of ontology languages are expressed using profiles, which when bound to an ontology model allow for querying of the information stored in the underlying RDF model, using the facilities supported by the ontology language.

E.3.2 Ontology API

makeConceptModel():

```
OntModel conceptModel = ModelFactory.createOntologyModel(ProfileRegistry.OWL_LANG);

conceptModel.add(ontModel);
```

[2]http://www.hpl.hp.com/semweb/

getConcepts():

```
ExtendedIterator iter = conceptModel.listClasses();
while (iter.hasNext()) {
        OntClass ontClass = (OntClass)iter.next();
        String classURI = ontClass.getURI();
        conceptList.add(classURI);
}
```

```
OntClass ontClass = conceptModel.getOntClass(ontClassURI);
String comment = ontClass.getComment(null);
```

Appendix F

Additional Evaluation Results

To demonstrate graphically the time taken in the subscription addition and removal, the following graphs show the times taken for each part of the two thousand queries. These graphs show the results for the AE (Figure F.1), shallowBushy (Figure F.2), deepThin (Figure F.3) and deepBushy (Figure F.4) queries. The top line of each of these graphs shows the total time taken for each query. The other lines show the time taken for each stage of the CSN operation, added to the time taken for each preceding stage. The three large gaps shown in all four of these graphs show the subscription, waiting and subscription removal times, which dominate all other time spent by the local CSN.

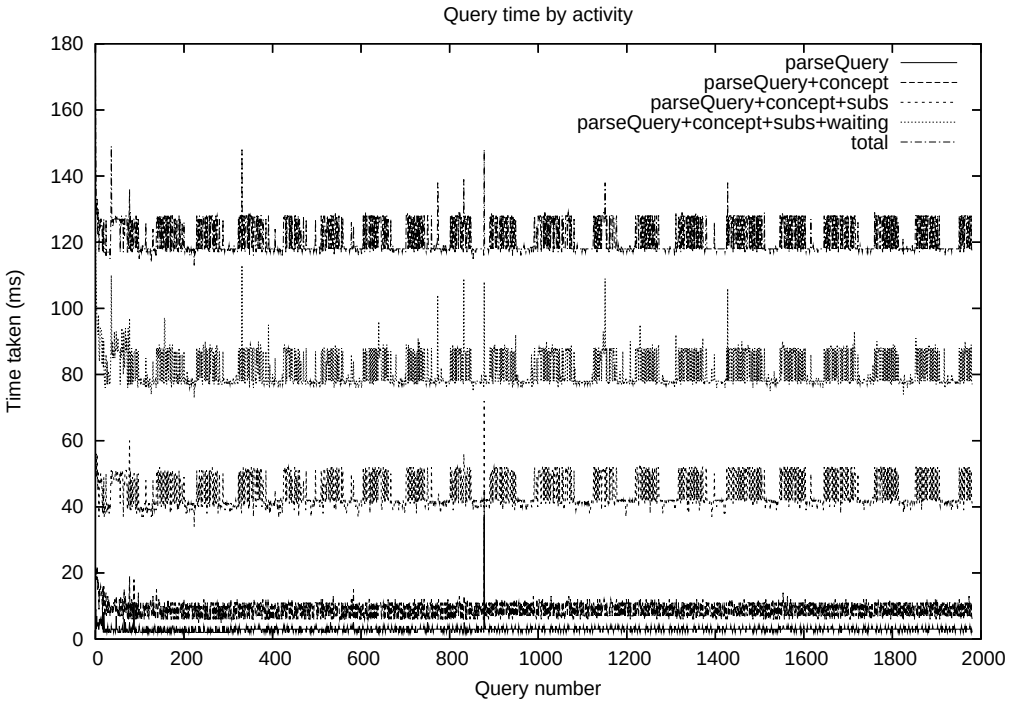

Figure F.1: Adaptive Engine queries on local CSN

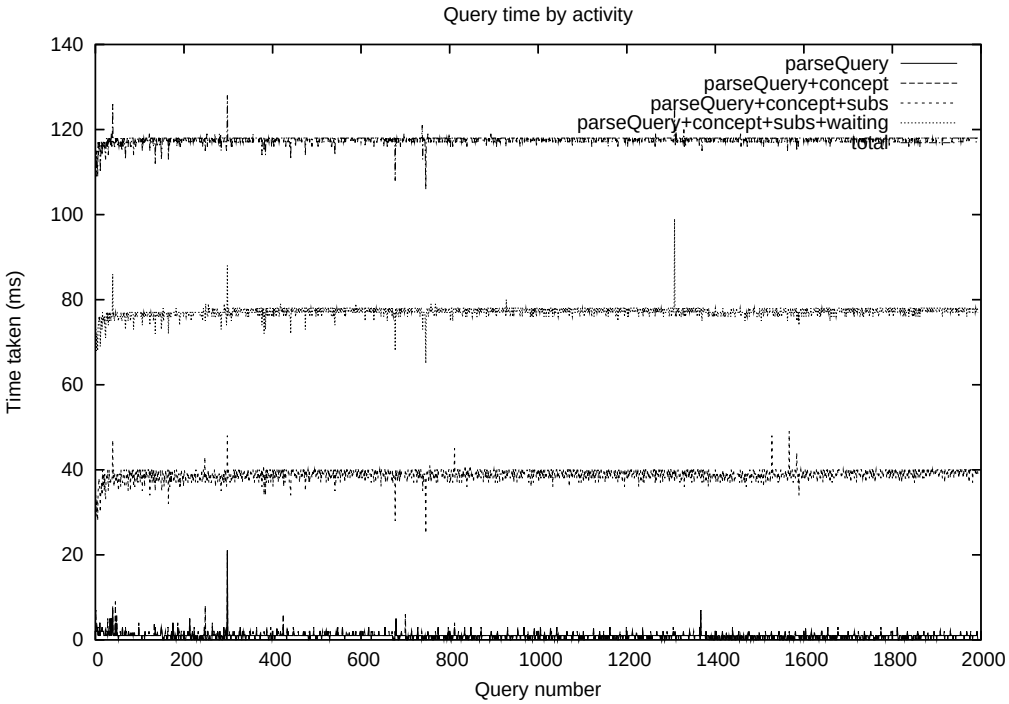

Figure F.2: Shallow, bushy ontology queries on local CSN

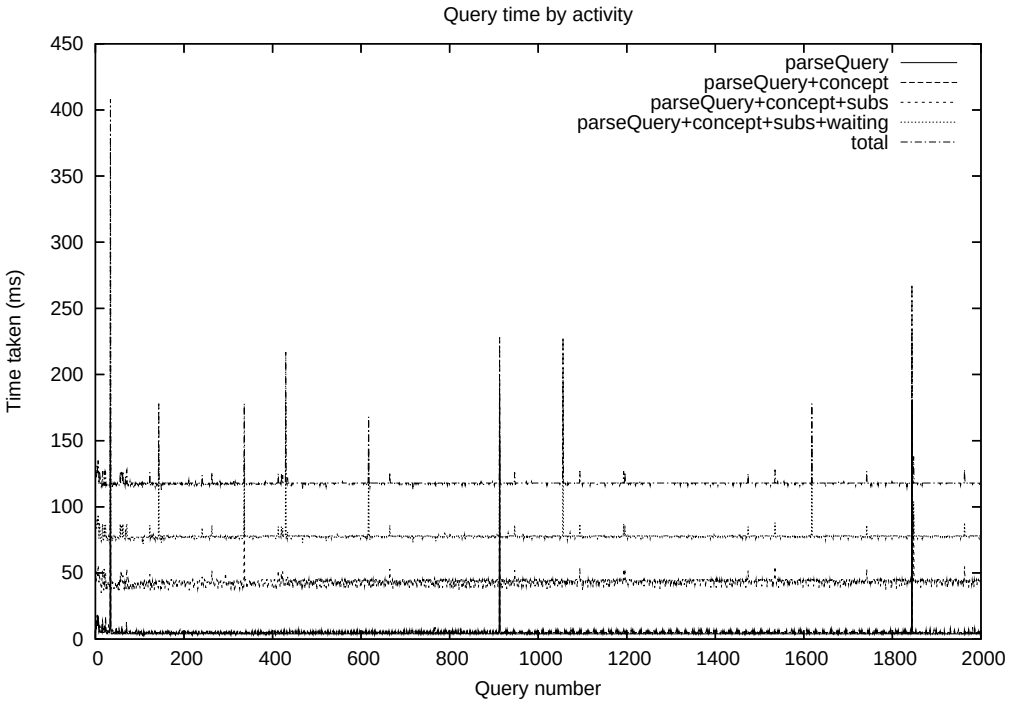

Figure F.3: Deep, thin ontology queries on local CSN

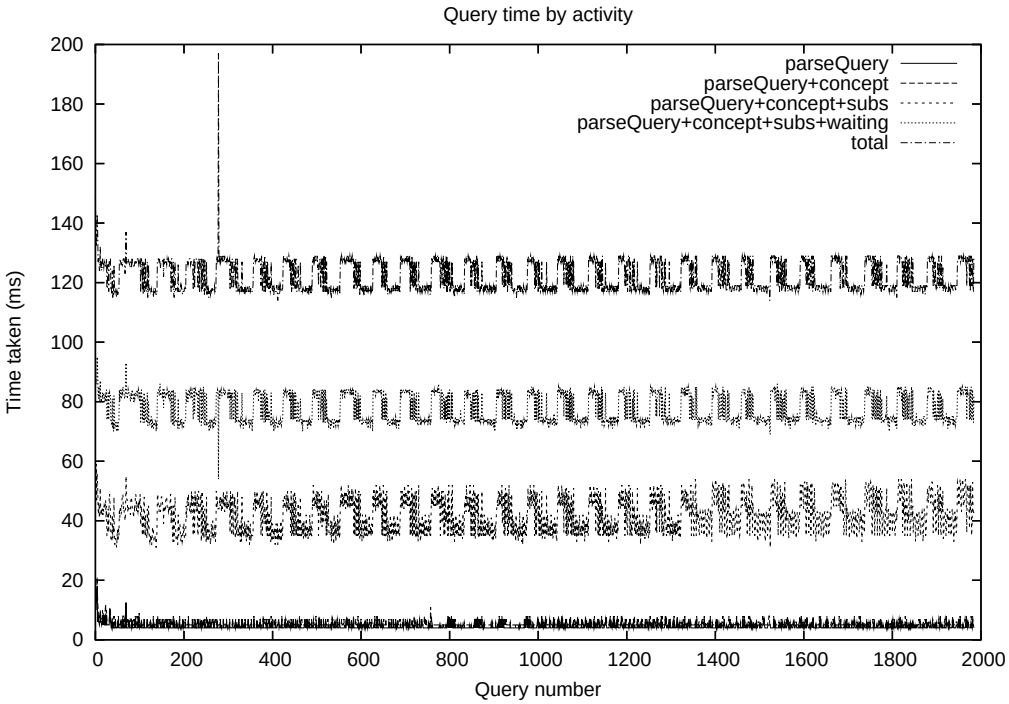

Figure F.4: Deep, bushy ontology queries on local CSN