Delivery Context Access for the Mobile Web

Sailesh Kumar Sathish

University of Tampere
Department of Computer Sciences
Computer Science
Licentiate thesis
May 2007

University of Tampere
Department of Computer Sciences
Computer Science
Sailesh Sathish: Delivery Context Access for the Mobile Web
Licentiate thesis, 80 + 31 pages
May 2007
Supervisor: Professor Roope Raisamo

Abstract

The advent of advanced mobile devices has ushered in a new generation of intelligent adaptive applications. With the mobile web gaining widespread prominence, there is a need to provide more intuitive and interactive services that are customized across a wide range of devices with varying capabilities. Delivery context refers to a set of attributes that characterizes the capabilities of the access mechanism, the preferences of the user and other aspects of the context into which an adaptation service is to be performed. Adaptation services can rely on delivery context information providing customized content. Adaptation can take place on three fronts: content adaptation, presentation adaptation and service adaptation. For applications and services to perform adaptation, there has to be an efficient and standardized mechanism for accessing delivery context information, especially for the mobile web. Towards this goal, a framework servicing mobile web for delivery context access has been developed. The consumer API part of the framework, developed jointly with industry partners, called Delivery Context: Interfaces (DCI) is undergoing standardization within World Wide Web Consortium (W3C) and had reached candidate recommendation status in 2006. The framework addresses both consumer applications that use context data and provider services that provide context data. In addition to the framework design and implementation, examples of adaptive web applications utilizing delivery context information are presented. Since context provision is as important as context consumption, details of two context provision services are also described. The first is a SIP-based context provision service and the latter is a conceptual design of an agent-based context provision service. The work presented is concluded by providing insights into future extensions and research aspects needed for successful and widespread adoption of the framework.

Acknowledgement

This licentiate thesis work was made possible through ample guidance, contribution and support from colleagues, friends and family.

First, I thank Professor Roope Raisamo from University of Tampere (UTA) for being my supervisor and examiner for the thesis. I thank Professor Tarja Systa of Tampere University of Technology for her role as the first examiner of the thesis. Professor Tarja and Professor Roope's invaluable comments, guidance and commitment have helped refine the thesis in both technical and readability front. Professor Roope has also been instrumental in guiding me with the studies and university procedures.

I thank Professor Kari-Jouko Räihä of UTA for accepting me within UTA PhD programme. I thank all lecturers at UTA under whom I have undertaken studies and for helping me widen my boundaries of thinking. I especially thank Dr Zheying Zhang from UTA for her help with metamodeling and Dr Heimo Laamanen from Helsinki University of Technology for agent-based technologies.

None of this work would have been possible without help and guidance from my colleagues at Nokia. Dr Ramalingam Hariharan at Nokia Research Center was my primary technical supervisor within Nokia. Dr Ramalingam was persistent in motivating me to take up studies along with my work and encouraging me move forward both in official and private life. My long association with him has been most productive, and motivating. My whole hearted thanks to Dr Ramalingam. My thanks to Dr Jari A. Kangas at NRC who was instrumental in fully supporting me with the Licentiate program, allocating the needed funds and hours for university whenever required, proof reading and approval of the thesis. My thanks to Janne M. Vainio, Akos Vetek, Dr Christian Prehofer, Dr Cristiano Di-Flora and Antti Lappetelainen for their total support towards this work A special thanks to Olli Pettay for his immaculate knowledge of web technologies and guidance with Mozilla extensions. I have had immense support and guidance from Dana Pavel and Dr Dirk Trossen, formerly of Nokia Research Center (NRC) with context awareness and service provisioning. Dana Pavel and Dr Dirk Trossen supported my work on DCI and significantly broadened my outlook towards network based provisioning services. Our association has helped in developing integrated frameworks, research papers and joint inventions. My wholehearted thanks to Dana Pavel and Dr Dirk Trossen for everything they have done for me. My thanks to supporters of my work within Nokia and in standardization bodies especially Dr Pertti Huuskonen, Bennett Marks, Shahriar Karim and Arthur Barstow. A special thanks to Dr Kari Laurila and Pekka Kapanen, my first manager at Nokia. To colleagues at NRC who have provided me with direct and indirect support at both professional and personal level. A special thanks to Nokia Corportation for the policy of continuous knowledge renewal and support for innovation.

To my friends and colleagues at World Wide Web Consortium (W3C). My fellow editors of DCI specification, Dr Keith Waters and Matt Womer (France Telecom), Rafah Hosn (IBM), Dave Raggett (W3C/Volantis), Max Froumentin (formerly W3C) and Stephane Boyera (W3C). A special thanks to Dr Rhys Lewis (Volantis) for heading the Device Independence Working Group, his expert help on the DCI specification amongst others, and for letting me into the DCO work. To Dr Rotan Hanrahan (MobileAware), Daniel Applequist (Vodafone), Rolland Merrick (IBM), Augusto Aguilera (formerly Boeing) and Cedric Kiss (W3C) for their standardization support and making the W3C meetings much merrier! Thank you.

To my friends at Tampere for all their support and making our stay in Finland so enjoyable!

To my parents for sparing no efforts and making me believe. Their undying support and love has helped me through in every walk of life. To my dear sister and brother-in-law for all their love and support. To my wife's family for all their love and support.

Finally, to my wife, Nithya for no words can express my gratitude for her deep commitment and support in making this work possible. I thank her for bearing the long hours, her explicit inputs, criticisms and strong opinions on the work. Her determination to finish her master's thesis was another major inspiration to make me finish my licentiate. And finally, for the laborious task of fine grained proof-reading, refining and organizing the thesis for me.

Sincerely,

Sailesh Kumar Sathish.

Contents

1. Introduction 1
2. Related Technologies 4
2.1 HTTP 6
2.1.1 HTTP headers 6
2.1.2 HTTP negotiation 7
2.1 CC/PP 8
2.2 UAProf 9
2.3 WURFL 10
2.4 Media Queries 11
2.5 SMIL 11
2.6 ICAP 11
2.7 Others 12
3. Adaptation Architecture 15
3.1 Adaptation in Multimodal Framework 16
3.2 Browser Adaptation Architecture 18
3.3 Extended Adaptation Framework 21
4. Delivery Context: Interfaces 25
4.1 DCI Property Hierarchy 26
4.2 DCI 28
5. Delivery Context Provider Interface 33
6. Dynamic Device profile (DDp) 37
7. Security and Trust 41
8. DCI Implementation 47
9. Context Provision 52
9.1 CREDO: A SIP-event based framework for context provisioning 52
9.1.2 Authorization policy component 55
9.1.3 Discovery Component 55
9.1.4 Ontology Component 56
9.2 CREDO and DCI 56
9.3 Agents and Context 57
9.4 Ontology for Context Domain 66
10. Dynamic Applications 68
11. Conclusion 71
References 75
Appendix A 81
Appendix B 83

Appendix C 86
Appendix D 89
Appendix E 92

1. Introduction

The Internet has come a long way since its inception more than a decade ago. As the Internet age expands, the ability to access content and services is taking on new dimensions. The requirement for ubiquity has gained widespread prominence and to support this, device manufacturers are coming out with a plethora of new devices with varying sizes and capabilities. Of particular importance to ubiquity is mobile information access. The improving connection speeds and access technologies are leading to richer content explosion and user experience. The addition of mobility means that devices would be present with users at all times. This means, services can be deployed that can generate cues on user environment and intentions. The next generation of applications would leverage user environment and system data to provide customized and adaptive services tailored to the particular context.

Context, according to Dey [2001], is defined as "*any information that can be used to characterize 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*". Context, in a generic sense can be considered to be any data that can be static or dynamic characterizing a particular instance of an application session but can well encompass situations having relevance beyond user interaction. An example would be a system middleware that would react to a bandwidth change reflecting a user preference setting to keep connection costs to minimum. Another example would be to automatically switch to another network provider that would bear no implications on the user's interaction with the application but would have an impact elsewhere. There have been several attempts to define context such as those by Brown et al. [1997], defining context to be the user's environment that the system is aware of. Schilt et al. [1994] and Ryan et al. [1997] support similar definitions restricting context to data such as user's location, identities of nearby people and objects, time, date, seasons, temperature for example. Dey [2001] supports more user context such as user's emotional state, focus of attention and others in addition to the ones enumerated within environmental context. Dey and Abowd [2000] describe further the need for automated availability of context data to a computer's run time environment. "Context", in this work, is defined as *any information that would be deemed relevant to the whole application session aiding adaptation to provide customized service to the user*.

Here, the adaptation of mobile web applications based on delivery context data along with relevant technologies will be explored. Delivery context [Gimson, Sathish and Lewis, 2006], refers to context data that would characterize a particular session.The term delivery context [Gimson, Sathish and Lewis, 2006] within web context can be defined as:

A set of attributes that characterizes the capabilities of the access mechanism, the preferences of the user and other aspects of the context into which a web page is to be delivered.

Adaptations would take place based on delivery context at several fronts. According to Sathish and Pettay [2006], the different types of adaptations are:

- Content adaptation that can take place at an adaptation server, a proxy, the content server or even the client.
- Presentation adaptation where the adapted content would be presented accordingly with the environment that characterize the interaction.
- Service adaptation where the client platform and the application utilizes delivery context information to provide services to the user.

In order for web applications to perform adaptation, a standardized mechanism for delivery context access is necessary. In addition to that, there should also be mechanisms for data providers that are device resident as well as network based to provide data to the context access service.

In order to realize an efficient adaptation platform for mobile web, I have developed a framework that is presented in this thesis. As part of framework development, I have contributed extensively to standardization activity within the World Wide Web Consortium towards a specification for delivery context access [Waters, Sathish et al., 2006]. I am also a joint editor for W3C Device Independence Group's Delivery Context: Overview work [Gimson, Sathish and Lewis, 2006]. The framework covers different aspects of adaptive web such as support for context consumers, context providers, generation of dynamic profiles for content adaptation, security and management aspects. With respect to the framework, I have developed a security model that was filed for patenting by Nokia Corporation. I have authored four papers [Sathish and Pettay, 2006]; [Sathish, Pavel and Trossen, 2006]; [Sathish, 2007]; [Sathish and Di-Flora, 2007]; on context-based adaptation frameworks. In the area of context provisioning, I developed an agent-based metamodel framework and also integrated a SIP-event based provisioning system developed by colleagues at Nokia Research Center to my implementation of the framework.

The thesis is organized as follows: the second chapter introduces some general adaptation techniques and methods for providing delivery context access. Chapter 3 presents the framework that was developed for adaptive web applications. Chapter 4 provides details of the W3C activity that is at the core of the adaptation framework. Chapters 5 and 6 provide detailed description of context provisioning support and device profile modules of the framework. Chapter 7 looks at security and trust issues while Chapter 8 provides details of the implementation that was carried out. Chapter 9 provides description of two context provision systems: the first is a SIP-based context service infrastructure and the second is a conceptual model of an agent-based context

provisioning system. Chapter 10 provides some examples of adaptation applications that can benefit from such a framework. Chapter 11 summarizes the work presented and provides insight into future directions of this work.

2. Related Technologies

This chapter looks at some adaptation technologies that are already prevalent in the web world. Here, technologies that are used for supplying delivery context information are surveyed. It is envisaged that with the advent of powerful mobile computing platforms, the usage of delivery context would become highly relevant for both client side and server side adaptation systems. This, in turn should provide the framework for realizing the end goal of fully authoring device independent content.

Delivery context would involve a range of potential characteristics that identify the particular environment in which the content is to be used. Some examples are hardware characteristics, software, location, user agent (browser) characteristics, connection, temperature, noise, light, trust, and privacy. The delivery context vocabulary has to be captured by an ontology that should be standardized and extensible to a large extend. Ontology describes the concepts and relationships within a domain. This goes further than a vocabulary which is a list of terms that has been enumerated explicitly. Taxonomy is a vocabulary that has been organized in a hierarchical structure. The ontology can represent taxonomy with additional information such as relations between organized concepts in vocabulary, concepts and behavior. An example of delivery context ontology is presented in Appendix E.

It is possible to organize delivery context data sources (providers) in a taxonomical form. Depending on the data provider types, there can be different levels in the hierarchy i.e. the representation model with a minimum level of two. The top node forms the root node of the hierarchy. The first level forms the groupings under which sets of properties can be grouped. The groupings in the first level can be hardware characteristics, software characteristics, user characteristics etc. The ontology would describe what properties can be grouped under which as well as relations between the different properties.

A particular question that arises during commercial deployment is the issue of ontology management. Ontologies need to be standardized as well as managed (addition of new properties, metadata of properties, value and data type management etc) so that interoperability can be assured. For web applications that depend mostly on scripts for application adaptation, having a standardized vocabulary is a must. Property relationships and hierarchical organization is important for platform management as well as application developer (consider using an XPath expression within a web script).

Currently, there are a lot of standardized ontologies such as UAProf [UAProf], Dublin Core Metadata Initiative [Dublin Core MI], and Friend of Friend [FOAF] that have been standardized or widely accepted. Since delivery context data can be dynamic and new data sources spring up everyday, having a complete ontology that describes all

provider types may not be feasible. Device manufactures can add new data sources to their offerings in order to provide differentiation that can break a standard ontology creating problems for application developers. So, any ontology that should address delivery context should support static and dynamic data, be extensible and standardized to a large extend.

The limitations of having a standardized ontology can also be compounded by business models that can drive a particular domain. For example, a network operator would want to control the particular data sets that an application can access (based on agreements between service provider and network operator), provision of network based context data as opposed to local data sources, user data privacy and management of control and security policies. Such operations would have a bearing on how the ontology is managed, i.e., whether the local middleware supporting dynamic data provisioning or through a network based management mechanism. The best bet would be that there would be a distributed vocabulary management with a standardization body standardizing the first few levels of the hierarchy with provisions for device manufacturers, network providers, service providers as well as application authors and users to add to the data model (if required and allowed by the domain policies). The vocabulary management, business models and operation framework for ontology updates and device management is outside the scope of this thesis.

As stated before, one of the most important aspects involved in adaptation process is for a server or adaptation service such as a proxy to know the characteristics about the device requesting the content. The term *user agent* is generally used to describe the software entity (such as a browser) requesting the content. Once the capabilities of the device (or user agent) is known, the content is adapted accordingly so that the best possible interaction can be presented to the user in the most optimal way. There are generally two ways in which content can be adapted: **selection** and **transformation** [Lemlouma and Layaida, 2003]. Selection is the process by which a content origin server or adaptation entity selects the best possible candidate from amongst a finite set of existing representations. With transformation, there would be a single data model from which a suitable presentation model would be derived. The dimensions of the presentation model would be dictated by the characteristics and capabilities of the user agent. An example would be a data model stored in XML format and transformed into xHTML or WML based on the type of user agent (such as an HTML browser or WAP browser) requesting the content.

There are several existing technologies that deal with some aspects of the entire adaptation processing model. The need to understand which presentation model to be delivered to user has been recognized since the early days of the web. Some related technologies that aid adaptation process are described below.

2.1 HTTP

HTTP [HTTP] or Hyper Text Transfer Protocol defines a protocol that is used for informing entities about format and transmission of content, and what actions browsers and end servers should perform as response to various commands. As a result of user initiation, the browser sends an HTTP command to a server requesting a web page. HTTP, as such, is a stateless protocol where any new command is independent of a previous command. Each command is treated as independent at the server and the server (usually) sends back a new page as response to a web request.

2.1.1 HTTP headers

The hyper text transfer protocol is the basis for most current web based content delivery. HTTP defines a set of accept headers that can be used to describe the characteristics of the requesting device. The browser uses the accept headers to inform the server about the capabilities and preferences of the device, in particular the user agent, for the requested content.

Standard HTTP 1.1 includes the following headers:

- Accept: media types (MIME) accepted by the user agent,
- Accept-Charset: character sets accepted by the user agent,
- Accept-Encoding: preferred reply encoding (compression) for the user agent, and
- Accept-Language: natural languages preferred by the users.

In addition, the HTTP request could also contain information about the user agent (browser) such as the manufacturer, version number and name. There could also be additional information about other characteristics of the device. Additional information about mobile devices requesting the content, such as device hardware and browser that is being used can be included in the user agent string. There are no particular standards about the format of the user agent string. However, sophisticated algorithms do exist that can process a wide range of user agent strings thereby identifying the particular device and its capabilities. Identifying the particular device helps adaptation services in fetching capabilities of the device from external repositories. Once the capabilities are obtained, the adaptation process can provide adapted content customized to that device.

The advantage of using HTTP based model for delivery context information conveyance is its widespread adoption and familiarity with developers. The major disadvantage is that it is not extensible. Also, in most cases, the user agent strings, the amount and type of information they can convey are not standardized.

2.1.2 HTTP negotiation

HTTP [HTTPneg] negotiation means that content is negotiated first before it is downloaded to client from server. HTTP supports two types of negotiation: *server-driven negotiation* and *agent-driven negotiation*. These two kinds of negotiation are orthogonal and can be used separately or in combination. When combined, it forms a *transparent negotiation*, where a cache uses the *agent-driven negotiation* information provided by the content origin server to provide *server-driven negotiation* for subsequent requests.

With *server-driven negotiation*, the server selects which content to send to client based on information present in the HTTP accept-headers. The accept-headers used (as described earlier) are Accept, Accept-Charset, Accept-Encoding, and Accept-Language. Each of these headers provides an additional dimension in negotiation for content adaptation. Browser rendering capabilities, language capabilities, encoding preferences and user preferences can be conveyed to a limited extent through these headers. In addition, a set of preferences can be conveyed through the headers with associated quality values. An example for conveying language preferences through the accept-language header is shown below:

Accept-language: en; q=1.0, fr; q=0.5

The above statement shows that English language is preferred over French. However, if English is not available, French too can be rendered by the user agent even though with lesser preference.

There are some disadvantages to *server-driven negotiation*. The main disadvantage is that there are limits to the amount of information that can be conveyed through HTTP accept headers. Secondly, it is inefficient for a user agent to describe its full capabilities to a server for every request it makes. This can be alleviated to some extend through server side algorithms that determine the type of device and user agent. Based on this, external repositories can be consulted that provide more detailed information on the requesting user agent. Third, it complicates implementation of origin servers and algorithms for generating responses to clients. Server-side negotiation also creates problems with caches served by multiple devices.

In contrast, with *agent-driven negotiation,* the user agent selects the content that will be rendered. The server presents the user agent with a set of alternatives out of which the user agent chooses the best alternative in-line with its capabilities. It then requests the server for a particular content from the available list. The disadvantage of such a system is that it introduces additional delay through multiple request-response round trips.

Some proprietary mechanisms do exist for informing server side entities about client capabilities using HTTP extension methods. These generally introduce new headers (within HTTP GET method) or within the body of an HTTP request (HTTP POST method). Other alternatives such as including SOAP (Simple Object Access Protocol) messages etc as body attachments are also being used. The HTTP Extension Framework [HTTPex] is now a standard that aims to bring existing extension practices within a single extension framework that is interoperable. There are also other inline HTTP based adaptation methods that use simple form data for getting device or user information and transmitting those back to server as part of a GET or POST method. Server side scripts (such as CGI) can then process these inputs and send back appropriate content.

2.1 CC/PP

CC/PP or Composite Capability/Preference Profile [Klyne et al., 2004] is a W3C standard based on Resource Description Framework (RDF) [Brickley and Guha, 2004] used for specifying metadata. A CC/PP profile specifies the capabilities of a user agent. This allows for adaptation entities to know the full extent of a device's capabilities and thereby produce optimized XML (or other) content as a best offering for a wide variety of user agents. When expressing device capabilities, CC/PP has the flexibility that HTTP negotiation lacks. The RDF-based framework allows for the creation of whole new vocabularies, enabling an infinite extension capability when describing device and agent capabilities.

CC/PP is used for describing and managing profiles related to device or user agents along with their software profile, hardware profile, user profile and other characteristics. The goal behind developing CC/PP was to provide a vocabulary neutral framework using which a device independent web model would become feasible. CC/PP is designed to work with a wide variety of web-enabled devices ranging from cell phones to PDA's to desk top machines. CC/PP itself does not define what the behavior should be when a profile gets exchanged between two entities.

CC/PP is vocabulary independent. This means that CC/PP provides a generic framework within which other bodies (such as standard bodies or vendors) can define their own vocabulary through use of the RDF schema language RDFS. The most extensive vocabulary written based on CC/PP is the User Agent Profile or UAProf (described in the next section).

CC/PP profiles are designed to be accessible via the web, for example from the hardware or software vendor. Thus, a user agent can send in an HTTP request to an origin server where it also includes the URL for its profile. The server uses this URL to fetch the profile from an accessible repository. The profile is then parsed to gather

information about user agent capabilities. This reduces the amount of information that must be directly sent from the user agent or a proxy to the server, which is an important factor for bandwidth constrained mobile devices. The CC/PP approach is better than other approaches as it provides an extensible framework for describing properties of user agents directly rather than identifying a particular browser or user agent type.

2.2 UAProf

UAProf [UAProf] or User Agent Profiles is essentially an XML file listing the capabilities of the device the UAProf represents. UAProf has been defined by OMA (Open Mobile Alliance [OMA], formerly the WAP forum) for WAP-enabled terminals to enable mobile web convergence with that of the web. UAProf is based on the CC/PP framework that uses RDFS for schemata definitions. CC/PP is a generic RDF based framework that does not define any vocabulary. UAProf builds on top of CC/PP defining vocabularies for different characteristics of a device. These include hardware characteristics such as CPU, memory, screen size, type of keypad, software characteristics such as browser, operating system, version numbers and others.

Each vendor maintains repositories of UAProfiles where each UAProf describes capabilities of the device it represents. When a user agent makes a request for content, it also sends the URL for the UAProf of the device it is running on. This can happen via certain headers within an HTTP request. WAP 1.2.1 [OMA WAP Specification] recommends transporting UAProf information using the HTTP extension framework [HTTPex] which was originally suggested for CC/PP [CCPP-exchange]. WAP defined the WSP protocol, which includes a compressed encoding, for use between the phone and the gateway onto the Internet. Due to the lack of implementations for HTTPex, WAP 2.0 instead recommended an extension of HTTP1.1 as an Internet protocol that uses custom headers. Typically, the URL for UAProf is found in *x-wap-profile* header within a HTTP request.

A UAProf contains a number of components and each component contains a number of attributes. Components will be high level containers such as HardwarePlatform, SoftwarePlatform, NetworkCharacteristics, BrowserUA, WAPCharacteristics and PushCharacteristics. The properties hosted by these components form their attributes. An example of a UAProf showing the screen size attribute within a HardwarePlatform component is shown below.

```
<prf:component>
        <rdf:Description rdf:ID="HardwarePlatform">
              <rdf:type rdf:resource=
  "http://www.openmobilealliance.org/tech/profiles/UAPROF/ccppschema-
        20021212#HardwarePlatform"/>
                    <prf:ScreenSize>208x208</prf:ScreenSize>
...
        </rdf:Description>
</prf:component>
```

The latest version of UAProf v2.0 has been defined by OMA based on the latest versions of RDF and RDF schema.

2.3 WURFL

WURFL or Wireless Universal Resource File [Passani and Trasatti, WURFL], provides a comprehensive repository of device profiles covering a wide variety of devices. There are over 400 devices supported with as many profiles in the WURFL repository. The purpose of WURFL is to collect as much information about WAP devices so that developers can write applications that would run on the different types available. WURFL is open source, so anyone with profile knowledge can update or add new profiles to the database. WURFL has been developed as an alternative to UAProf. WURFL uses a “family of devices” principle where devices that fall within particular groups share capabilities of that particular family and the difference is noted separately. This makes the WURFL file compact and easier to maintain.

The main difference between WURFL and UAProf is that UAProf is created and maintained by device manufacturers. UAProf needs third party services to host and maintain while WURFL can be installed at a developer site. WURFL depends on developers to provide updates to its repository file as it is open source. This sort of ensures upto date and accurate information (even though WURFL does not guarantee it). WURFL also takes data from other sources such as UAProf for profile updates. Properties in UAProf are limited to those in the vocabulary whereas WURFL can extend beyond those provided by the manufacturer. However, WURFL files are much longer than those of UAProf as they contain information about a plethora of devices whereas UAProf targets a single device. Developers need to download the WURFL repository (so that the content/origin server can access it) and maintain periodic updates to keep the repository upto date. WURFL has its own XML format for device description.

2.4 Media Queries

CSS or Cascading Style Sheets [CSS2] can be used in conjunction with web pages to provide custom presentations. CSS is a separate page (or can be embedded within the web page) that provides the browser with information on how the web page is to be presented. This provides a separation of presentation information from the actual content. CSS2 defines a set of media types such as Aural, Braille, Embossed, Handheld, Print, Projection, Screen, TTY and TV. Media Queries build upon these types in CSS2 allowing conditional selection of presentation styles based on the media type detected. The style selected can thus be made conditional based on the characteristics of the device. The 'display' property of CSS can also be used to completely leave out certain elements in the markup if needed.

Media Queries, like CSS, are supposed to be processed at the user agent. There can also be mechanisms where media queries get processed at the origin servers or intermediaries. Such mechanisms have the advantage that less content is sent to the user agent and no user agent-side processing is required. However, this also requires that device specific characteristics be sent along with the request and that there needs to be correspondence between the device characteristic vocabulary and those supported by Media Queries.

2.5 SMIL

SMIL or Synchronized Multimedia Integration Language [Bulterman et al., 2005] is a language for specifying audio-visual presentations. SMIL is an XML based language that is a W3C standard, latest being version 2.0. SMIL 2.0 has been defined as a set of markup modules that can be integrated into specific language profiles. SMIL also defines some basic device characteristic vocabulary that can be used to check device capabilities in order to adapt and coordinate media presentations. SMIL defines a BasicContentControl module that defines the required device characteristics that can be used to control SMIL presentations. The characteristics are fed to the SMIL player by the runtime environment. This is similar to Media Queries where the capabilities are queried to adapt presentations. The characteristics defined as part of the specification involve presentation-related capabilities such as screen size, network bandwidth, text and audio captions, as well as system-related characteristics such as CPU and operating system identity.

2.6 ICAP

Internet Content Adaptation Protocol [ICAP Forum] is a protocol put forth by a consortium of industry players aimed at off-loading content adaptation and other value-added services to edge services from origin servers. Web servers are expected to provide only content to the end devices whereas other services such as adaptation, authentication, content translation or filtering happen with dedicated servers running the ICAP protocol. At the core of this process, there is a cache that will proxy all client transactions and will process them through ICAP/Web servers. Off-loading services from web servers allows the deployment of scalable and efficient services compared to raw HTTP throughput with overloaded servers processing extra tasks. ICAP can be seen as a “lightweight” HTTP based remote procedure call protocol. All client HTTP requests get proxied to an ICAP server where the request and/or response would get modified before it is sent back to client. ICAP thus allows clients to send HTTP messages and response (content) to ICAP servers for adaptation.

2.7 Others

In addition to the more popular standards, several approaches have been proposed that addresses different aspects of content adaptation. This section briefly describes three such technologies and provides an overall summary of all the approaches described.

Transparent Content Negotiation [TCN]: was first proposed as an experimental protocol in RFC 2295. Transparent negotiation uses both HTTP server-driven and agent-driven negotiation mechanisms, together with a caching proxy that supports content negotiation. The proxy requests a list of all available representations from the origin server using agent-driven negotiation, then selects the most appropriate and sends it to the client using server-driven negotiation. However, this technique has not been widely implemented.

Conneg: The IETF Content Negotiation [Conneg] working group focussed on defining a set of features which would form the basis of negotiation.

MPEG-21: The MPEG-21 [MPEG-21] (ISO/IEC) framework is intended to support transparent use of multimedia resources across a wide range of networks and devices. The fundamental unit of distribution is the 'digital item', which is an abstraction for some multimedia content with associated data. One aspect of the requirements for MPEG-21 is Digital Item Adaptation which is based on a Usage Environment Description. It proposes the description of capabilities for at least the terminal, network, delivery, user, and natural environment, and notes the desirability of remaining compatible with other recommendations such as CC/PP and UAProf.

To summarize, several adaptation technologies are available that try to address some part of adaptation processing model. Adaptation can be carried out at the content server, a mid proxy or at the client side. The key to adaptation is to let the adaptation

service know about client capabilities, enabling content transformation that can be most suitably rendered. The most popular HTTP mechanism provides certain extensions to HTTP headers through which characteristics of the user agent can be conveyed. However, the amount of information that can be conveyed is very limited. Any extensions would also require standardizing the new headers.

User Agent Profile [UAProf] uses a vocabulary set to describe the capabilities of a user agent. The UAProf of a user agent is meant to be network resident. The user agent conveys the URI of its profile through an HTTP request. The profile is then parsed to know the device characteristics. UAProf provides better description of the user agent than is possible through HTTP based mechanisms. However, the static nature of UAProf and its support for extensibility is the main problem. The profile does not necessarily reflect the exact characteristics of the user agent since modern devices allow users to upgrade their software including new versions and more capable browsers.

WURFL is another mechanism similar to UAProf where device descriptions are available. WURFL is open source and enable developers to extend the vocabulary thereby providing more and accurate information. Again, the WURFL profile is static in nature and do not reflect the features of a personalized device. Other technologies rely on more direct feature access at the user agent. Technologies such as Cascading Style Sheets [CSS2] and Synchronized Multimedia Integration Language [SMIL] define their own API's for gathering user agent characteristics so that run time adaptation is possible. The drawback is that the vocabulary such services use is limited and not extensible to support new data sources when they become available. Also having fixed API's to device property access means that, manufacturers have to provide specific support for each property.

Transparent Content Negotiation [TCN] performs proxying service choosing the best available content for client presentation from an available list of content choices from origin server. The content that is sent to a client may not be the most appropriate but only the best fit available within that context. Conneg and MPEG-21 propose description of a set of features that would form the basis for negotiation between a user agent and an adaptation service. The drawback of such systems is their fixed vocabulary and the static nature of the values that are exposed.

The most accurate information regarding client capabilities resides within the client. The client should be aware of its current system characteristics as well as those of its environment. This dynamic nature of the environment should be accurately reflected to an adaptation entity thereby ensuring the best adaptation service. When new properties are added, they should be reflected in a transparent manner so that services can cater for such extensions. Adaptation services should be capable of polling client characteristics specifically, those dynamic properties that can be best addressed by the requested

content. The same mechanism should be capable of supporting both server side and client side adaptation along with support for run time adaptation at the client side.

3. Adaptation Architecture

The aim of an adaptation service is to provide adapted content and services that are customized to the particular situation characterizing the user. The input to adaptation mechanisms can be device characteristics, user input, and current context, including system and environment data. The type of adaptation mechanism would depend on the type of services requested. As mentioned in the introduction, adaptation can be provided on the following fronts:

- adaptation of content based on device characteristics,
- adaptation of presentation of content on device, based on system and environment data, and/or
- adaptation of services based on context data.

Based on current device capabilities, it can be argued that all types of adaptation can take place either at server side, client side or both. In a split adaptation process, a network based service can perform a first level adaptation based on client profiles as outlined by Sathish and Pettay [2006], and the client side adaptation mechanism can conduct a more fine grained adaptation based on specific device properties. Such distributed adaptation services can take place if the server or a proxy mechanism relies on static device profiles for adaptation and the client side can rely on more dynamic updates from the system to perform better adaptations.

Presentation adaptation refers to how the content is presented to the user and how the user may interact with the content. Distributed presentation adaptation depends on the user interface capabilities of the client device as well as the content itself. Traditional user interfaces use unimodal interfaces such as graphical user interfaces (GUI), keyboard, mouse, or there can be speech only interfaces, touch screen inputs etc. Multimodal interfaces are those that combine multiple modalities to provide a combined input/output capability. With multimodal interfaces, users can interact with the application through simultaneous (if supported) multiple modalities such as speech, gesture, gaze, text input etc. Using such simultaneous modalities on a resource constrained device such as a mobile phone would mean that some of the modality processing has to be distributed in the network. An example would be a distributed speech recognizer with a light-weight front end supported by a suitable back end. Since devices are mobile, it is imperative that sessions with distributed services would be dynamically set up and released. This would have an impact on the way information is requested or presented to the user. Even with unimodal platforms such as a GUI browser, there could be presentation adaptation that can be dynamic based on device orientation, browser settings, user profiles (people with eyesight problems can be presented with better fonts and lesser content) amongst others.

Adaptation of application services would benefit the most through use of a standard framework for context access. Applications can access device, environment and user context to provide adapted services to the user. An example would be using the GPS coordinates and calendar data simultaneously (such as meeting information) that indicates the user's current location to automatically activate a phone profile. Applications can access sensor data, connectivity options, software and hardware characteristics that can provide good basis for adaptation. The service adaptation itself can be provided directly by the client device or where appropriate, new sessions can be established with remote services using context data and other information that may be needed. The client device would act as a session manager and render the information.

In providing application access to device context data, security and privacy are major issues that need to be addressed. Applications should not be granted access to data that the user may consider private or granted only on a trusted basis. Applications should also have limited access to device property access mechanisms and based on trust level, should not be allowed to modify any data on the device.

The following sections present more information on frameworks that are specifically aimed at user-agent based adaptation services.

3.1 Adaptation in Multimodal Framework

As explained in introduction, multimodal platforms allow users to use multiple modalities simultaneously or sequentially depending on the underlying framework and platform capability. Multimodal user interfaces can be provided for all applications, either on a per application basis, or as a standard service by the underlying platform. Since this work concentrates on browsing context, a multimodal browsing framework and context based user interface adaptation is presented.

The W3C's Multimodal Interaction Working Group (MMI) is one of the main standards proponents for bringing in a standardized framework enabling inter-working of different independent components. The W3C MMI's multimodal framework [Barnett et al., 2006] is shown in Figure 1.

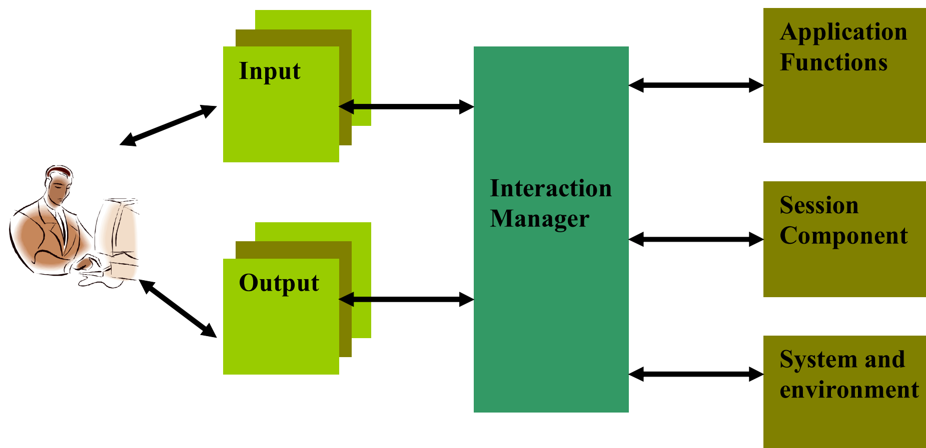

Figure 1: The multimodal interaction framework (MMI, W3C) [Barnett et al., 2006].

Figure 1 shows the MMI framework for multimodal browsing. The input module allows the user to interact with the application using multiple modalities. Examples of modalities are speech, graphical UI, text input, touch, pen input, gestures, gaze input etc. The input module deals with direct input from the user. Each modality that is supported may (in most cases) have their own processors that interpret the user action. The processors for each modality can reside on the client device or on the network. For those processors residing on the network, a suitable front end for gathering input and sending to back end is needed. An integrator component (not shown here) integrates the input from each of the modalities based on some integration rules or patterns and feeds the integrated input to the interaction manager.

The output module provides output to the user in multiple modalities. The output provided to the user can be simultaneous or sequentially presented in each modality. The output is given by the interaction manager to the output modules. The output modules can be split into a presentation generator and a rendering engine. The presentation generator creates the content to be presented for a particular modality while the rendering engine renders the generated output content to the user. The generated content for each modality can also have additional styling that would determine how they are presented to the user. The style rules can be attached with the generated output or present at the rendering engine as default.

The interaction manager is a logical component that is responsible for coordinating input and output and application logic. The interaction manager can provide data management functionalities and flow control and interacts with the user interface objects. The interaction management functionality can be provided by a host

environment dealing with one particular modality or it can be split between the modality components. The interaction manager is responsible for synchronizing the data model of the application and synchronizing input/output where applicable.

The session component is responsible for providing session management functionalities for the platform and the applications. The session component can be used for session establishment with remote services that can be used in distributed multimodal processing and for services that may be required by the application. The session manager can be used for replicating the state and synchronization across multiple devices in a multidevice scenario. The session manager can also be used for finding resources, querying resources and even offering distributed services for multimodal processing.

The system and environment component is responsible for providing the framework with all data related to the system and environment state. The system and environment component would also encompass all profile data such as user profile, device profile, network profile etc. This is the component that will supply context data to the platform. The data that would be provided can be static or dynamic. The framework relies on the system and environment component for all context information and performs dynamic adaptation based on this data. The interaction manager or the browser platform can look for some standard dynamic data such as a topology or a profile change (such as user muting the phone) and perform default adaptation behavior. The application can also subscribe to certain context data that it is interested in such as location data, sensor data etc. The system and environment component would generally be used for client side adaptation but there are also extensions that can be used to generate dynamic profiles that can be sent to server for server side adaptation.

3.2 Browser Adaptation Architecture

This section describes a tightly coupled architecture for adaptive web applications specially suited for mobile devices. This approach is based on an ongoing standardization effort within World Wide Web Consortium (W3C) for client side device context access. The specification, Delivery Context: Interfaces (DCI) [Waters, Sathish et al., 2006] is intended to be used as an access mechanism for context consumers. The browser adaptation architecture is shown in Figure 2.

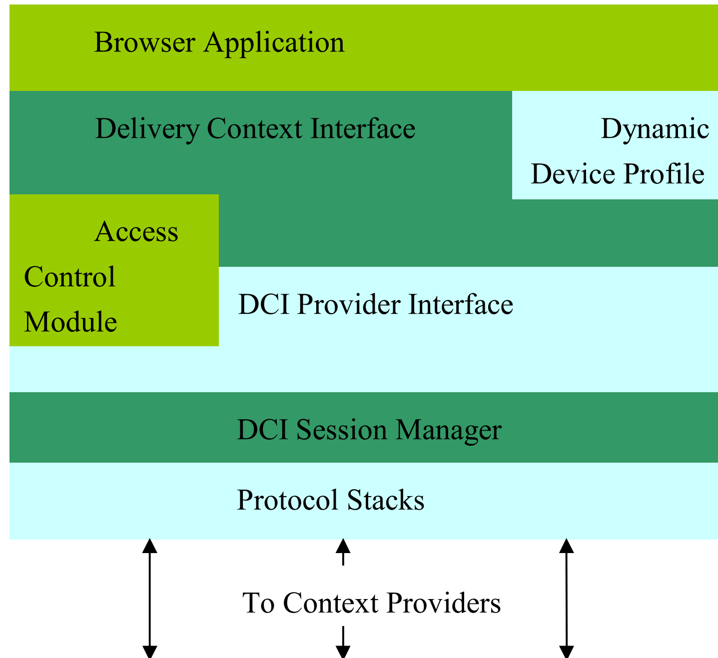

Figure 2: Delivery Context Adaptation Framework [Sathish and Pettay, 2006].

The architecture for device context access and content adaptation is shown in Figure 2. The access mechanism for context access is the DCI. The provider component exposes an API for context providers to supply data to the DCI component. The dynamic device profile component generates an XML/RDF serialization of the client device delivery context to the adaptation server, or proxy that performs device specific content adaptation.

It is to be noted that any application can use the DCI context provisioning system to access device data. One example would be an Interaction Manager (IM) that employs DCI services in a multimodal session as described in the Multimodal Interaction Architecture document [Barnett et al., 2006]. The DCI session manager is responsible for managing the access mechanism between the DCI module and external devices/properties. The session manager would use different mechanisms for providing access that is platform dependent. It can use protocol stacks for communication with context providers as is mostly done in Linux or Windows environment or use a server/client mechanism that is suitable with Symbian platforms. The access control module determines whether and where to provide access control for external properties within the DCI tree. The access control module in Figure 2 spans the Delivery Context Interface module and DCI Provider Interface module. This is because access control is needed for consumer applications that access Delivery Context Interface module and access control is needed for providers who access through the DCI Provider Interface. The Dynamic Device Profile provides a snapshot of DCI at any point of time by

serializing DCI and is used for server side content adaptation. The Dynamic Device Profile forms part of Delivery Context Interface as it relies on information from Delivery Context Interface in order to serialize parts or whole of delivery context. The DCI specification is explained in more detail in Chapter 4, the Delivery Context Provider Interface is explained in Chapter 5 and Chapter 6 provides detailed description of Dynamic Device Profile approach.

The Delivery Context: Interfaces is a new approach taken by the Device Independence Working Group (DIWG) of W3C as an access mechanism for static and dynamic properties of the device. It is a mechanism that is suited for web applications but can also be adopted with other frameworks because of the generality and extensibility offered. DIWG advocates this approach as this fits as a complementary mechanism to their Composite Capability/Preference Profile (CC/PP) model for server side content adaptation and the delivery context approach described in Delivery Context: Overview (DCO) [Gimson, Sathish and Lewis, 2006] document. DCI, as a client based mechanism can fit within a content adaptation framework where web content can be adapted based on the capabilities of the device. Also, beyond content adaptation, DCI would be used by applications themselves to gather context data and provide application adaptation through simple access methods. This reduces reliance on external services for providing the same information. It is envisaged that an extensive adoption of DCI platforms would enable the generation of a new genre of applications that performs intelligent client-based adaptation services. This would bring about the next generation of user experience with specific applications for mobile devices.

The W3C's Document Object Model (DOM) [W3C DOM 2004] is *a platform and language neutral interface that allows programs (scripts) to dynamically access and update content, structure and style of documents*. The DOM is the mechanism through which the document (well formed XML documents) is exposed to application programs as object model. Through the DOM model, the scripts view the document as a hierarchy of DOM nodes corresponding to each element within a well-formed XML document. The scripts can use the DOM API to traverse and manipulate the document objects. DOM also supports an event system that involves event propagation and handling mechanism for listening and capturing events. The DCI also takes a similar approach to representing device properties in a hierarchical manner organized through a taxonomy that would be defined outside DCI scope. The approach was adopted due to the popularity and familiarity of DOM mechanism among application developers as well as its fit with current browser support for DOM. DCI provides an API for property access by extending the standard DOM interfaces and using the same event mechanism as DOM. DCI mandates the latest recommendation of DOM level [W3C DOM 2004] and DOM event [Pixley, 2000] specifications.

3.3 Extended Adaptation Framework

The adaptation framework in Figure 2 depends on an access control module to provide access rights to DCI. Since DCI is a vocabulary dependent mechanism, providing simple access control may not be enough. There are other issues like integrity management, logical mappings, maintenance of hierarchical relations, security management and vocabulary extensions that have to be addressed. Another concern is device access policy. A significant amount of mobile devices are sold through network (service) providers. Service providers control device access policies and management and as such, need to maintain a certain level of control in accordance with their business models. Usage of management objects for control and management is one such example. A fully fledged framework has to take all these into account. In order to address these, an extended framework to Figure 2 is proposed.

The extension mechanism uses an ontology based management for addressing the issues mentioned above. Ontology describes concepts used in a particular domain that is machine understandable along with relations among the concepts used. Ontologies resemble extended taxonomies that use richer semantic relations among terms and attributes, as well as strict rules about how to specify terms and relationships. Ontologies go beyond controlling a vocabulary and can be seen as knowledge representation models. The often quoted definition for ontology is "*the specification of one's conceptualization of a knowledge domain*" [Ontology]. In simple terms, ontology is a hierarchical taxonomy of terms describing a certain area of knowledge. The ontology can be described using any of the standard ontology languages such as OWL [OWL], DAML+OIL [DAML+OIL], and RDF/RDFS [Brickley and Guha, 2004].

DCI requires ontology for describing the vocabulary for properties and the relations these properties might have to each other. The ontology can be specified by some standards bodies (see UAProf as a standardized ontology), Original Equipment Manufacturers (OEM's), others or jointly managed by multiple entities. The standardization and management of ontologies is beyond the scope of this thesis.

The framework shown in Figure 3 depends on an ontology describing the entire set of vocabularies for properties that can be exposed by the DCI framework to the calling application. The ontology would describe the hierarchical relations (logical such as Software, Hardware, and Location) and the set of properties that would fit under each set. The ontology would be formed partly from standard ontologies such as UAProf schema, Dynamic Profile Extension (DPE) [OMA-DPE] which is an ongoing activity within Open Mobile Alliance, and others. Device manufacturers can provide proprietary property extensions that will not be standardized. It would also be difficult to standardize the entire set of properties possible. Thus, the ontology should be

extensible, in that, the device manufacturer can extend the vocabulary based on new properties that would emerge.

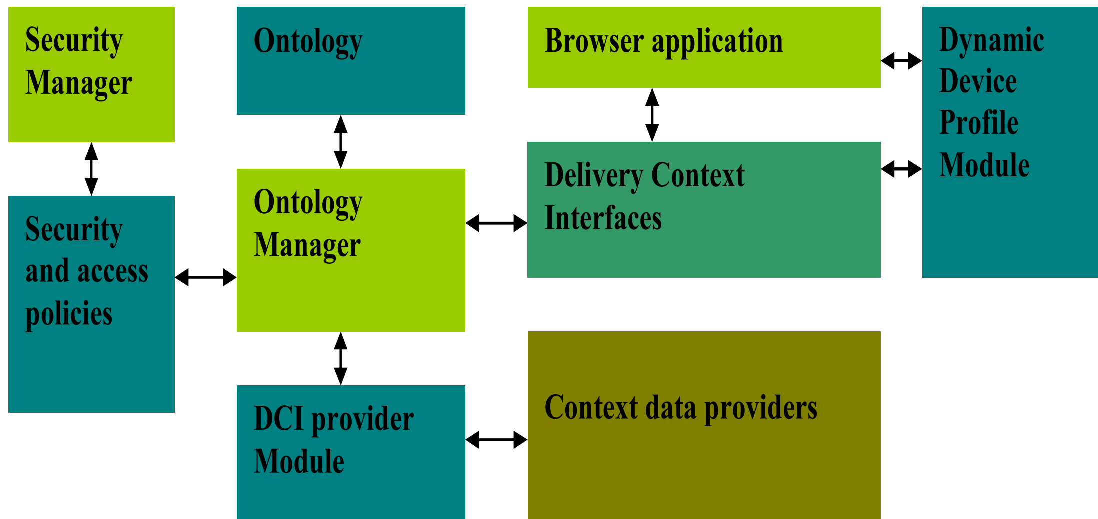

Figure 3: Client side context access framework using ontology based mechanism for access and delivery context management [Sathish, Pavel and Trossen, 2006].

The security and access policy module describes security and access rights policies that can be managed through the security manager module. The security manager may provide access to service, network and/or device manufacturers so that they can control and manage access policies applicable to DCI tree. The ontology manager could also provide similar controls required for management of the ontology to external services. Periodic updates to the ontology can thus be provided through trusted services.

The framework shown in Figure 3 is an extension of Figure 2. The context data providers seek access to the DCI tree through the DCI provider interface. The DCI provider interface (shown as DCI provider module in Figure 3), takes the property metadata (such as OWL-S [Martin et al., 2004] description or RDFS metadata) and queries the ontology manager for DCI tree access. The ontology manager then obtains the access right policy for that particular type of property from the security and access right module. Based on this, the ontology manager checks the ontology and decides where in the DCI tree the particular property should be given access to. This helps protect the integrity of the DCI tree. It then checks the DCI tree to see if a new node needs to be created or whether an existing node matching the same metadata can be overridden. If a new node is required, it creates a new node following the topology constraints and initializes the node (such as parent information). The node pointer is then passed to the DCI provider interface that forwards it to the requesting service provider. In case no access is granted, an empty (NULL) pointer is passed. The DCI

provider module does not permit the providers to directly start providing data. To optimize performance, they can start so, only after receiving a start event from the DCI provider. This event will be triggered by the DCI implementation only when a consumer asks for that property. The property though, would have a node in the DCI tree with the metadata interface describing the services that can be subscribed to by the consumer.

All context providers are issued a unique session ID if the provider has been deemed secure by the access control module. The DCI provider module is responsible for managing the session with the context provider once a session ID has been generated. The context data provider will use the unique session ID that was generated for all subsequent communication with the DCI provider module.

To summarize, an adaptation framework based on DCI is described. The framework is aimed at supporting adaptive web applications through extensions to browsers. Applications rely on delivery context information access for performing content, presentation and service adaptation. The framework provides support for consumer and provider services. The dynamic device profile module supports serialization of delivery context information for external proxy adaptation services. A security and access module works in conjunction with ontology module for supporting access and integrity check of delivery context model. The ontology models delivery context hierarchy.Chapters 4, 5 and 6 describe the DCI framework, the provider interface and the dynamic device profile API in more detail.

The framework as such is not complete but is limited on certain fronts. The framework is designed to provide context access for web applications in particular. Towards this end, the delivery context model that is based on DOM is supported. A full adaptive framework has to support multiple applications that do not necessarily support a DOM type of information access. Also for multi-application support, the models have to be part of the middleware without any tight coupling to particular applications. Of particular concern when supporting multi-applications is modelling the behaviour when multiple applications listen for the same event. The properties can describe themselves through the ontology and it is upto the individual applications to decide how to interpret that information. For example, a volume controller on a stereo can be used to change the volume of a media player running within a browser. Similarly, the same volume controller can be interpreted by a user interface manager to select an item from a list by supporting list scrolling. Thus it is imperative that the framework should support multi-application disambiguation in some way because a user interaction with one application should not cause an unintended change in another application because both the applications were listening to changes from the same property.

The framework assumes a uni-device model. It assumes that applications access the device system and environment data. In addition to multi-application support that is needed, the emerging capabilities of devices would also warrant a multi-device support.

So, instead of a single model that is meant for local access, when multi-device scenario comes into play, a compositional approach for multi-models hosted by individual devices is needed. Compositional approaches essentially build a single composite model from multiple models. So applications get access to a single logical model that depicts the compositional capability of its current environment. This essentially constitutes dynamic smart spaces where services and capabilities of external devices and environment can be utilized. The current framework has no provisions for supporting such an advanced scenario.

The framework does not support the issue of working with heterogeneous access mechanisms but assumes that the different protocol stacks works in conjunction with the provider module. When the framework is extended to support smart space interaction, the use cases themselves changes to support more features than the current data-only access for consumers addressed in this thesis. There would be the need for applications to communicate with the environment and vice-versa. For example, through a browser interface, the user should be able to control the temperature of an air conditioner within a smart room. The current framework has no provision for consumers to communicate to providers.

4. Delivery Context: Interfaces

In this chapter, a more in depth description of W3C's Delivery Context: Interfaces (DCI) specification which is a consumer interface for static and dynamic properties is provided.

The Delivery Context: Interfaces specification is currently in candidate recommendation status in W3C specification process. Once the candidate recommendation period is passed, the specification is expected to go to proposed recommendation status and finally become a standard. The current editorship of DCI specification is shared by Nokia, France Telecom, IBM, and Canon.

The Delivery Context: Interfaces provides access methods for manipulating static and dynamic properties that may be exposed by the device, system and environment. The properties can be local or remote, i.e., residing on the network and seen as part of Delivery Context: Overview document. Dynamic properties change over a period of time. The rate of dynamic changes can vary depending on the type of properties. For dynamic properties, it is important to respond to changes as they occur. An example of such dynamic property is location property that can be GPS coordinates, triangulation data or other location technology that changes based on user movements. When a dynamic property value changes, notifications have to be sent to entities that would be affected. Consequently, a mechanism is needed to subscribe and unsubscribe to specific events.

DCI, as a client based mechanism can fit within a content adaptation framework where web content can be adapted based on the capabilities of the device. But, beyond content adaptation, DCI would be used by applications themselves to gather context data. This would provide application adaptation through simple access methods thereby reducing reliance on external services for providing the same information. The DOM model is the mechanism through which the document (HTML/xHTML) is exposed to application programs as an object model. Through the DOM model, the scripts view the document as a hierarchy of DOM nodes corresponding to each element within a well-formed XML document. Scripts use the DOM API to traverse and manipulate the document objects. DOM also supports an event system that involves an event propagation mechanism and handlers for listening and capturing events. The DCI also takes a similar approach to represent device properties in a hierarchical manner organized through a taxonomy that would be defined outside DCI scope. The approach was adopted due to the popularity and familiarity of DOM mechanism among application developers and also because it fits well with current browser support for DOM.The DCI is based upon the following fundamental normative concepts (taken from DCI specification [Waters, Sathish et al., 2006]):

1. All properties have a name, namespace and parent node all of which can be used to compare properties. This enables properties to be defined by multiple organizations without fear of conflict.
2. Property values may be dependent on data held remotely, however blocking for any significant length of time is unacceptable for DCI applications.
3. There is a flexible means to subscribe and unsubscribe to notifications of changes to properties. This gives content developer detailed control over notifications which are raised in a property specific manner.
4. Properties may be dynamically added, removed or changed at runtime. For instance, when a camera module is connected to a cell phone the capability to take photographs is added.
5. The DCI framework does not guarantee the property order.
6. The DCI framework does not specify a required DOM level.

4.1 DCI Property Hierarchy

The DCI represents properties in a hierarchical order. The properties are grouped into logical sets based on the type of property and their functionalities. Thus, the DCI representation forms a tree of properties. Each property is associated with a name, a namespace, a value, parent and sibling information. The DCI tree is rooted in DCIComponent property. The DCI property representation is shown in Figure 4.

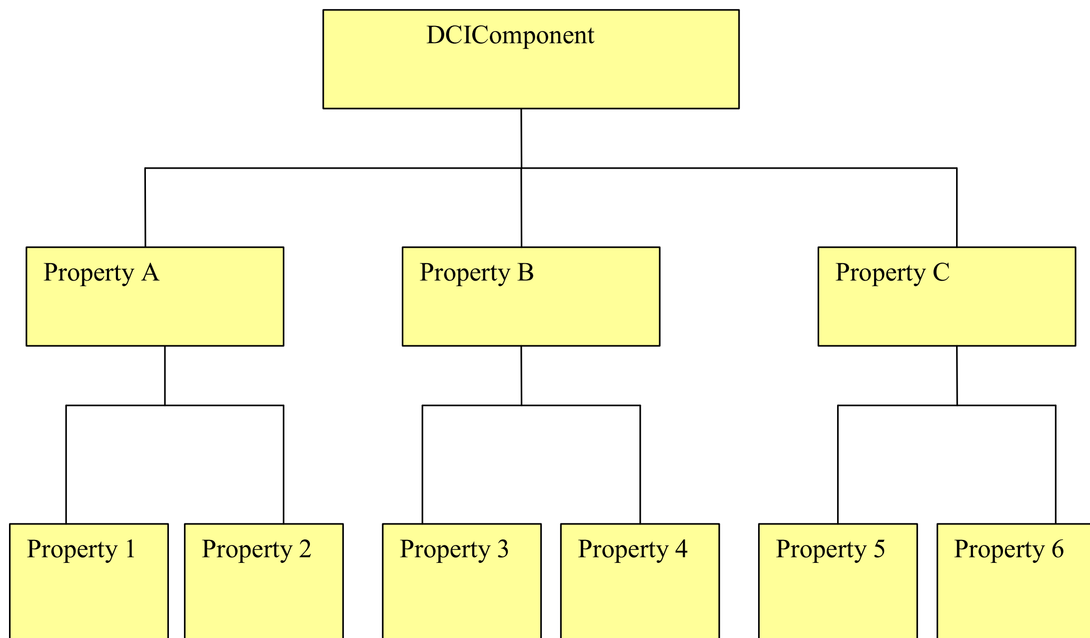

Figure 4: DCI property hierarchy representation (DCI ontology) [Waters, Sathish et al., 2006].

Figure 4 shows the DCI hierarchy representation with DCIComponent property as the root node. Property A, Property B and Property C show the first level hierarchy which would be logical components that houses different sets of properties. Examples of such a level would be Software, Hardware, Location, and User Data. Under each first level component or category, there are properties that form child nodes that can be grouped together. Example of such properties can be a GPS node and a triangulation node under Location category, while the Software category can have, for example, browser, operating system, and middleware data. It is to be noted that Figure 4 is shown for illustration only and a practical DCI representation can have many first level components as well as multiple levels in the hierarchy rather than the two level depths shown here. The ontology for DCI is expected to be standardized by standard bodies or consortiums and can well encompass existing ontologies such as OMA UAProf and other ongoing works such as Device Profile Extension [OMA-DPE] within OMA. There are also ongoing efforts within W3C (such as by Device Independence working group) to define suitable vocabularies for delivery context that can provide input to a DCI ontology. However, as of now, there is no standardized ontology for DCI.

The ontology defines the concepts, vocabulary and relations between the concepts that go into the DCI representation. The DCI tree deployed in the device would form an instance of this ontology in accordance with the framework described in Section 3.2. This is shown in Figure 5.

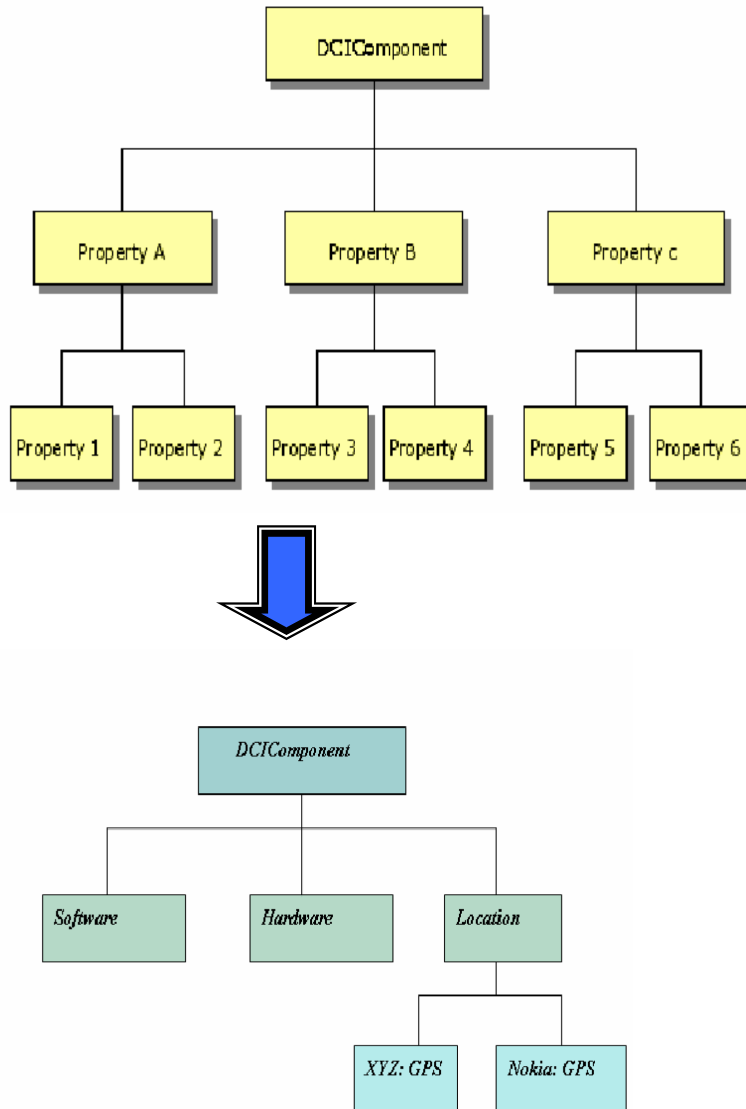

Figure 5: DCI ontology describes the concepts and relations while DCI forms an instance of this ontology.

4.2 DCI

The DCI is formally defined in terms of OMG IDL (Interface Definition Language [OMG IDL]). Each node in the DCI tree is a *DCIProperty* interface. The *DCIProperty* inherits from DOM *Node* [W3C DOM 2004] and DOM *EventTarget* [Pixley, 2000] interfaces. The *DCIProperty* provides additional methods that are applicable within the DCI context. All DOM entities are expected to be viewed within the DCI context when using the DCI interfaces unless specified explicitly in the specification. The

DCIComponent, which forms the root node of the tree, is an interface that derives from *DCIProperty* with an additional attribute *version* that lists the current version of DCI. The value for this attribute is now 1.0 in accordance with the latest DCI specification. Properties raise events to notify changes in property values, or the removal or addition of properties. *DCIProperty* supports methods for adding and removing event listeners. The *DCIProperty* supports the DOM event capture/bubbling model. This allows an event listener to be added at different levels of the property hierarchy. This is independent of the event propagation model in the host environment.

Properties are accessible to the calling application (consumer) once they have been initialized. The initialization model for each node is property specific and since there are different property types and initialization values, these have been deemed outside the scope of the DCI specification. Static properties can be directly initialized by the system as the values of these are not expected to change. There can be properties that are requested by the calling application that may not be present in the DCI. Depending on the DCI implementation, the application may provide URI for external services including supported protocol and initialization information. Depending on the available protocol stacks and support, the DCI implementation can establish remote sessions with those services in order to route them to the calling application. The advantage here is that applications can utilize the device protocol stacks for remote subscription in a seamless manner.

DCI specifies a generic interface that is property agnostic. Properties would have additional capabilities that may not be supported by DCI interfaces. In order to provide additional services to consumer applications, properties would have to expose additional interfaces to those provided by DCI. To support this, DCI provides an additional attribute *propertyType* which is of DOMString type that shows what type of property this node is. Properties being of a particular type can then derive from *DCIProperty* interface and provide additional methods. In such cases, the requirement is that such *propertyType* should be standardized or well known.

DCI properties expose different types of values to the consumer application. The values can be static or dynamic depending on property type. The values can have their own formats and data types. To address these, DCI specification leaves *value* attribute to be of *any* type. This means they can take any type of value and any format. For consumer applications to understand the *value* attribute, DCI provides an additional attribute called *valueType* that denotes the type of value the property exposes. Applications use the *valueType* attribute to understand what type of *value* the property provides.

Properties have metadata that provides additional information about the property. Typical metadata can include version number of property, time and date of addition to DCI tree, manufacturer information, precision or granularity, update frequency in case

of dynamic values and others that can be property specific. DCI does not specify a format for metadata representation. Like *value* attribute, it provides a *DCIMetaDataInterface* that has been type defined to be of *any* in order to cater for different types of metadata that are possible. It also defines an additional attribute *DCIMetaDataInterfaceType* that is of type DOMString denoting the type of metadata interface the particular property supports.

DCI does not provide any restrictions on the name or type of property that can be present in the tree. In order to distinguish between multiple properties of the same type, DCI supports the use of namespaces that can distinguish one property from another. For example, a device may be associated with two GPS nodes, one from Nokia Corporation and another from IBM Corporation. These two form child nodes of Location property. Consumer application can look at namespace attribute that are distinct for each vendor and thus select the one that is most suitable. There can even be multiple properties with the same namespace that can then be distinguished by comparing their metadata through the *DCIMetaDataInterface.* Since DCI specification does not place any restriction on where nodes can be placed (even though they are expected to conform to an ontology that is beyond DCI specification scope), nodes with same namespaces can be present at multiple levels in the hierarchy as shown in Figure 6. Here, two GPS nodes with A and B namespaces form child nodes under a parent GPS node with namespace C. There is also a GPS node with namespace A under B node. In order to distinguish between A:GPS under C:GPS node and A:GPS under B:GPS, the consumer application can check the *parent* attribute to understand the hierarchy and/or check metadata of the properties. Figure 6 illustrates the use of namespaces for distinguishing properties.

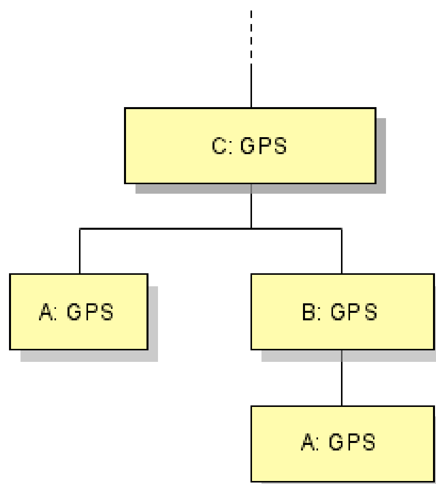

Figure 6: Same type properties can be distinguished based on namespace attributes. Between same namespace nodes, they can be distinguished based on parent nodes and/or metadata property [Waters, Sathish et al., 2006].

DCIProperty interface provides additional methods to check for properties, search for properties that are not provided by DOM interfaces. For searching, DCI provides *searchProperty* method that returns a *NodeList,* which is a collection of nodes that satisfies the search criterion. Applications can also pass an application defined property filter called *DCIPropertyFilter* which is an interface that takes in one property at a time, checks if it fits the client supplied search criterion and returns a Boolean that determines whether that property can be included in the *NodeList* that would be returned after a *searchProperty* call. The *hasProperty* method returns a Boolean value that shows whether a property is present in the DCI tree. It can be used as a quick call to search for property instance within the DCI tree.

DCI supports the DOM event model for notifications of property changes to consumer applications. Event flow is the process through which an event originates from the root node (*DCIComponent*), travels through the DCI tree to a *DCIProperty* (capture phase) that is the event target and then traverses back to the root node from the *DCIProperty* event target (bubbling phase). Each event has an *EventTarget* toward which the event is directed by the DOM implementation. This *EventTarget* is specified in the *Event*'s target attribute. When the event reaches the target, any event listeners registered on the *EventTarget* are triggered. Although all *EventListeners* on the *EventTarget* are guaranteed to be triggered by any event which is received by that EventTarget, no specification is made as to the order in which they will receive the event with regards to the other *EventListeners* on the *EventTarget*. If neither event capture nor event bubbling is in use for that particular event, the event flow process will complete after all listeners have been triggered. More information on DOM event model can be found in DOM Level 2 Events [Pixel, 2000]. The DOM Event specification defines certain types of events that are applicable to a DOM tree. The DCI specification defines an additional event called *dci-prop-change* (denoting change in property value). In addition, DCI specification also recommends the use of following DOM events (defined in DOM event specification): *DOMSubtreeModified* (denoting change in DCI subtree topology), *DOMNodeInserted* (denoting insertion of a new node) and *DOMNodeRemoved* (denoting removal of a node). In addition, DCI also specifies certain exceptions that are applicable within the DCI context. The DCI IDL is listed in Appendix A.

The DCI model as a standard has several advantages and drawbacks. They are listed below.

Advantages

- DCI provides a standardized model for context data access.
- DCI supports dynamic and static data.
- DCI supports the full DOM asynchronous event model.

- DCI models data based on a hierarchical representation thereby maintaining property relations, efficient search and navigation mechanisms.
- DCI hierarchy representation allows using web addressing mechanisms such as XPath expressions.
- DCI supports extensible metadata structures for properties.
- DCI supports extending the standard set of interfaces with property specific interfaces.
- DCI model can be serialized to support external adaptation services.
- DCI provides namespace support allowing multiple properties of the same type to be added.
- DCI supports application specific search criterion allowing for powerful search mechanisms.

Disadvantages

- DCI is essentially a web oriented context access model and hence addresses a specific domain.
- DCI does not normatively describe a value and metadata structure for properties leaving it to external interpretation.
- There is no standardized ontology for DCI representation.
- There is no security model defined for DCI specification.
- The specification does not normatively describe how to access the DCI model leaving it to browser specific implementations.
- DCI does not specify how multi-device and multi-application scenarios are supported.

More information about DCI can be accessed from the Delivery Context: Interfaces Accessing Static and Dynamic Properties specification [Waters, Sathish et al., 2006] available at W3C.

5. Delivery Context Provider Interface

This chapter describes the DCI provider API that has been developed internally within Nokia. The DCI API is intended to provide a DOM-like view of a context repository to calling applications. For a context platform to work, having a consumer API is not enough. There must also be a mechanism for context data providers to feed data to the context platform. The DCI provider API works orthogonally to the DCI API in that it provides an access route for context data providers to the context platform.

Figure 7 shows the data flow within the client context platform. The data flow in Figure 7 conforms to those described in Chapter 3.

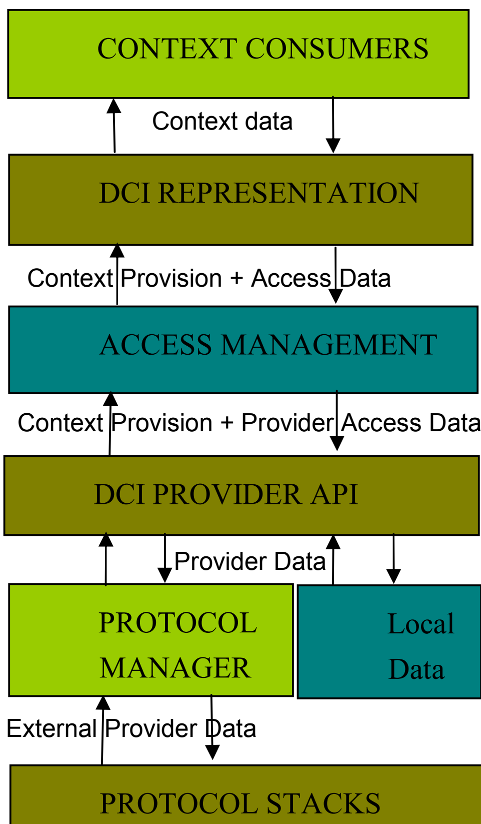

Figure 7: Data flow in delivery context framework.

For the consumer to access data, the DCI model has to hold data. For DCI to hold data there must be data providers that have established session with DCI feeding data to it whenever relevant, in case the data is dynamic. The DCI platform can initiate service discovery or service providers can initiate session with DCI. The DCI can initiate session with a context provider if the platform sees a need for new data that is not

already available within the model. Also, depending on the implementation, DCI can be configured to perform run time discovery of data providers even though this would require platform dependent implementations and modification to platform specific directory services that provide descriptions of dynamic data sources. In addition, consumer applications can also get DCI to initiate sessions by providing necessary information of remote services.

Local applications can also initiate sessions to DCI if they are "DCI enabled". This would mean discovering the DCI platform (either through direct DCI provider API call by using a DCI library or through directory services), providing description of their services and getting access to the DCI tree. Data providers can also initiate sessions under other circumstances. One would be when the network provider initiates a PUSH-type service of context data through its trusted provision services. Another case can occur during a redirect by a service provider. Here, a network based service provider or proxy can perform redirection to another service from an already existing (session established) service, if it finds that the service no longer satisfies customer requirement. The new service (due to redirection) then has to establish a new session with the client (through instruction from proxy) or the proxy itself can direct the client running DCI platform to initiate session with the new entity. New negotiation protocol might be needed or existing protocols for session establishment such as SIP can be extended with new payloads so that end entities can understand the dynamics of the session.

It can be assumed that static data for DCI such as screen size, colour depth, CPU, memory, keypad, browser software, or version number (semi-static) can be initialized by the system middleware itself. Also dynamic data that comes pre-configured with the system such as screen orientation support, accelerometer and sensors can be directly hooked up to the DCI tree by the system. For such providers, the DCI platform does not need to perform service discovery and session establishment. These are statically configured. There can also be run time discovery of services that DCI can perform. The DCI specification leaves discovery out of scope and it is up to the implementation and the particular platform it is running on to make such run time discovery of services. Standard applications may have to be modified to support DCI query for dynamic data or they may have to support additional subscription protocols such as Web Services in order to enable discovery and subscription. The Symbian platform supports getting application data through use of UID values. The application data can then be used for subscription purposes. This poses additional requirements that the applications have to publish data that would seem relevant to DCI such as dynamic data, data and service description, subscription handling etc.

There are many applications that are capable of providing relevant dynamic data. For example, a calendar application can provide data such as user's current activity, upcoming events, date and time etc. As such, the calendar application can have multiple

entries in the DCI tree under a parent node called "Calendar". Because of the diverse nature of applications and the diverse data that each application can provide, it is not a good design idea for DCI to initiate sessions and query data. An alternative would be to make applications "DCI aware" so that they understand the DCI Provider API and establish session with the DCI once they are installed. Applications can check for DCI support by possibly checking a DCI UID (Universal Identifier) with the platform, usage of a registry service or through knowledge of DCI support with a particular version of a platform. The applications can then get a DCI node pointer for each dynamic data they can provide by providing the DCI platform with a semantic description of the services offered and their fit to DCI ontology if one exists. In cases where DCI cannot determine the semantics of data, user confirmation can be obtained to offer a new node in the DCI tree.

Applications (data providers) should not start sending dynamic data as soon as they have established session with the DCI tree. The providers should only start updating their values once a consumer application attaches an event handler at the DCI property node. Once an event handler has been attached, the DCI platform through the provider API sends a START request to the application, after which the application uses the node pointer to update data. The START request can also have as optional parameters, the refresh rate for data, the granularity of data etc if supported by the data provider. The default behaviour would be to support the same dynamic rate of the data provider. Once the event handler has been removed by the consumer, the DCI platform can send a STOP request to the data provider after which the provider would stop sending data. There is no standardized protocol for DCI communication with data providers yet and this is still under development at the time of writing of this thesis.

DCI can initiate sessions with remote services if they are either requested by the consumer application, pre-configured or redirected by a proxy or discovery service. A majority of context providers reside on the network side and due to the lack of a standardized recommendation for context service provisioning protocol; it can be assumed that the client needs to support a set of protocol stacks. These include support for protocols such as Session Initiation Protocol (SIP) and SIP events, Web Services stack including WSDL, Simple Object Access Protocol (SOAP) and transport protocols. Consumer applications such as web applications can utilize these stacks and request DCI to subscribe to remote data services using these mechanisms. If a node for the desired remote service is not present, the consumer node can create a new node (subject to DCI permissions) and add information about the remote service through the metadata interface for that node. The trigger for establishing session with the remote service can be the addition of an event handler by the consumer for that node. The DCI platform can perform clean-up of the node once the event handler is removed by the application

or if the application closes prompting automatic clean-up of all resources used by the consumer.

A framework for DCI based context provision was introduced in Section 3.2 (Figure 2). As described in Figure 2, any context provider that wants to provide data to the DCI tree will contact the DCI Session Manager (DSM). The DSM can be discovered through service discovery mechanisms such as SIP or other procedure calls. The DSM provides the translation and session management between the DCI provider part and the protocol stacks for provisioning. A data provider contacts the DSM which then queries the DCI provider module for a session ID. The session ID is generated if the data provider is deemed secure by the access control module. The access control module deals with access control policies and is used by both the consumer (DCI) API and the provider API. The policies dictate which consumers and providers have access to the DCI tree and modification rights. Once a session ID is generated, it is up to the DSM module to manage the session with the context provider. The context data provider will use the unique session ID that was generated for all subsequent communication with the DSM. The framework shown in Figure 3 also behaves the same way, the difference being that the ontology manager controls access and placement policies of nodes in accordance with a prescribed policy.

The DCI provider API provides a set of methods for the following:

- search the location of a property within the DCI tree,
- check for properties,
- add a new property,
- remove an existing property,
- set a property value,
- get and set metadata for a property, and
- set namespace prefixes for XPath [Clark and DeRose, 1999] usage.

The provider API supports usage of XPath expressions for addressing nodes in the DCI tree. A major requirement for using XPath expressions is that the expressions should be resolvable within the DCI context. The API provides support for the initial setting of a prefix for a namespace URI so that the prefix can be used with XPath expressions that the provider uses. This eliminates the need for a namespace resolution mechanism. The namespace prefix is only valid for the particular provider and is identified based on the unique identifier generated during session establishment. Prefixes have to be set before calling any method that uses namespace prefixes. The DCI API presented in this thesis is not standardized. Even though it would be good to have a standardized API, a provider API can also be proprietary. This is because the provider interface is used directly by local applications installed on the client device. For remote data providers, a protocol stack can be used that addresses the local provider API functionality. This thesis does not provide details of requirements for such a

protocol. The protocol should be standardized and where possible, existing standards should be re-used. The protocol should normatively mandate session establishment procedures and control parameters for data synchronization. The protocol should also be extensible enough to support bi-directional communication between consumers and providers when negotiating for specific services as future support.

The DCI Provider API is presented in detail in Appendix B. The API reuses standard DOM data types wherever feasible. This is to maintain data type synergy between DCI internal representations and those supplied by providers.

6. Dynamic Device profile (DDp)

Device profiles are used by adaptation mechanisms (residing on a proxy, an adaptation server or the content server itself) for adapting content in accordance with the device capabilities and its environment. The most popular profiling mechanism is the User Agent Profile [UAProf] developed by OMA. The drawback of UAProf is that the profile is static and does not necessarily reflect the current profile of the client. Moreover, there are limited extension mechanisms available for creating ontology reflecting new capabilities. This section describes the ongoing efforts in creating a dynamic profile that can be generated by web applications utilizing the DCI architecture. The mechanism highlights the use of DCI as a context provisioning mechanism as well as a source for dynamic profiles that aid with content adaptation.

As seen from Figure 2 and 3, the dynamic device profile (DDp) module supports an API at the client side for creating a profile that can be sent to the server at any point during a session. The DDp module uses the DCI tree for data that it uses for serialization. The DDp module through the DCI handler can access the property tree and listen to property updates or other changes that might be deemed important by a service provider. The application (web page) can embed DDp specific content that can be processed by the browser. The application through markup can specify what properties it is interested in, what thresholds for values of properties it should watch for, and what are the changes in topology that has to be communicated back to server. The whole purpose of having DDp is to aid server side content adaptation. The server can also communicate with DDp directly to provide DDp related markup but this has not been addressed in this work. The DDp should support serializing parts of the DCI tree (certain sections based on categories or properties) or even the whole DCI tree if required. The platform (browser platform, middleware or framework that the user agent is part of) can also utilize the DDp for serialization if changes to the topology are noticed. An example would be in a multimodal framework where the Interaction Manager (IM) notices changes in modalities supported that can be conveyed back to the origin server for new content that is suitable for the new topology.

DDp serializes part or whole of the DCI tree to the server for adaptation. The serialization format can be XML, standardized formats such as UAProf or some proprietary format. The DDp module should support the use of any type of serialization format thereby independent of any particular format. This means that server side adaptation mechanisms can rely on proprietary formats that they can understand. This would also absolve the need to have a standardized format for serialization even though having standard formats and complying with standard formats would be desirable. Having standard formats also helps intermediate entities such as proxies to add further

information if desired that may be of use to the adaptation mechanism. A DDp module should support services in serializing DCI in any format that may be desired.

Taking the above design considerations into account, an API has been developed to work with DCI that can support server side adaptation mechanisms. The key features of the API are:

- The application/server can define own protocol for serialization so they do not have to wait for some dynamic profile to be standardized.
- The client can have multiple serializers available and the application can choose which serializer to use.
- Application authors can determine when to send the profile to server based on scripting control.
- The API supports event mechanism for dynamic notification.
- The API conforms to standard DOM mechanism wherever possible.
- There is a filtering mechanism for getting only nodes that are needed.

The above features are achieved through a series of interfaces. The serializer interface provides a serialize method that takes in a set of *DCIProperty* nodes and provides a *DOMString* output. This requires that the active serializer is set through a pervious method call. The application provides a filter interface that determines what nodes within the DCI tree need to be added to a list to be serialized. Thus the logic for filtering properties is handled by the application. The response handler interface is also an application provided handler. This interface is responsible for handling a response from the server once a profile has been sent. There can be default implementation behaviour for handling responses if the interface is not supported by the application. A serialization list interface provides methods for creating a node list for serialization. This interface extends the *DCIPropertyList* interface and adds additional methods for appending and removing property nodes from the list. The main interface is the dynamic device profile (DDp) interface that provides support for adding, removing, activating serializers as well as methods for setting response handlers and submitting the profile to the server using a DOMString based method identifier. Additional exceptions have been defined related to removal of serializers.

The DDp requires that at least a default serializer is installed as part of the browser platform in order to generate a dynamic profile. This by itself can introduce heavy load on browser platform and on the mobile middleware. DDp does not mandate any particular serialization vocabulary but one suitable candidate would be OMA's UAProf [UAProf]. In its current form, since DCI does not have a standardized ontology, providing a serializer to work with DCI would mean performing searches for properties of interest and attaching handlers for events at their respective representations in the DCI tree. Having a specific ontology would mean using direct addressing such as XPath expressions to create a list of properties to serialize. This would avoid the step of having

to search for properties resulting in some optimization. Utilizing the DDp mechanism would need changes to the way current adaptation services work and the way web applications are written. Adaptation services need to cater for in-session dynamic data. Web applications can embed DDp specific content and application authors need to be aware of DDp specific mechanisms and DCI vocabulary. Even though DDp specific content can be embedded within web pages, the same can also be transferred through an HTTP extension. The exact method for DDp conveyance and processing is not clear at this point and is outside scope of this thesis.

The full Dynamic Device Profile Interface is presented in Appendix C.The dynamic device profile API set has been developed internally within Nokia and is intended to be submitted as a proposal to OMA or W3C within the dynamic profile activity.

7. Security and Trust

“Context” represents a set of data that characterize a particular situation of a user. Some of this data could be deemed private by the user and access to such data should be provided on a case-by-case basis. The user should be able to give access rights to trusted applications where usage of such data is not intended to be used in a malicious manner. As such, issues of privacy and trust have to be taken into account when designing any context access mechanism. Services that provide context data to the application have to be trustworthy. This is needed so that information that is fed should is correct and not intended to mislead or lead to undesired consequences.

As the world adapts to new e-business models, the traditional ways of doing business are changing. In the traditional world, a purchasing manager would get request for commodities, after which he/she would go through a number of their regular contacts and suppliers. Their requirement would be addressed by sales people wooing company contacts with deals, gifts and other packages supplemented by trips to manufacturing premises to “inspect” products at hand. Depending on the nature of the business and the lucrative interests of the buyer and the seller, long term relationships can be established guaranteeing business and establishing a level of trust. Companies, through such deals ensure that their long term goals – top quality products at competitive prices are delivered on time, every time. In the world of e-business and e-services, the issue of a known partner is increasingly rare. Here, companies or individuals have access to a vast number of vendors and service providers through the e-market. As such, it has been noted that companies and individuals are not ready to trust such third parties with whom no previous contacts or trade deals have been had. A significant percentage of e-transactions still occur through previously established relations and practices. To address these, new mechanisms that deal with trust and privacy issues are slowly getting established. Some of these mechanisms are:

- Verification and authorization: Verification and authorization deal with confirmation of the authenticity of the service that is being dealt with. Verification and authorization, on a global scale can now happen through trusted third-party certifications such as Verisign, Identrus and services such as CommerceNet through use of digital signatures and certification services. Verification and trust can also be established through reviews of peer users such as Friend Of A Friend [FOAF] services, comments and reviews etc.
- Quality: This is more or less related to verification and authorization in that the information or service that you have requested for is the one that would be provided. In ICT, quality of service has less relevance compared to issue of verification and authorization.

- Ratings and community functions: In addition or complementary to a trusted-third party formal certification, community ratings are one way forward to establish trusted relations. We act on recommendations by people we trust, share knowledge of their experiences, and make an educated decision about trusting a particular service or commodity. In the world of ICT, individuals post their opinions and experiences through such electronic notes posted either at the service site, bulletin or social boards as well as with independent media. Ratings by such community functions are a social measure of how trustworthy the particular service is.

When establishing session with a service provider, it is imperative that mechanisms of authentication and verification are employed. One way of providing automated verification process would be to mandate certification data through metadata passed on about services to the context platform. A verification service can then authenticate the provisioning service. Brand is also a socially accepted norm of implicit authentication. For brand-based authentication, it would require confirmation from the user, and a self-learning platform can store the behaviour for future transactions, thereby eliminating the user role. For verification and trust based on ratings and community functions, the user role is highly significant and confirmation to accept the service has to be explicitly obtained from the user.

Apart from trust, the security of the system needs to be taken into account. Since the DCI specification does not mandate any particular security aspects, it is up to the implementation to ensure that malicious consumers or providers do not perform undesirable modifications to the context model (DCI tree). DCI suggests (not mandates) that properties follow a logical order, i.e., conform to an ontology that would get standardized. Ontology describes the concepts and relations between the concepts that exist within a particular domain. This would make it easier for applications to find properties, authors to logically figure out property positions and apply mechanisms such as XPath expressions for property addressing and expect a predicable model for DCI tree. Having ontology also helps with maintaining the integrity of DCI tree (see Section 3.2.1) and also deal with security issues. While integrity constraints and checks have been explained in Chapter 3, this chapter looks at one possible model for providing security policy for DCI. It has to be noted that the security policy is suggestive and has not been tested yet in real world.

The security policy deals with mapping each provider and consumer to a particular class where specific rights have been assigned for each type of class. The solution aims to minimize the risk of supplying invalid or incorrect information to the calling application or of creating bogus properties within the framework by malicious programs. The policy involves utilizing the metadata interface of DCIProperty interface that gives information about the property and (vendor specific) additional data that can

be utilized by the calling application or underlying implementation. For example, metadata can contain information about the version of the property, the time of addition, property specific data such as location of the property, megapixel data for a camera, grammar support for a speech recognizer and so on.

The security component (see Figure 3), exposes security classes which define security rights for components. The term “component” refers to a software program that can be a consumer or provider, an application, or a physical component (such as sensors, audio modules). The components need to register to one of the classes according to a class identifier, which is assigned to the components by an operating system or a middleware component. For external service providers, the class identifier should be obtained either through a third party assignment service or via direct negotiation with device middleware component through metadata verifications. The exact formats and process for class identifier procurement is out of scope of this thesis. The class identifier is generated in such a way that one part of the identifier will determine which class is in question, and the other part of the identifier uniquely identifies the component or application within said class. The class identifiers are used to determine, which class the programs can be registered to. The components will register to one of these classes based on their priority (class identifiers) that has been assigned to them. The security module will look at the particular class to which a component belongs and then grants the appropriate rights.

In simple terms, security is managed through a set of classes, with each class having a set of associated access and modification rights to the DCI tree. Components, based on their requirements and rights, are mapped to one of these classes where they inherit the rights that have been assigned to the classes. Sometimes, it may be required that the class right itself is modified based on certain situational requirements. This means that even the class rights have to be managed. To deal with this, each class is given a set of default rights. In addition to the default rights, each class is also associated with a modifiable schema that can override the default behaviour. The schema will be maintained by the security component and each interaction request will be validated against the schema before execution. The schema can be edited by the user, operating system or underlying implementation.

To provide security features, we also suggest introducing new tags for the metadata interface within DCIProperty interface. One tag is the “owner” of the property, and the other is a “visibility” tag. The owner is identified through the owner identifier in the metadata interface. The owner entry is added by the DCI implementation platform and the entry corresponds to the class identifier assigned to the component. This means that the owner of that node would be the property that requested the new node. Depending on the class that the owner (property) belongs to, the class rights would be assigned to

the new node. The high priority classes can have the authority to read and delete any property that is deemed beyond current context scope.

The visibility tag can be set in the metadata interface by the owner of a property or by a higher priority class. The visibility tag defines whether the property or component entry can be seen by other components. By setting visibility tag to “OFF”, the property in question will not be visible to other components. This depends on the type of DCI implementation. An implementation can allow partial visibility of DCI tree depending on visibility rights of classes. It is also possible to set visibility for particular components based on class identifiers if the class identifiers are known. This can be set in the schema for that particular class to which a component might belong or in a master schema that is derived by all class schemas. This is a useful feature to block visibility of particular components to other malicious components. In future, this may also enable service providers to request blocking of their service usage by certain sets of components within the system. The visibility would be hierarchical in nature so that setting a visibility at a particular node would also apply to all children of that node. However, the setting would not apply to siblings of that node. Child nodes can override these by explicitly requesting and setting privileges within the schema for their associated class (based on their class identifier).

As an example, we describe here four classes. It has to be noted that there may be other classes as well in a typical implementation. Moreover, new classes can be added whenever required.

- Class A

Components that are registered to Class A have the ultimate control in the system and are so called “priority class components”. The components of Class A can add, delete, modify or replace properties and parameters of properties anywhere in the DCI tree. Visibility tags do not apply to Class A components. The properties cannot set individual class identifiers if those class identifiers belong to Class A. The security module can be implemented so, that only the operating system can add Class A components, whereby no component can register by itself for this class. An example of a Class A component is a System component or an Interaction Manager for a multimodal system. Only a Class A component can delete a property created by another Class A component.

- Class B

Components that are registered to Class B can add new properties and are allowed to add subproperties as children to the newly added properties. Class B components can modify, delete, add and replace only those properties that were created by that particular component and those Class C type properties whose security settings are set as default. No other properties, such as Class B entries that are not owned by that particular component, can be modified. All registered components can access the newly added

properties and register event handlers for property updates. Class B component can add to any properties within the hierarchy tree within the constraints applied as dictated by the hierarchy (e.g. a GPS property cannot be added under a video property). A Class B property can also set the visibility tag for any property created by any Class B component for class C and class D categories (all Class B settings remain the same) but not for Class B unless the owner is setting the visibility.

- Class C

Components that are registered to Class C can create DCI nodes but they can modify only those that they have created. For such properties, Class C component can set visibility (for their own nodes) for Class B, Class C and Class D categories. If a visibility has been set to OFF (other than default) for Class B category, a class B type property cannot add a new entry under class C type property. If the visibility is ON, then a Class B can add a child to Class C property but after that, the visibility of that Class C property cannot be modified by any property other than a Class A property of until the class B property that was added got removed. Class C components can register for property updates anywhere within the DCI tree.

- Class D

Class D category is applied with the highest security settings. The components registered under this category have the least priority and access rights. Class D components get only a partial view of the DCI tree, which means that such components can only read data from DCI nodes for which the visibility is ON. They cannot add, delete, modify or replace any entry within the DCI tree. Class D can be used for blocking user specific details such as personal codes, preferences etc. from malicious applications. The extent of blocking can be governed by the operating system as well as customised by advanced users.

When a property belonging to a particular class try to access the DCI tree, the schema for that class is consulted and a view corresponding to that class is created. In this view, all the properties that have visibility are added and all those whose visibility is OFF are not added. Thus, there will be the same amount of views as there are classes, a view per class. Depending on the class identifier, further refinement of visibility is possible where a secondary schema or mask is applied after applying the class schema to the DCI tree. Hence, there can be a DCI tree which would be a master repository along with subsets of that tree corresponding to each class.

The default behaviour of the security class is that when a component creates a DCI property node into the DCI tree, the security settings that are default for that component class and visibility ON for higher class comes into effect. The owner can turn the visibility off for classes B, C and D, if it is desired or the owner can turn off visibility for specific class identifiers. It should be noted that if there exists a child property that

belongs to a higher class than the parent property, the parent property owner cannot turn the visibility of that property (parent property) OFF.

The security solution suggested here is only one of several mechanisms that can be employed. The DCI specific security implementation can be coupled with standard web mechanisms for security such as XML signatures and XML encryption and thus a single solution can be used. In the proposed solution, the exact format of security profile, the schema, inheritance controls etc are not mentioned. These are outside the scope of this thesis. A full security framework has to support other mechanisms such as verification, authentication, encryption and secure communication. The solution suggested here only supports authorization including access control while leaving other parts of security framework outside scope. It is envisaged that standard and proven methodologies would be adopted for a fool-proof framework with relation to security. The thesis does not address privacy policies. Although the security framework in Chapter 3 provides a module for security and access policies, it is also important that user privacy is addressed properly and that private data is not accessible for all consumers. Users need a way to provide information to the framework for controlling data access. Users should also need a mechanism for dynamic control of privacy policies through an easy and intuitive manner. Such a mechanism need to be developed. When moving from a uni-device to multi-device scenario as well as support domain specific smart spaces, the policies themselves have to be expanded. Smart spaces would have certain policies that are domain specific and these need to work together with user privacy policies. The methods for integrating and supporting domain specific and user policies are not clear at this point. This would be part of our future research into security and privacy for smart applications and smart environments. Efforts have also started within standards bodies such as W3C to look into access policies and privacy concerns especially in relation to context data. Once the solution is more complete and mature, it is intended that we (standards body representatives at Nokia) propose this as a candidate solution for contextual data privacy policies.

8. DCI Implementation

This chapter describes the DCI implementation carried out by me. The implementation process was carried out as a part of providing a reference implementation for moving forward DCI specification from candidate recommendation status to proposed recommendation. The DCI specification has gone through two last-call status within W3C where the specification underwent significant changes. Overall, three implementations for DCI specification were made and findings from implementation analysis were incorporated in the specification.

Two approaches were followed for implementing DCI: 1) provide a wrapper implementation with an existing DOM package to provide DCI functionality and 2) to develop our own DOM-based DCI implementation from the scratch. The second approach was needed as we found that a wrapper-based implementation was not feasible since it violated certain aspects of DOM specification.

One of the main design criteria for DCI specification was to leverage as much existing work on DOM as possible. There are several DOM implementations and libraries available. Hence utilizing them for rapid deployment made sense. As such, the first implementation for DCI was done as a wrapper over an existing DOM package. The aim was to provide a DCI extension to Mozilla Firefox browser to run DCI-enabled web pages that can adapt based on data from DCI tree. The approach was to implement a stand-alone DCI module with a test application and later incorporate into Firefox browser through its extension mechanisms. The platform was Linux RedHat 9 with a C++ implementation. The wrapper based approach revolved around providing DCI functionality over Xerces DOM parser [Xerces DOM] which is a popular open source implementation by Apache foundation. However, we were unable to provide dynamic value changes to the *nodeValue* attribute as the DOM Node specification allows a node value to be set only for attribute, CDATA and comment type nodes. The *nodeValue* is defined to be NULL for other node types and the value could not be set with the Xerces C++ DOM bindings. At this time, the DCI specification also directly inherited the DOM *nodeValue* attribute. The *nodeValue* attribute was treated as the “value” attribute for static and dynamic values within the specification. This violation of DOM specification was conveyed to the working group. Based on this input, a new attribute *value* that is type defined to be of “any” type was introduced to *DCIProperty* interface.

The second approach centered on a more direct integration using Firefox browser mechanisms for providing extensions. Since the extension is not a user interface extension, the traditional Mozilla XUL (XML User Interface Language) and JavaScript extensions do not apply. Native code was used that integrates with Mozilla code through the XPCOM (Cross-Platform Component Object Model) [XPCOM] interfaces. Mozilla is built using components. A component implements functionalities that have

been described by interfaces. A component can implement multiple interfaces or multiple components can implement a single interface. The job of a component is to implement the interfaces that it supports correctly. Thus, application developers can have components that implement the same interface for Windows, Mac OS, Linux and other operating systems that have different implementations but conform to the same interface. Any code that relies on those interfaces functions the same, even though the underlying implementation may be different. XPCOM (as well as Microsoft Component Object Model) components can be checked (queried) to see if they implement a particular interface and if so, the methods of the implemented interface can be called. The general way to access an interface in XPCOM involves the following steps:

1. get the particular component,
2. create an instance of the component,
3. query the created instance whether it implements the interface that is needed, and
4. if the interfaces are implemented, call the required function.

The interface language that Mozilla uses to describe interfaces is XPIDL (Cross Platform Interface Definition Language) which is similar to OMG IDL. Once the interfaces are available in XPIDL, C/C++ headers can be directly generated from the interface files through use of Mozilla's own IDL compiler, xpidl. Interfaces have additional attributes such as a UUID that uniquely identifies the interface. Interfaces support a *script* attribute, which means these interfaces will be accessible from the JavaScript code. A scriptable interface is only allowed to use data types that are valid within the JavaScript runtime.

In my implementation, a class structure was generated using Mozilla XPIDL generator from the DCI interface definition language (IDL). The *DCIProperty* interface inherited from Mozilla DOM Node interface. The *DCIProperty* interface has additional methods for searching and checking for properties along with additional attributes. The *DCIComponent* is also of *DCIProperty* type with an additional "version" attribute with a read only value of "1.0" for the current implementation. The *DCIComponent* (as shown in Figure 4) forms the root of the DCI tree. The DOM Event interface has been implemented as a separate interface rather than making DCIProperty inherit from DOM Event. The reason for implementing these as separate interfaces is due to the fact that Mozilla XPIDL allows only single inheritance between interfaces. The implementation conforms to XPCOM mode of functionality. The DCI implementation has been done as a separate component where the interfaces can be queried to get handles to access the methods. There is a single component that implements DCIProperty interface, DOM Event interface and DCIComponent interface. Each interface can be accessed separately by querying the component and getting handles to the interface. Some private methods are also defined that deal with initialization of the DCI tree. When the DCI component

is accessed through a JavaScript call, the initialization methods that build up the DCI tree are called.

The current implementation does not provide the feature to add an application defined property filter when searching for properties that is defined in the DCI specification. This would be provided in a later implementation. A mock taxonomy for a first level hierarchy of property vocabulary including Software, Hardware and DeviceContext was created. Providing such a vocabulary was sensible because a DCI vendor would include a logical grouping of properties that are available with the platform. The Software and Hardware nodes were initialized with some static child nodes that were added under each of the first-level categories. Software node has properties Browser (where name is the only value with version number as metadata), and Operating System (with name as value and version as metadata). Hardware node has child nodes Screen Size, Colour Depth, and Keypad Type. The DeviceContext node has child nodes Location, Presence, Date and Time. The Location node has child node GPS. For a DCI implementation that was shown on a desktop machine, we simulated the GPS data since using an actual GPS data was not feasible. The simulated GPS coordinates were fed through a microserver based data feed. A demonstration with an actual GPS device was also made, details of which are provided in Chapter 10. The presence data was simulated while the Date and Time nodes were fed directly through system calls. The DCI implementation for Firefox was packaged as an extension that could be used with Firefox 1.5 and above.The Firefox extension is nothing more than a collection of files and folders that have been compressed using a “Zip” utility. The extension has an “.xpi” extension that is a .zip file that has been renamed. In addition, at the root directory, an install manifest file needs to be added.

The first implementation, integrated with Mozilla Firefox browser was for a desktop demonstration of simulated location data. In addition, two more implementations targeted at different devices were made. The second implementation and porting of DCI code was made for the Nokia 770 Linux Tablet. The DCI was integrated with the MANAOS [Manaos Browser, 2006] browser that is a Mozilla implementation for the Linux tablet with a customized User Interface. The snapshot of the browser running a DCI application (described in Chapter 9) is shown in Figure 8. Here, the GPS coordinates were simulated to platform through use of a microserver. The mashup application takes the GPS coordinates and plots polylines along the coordinates, thereby simulating user movement.

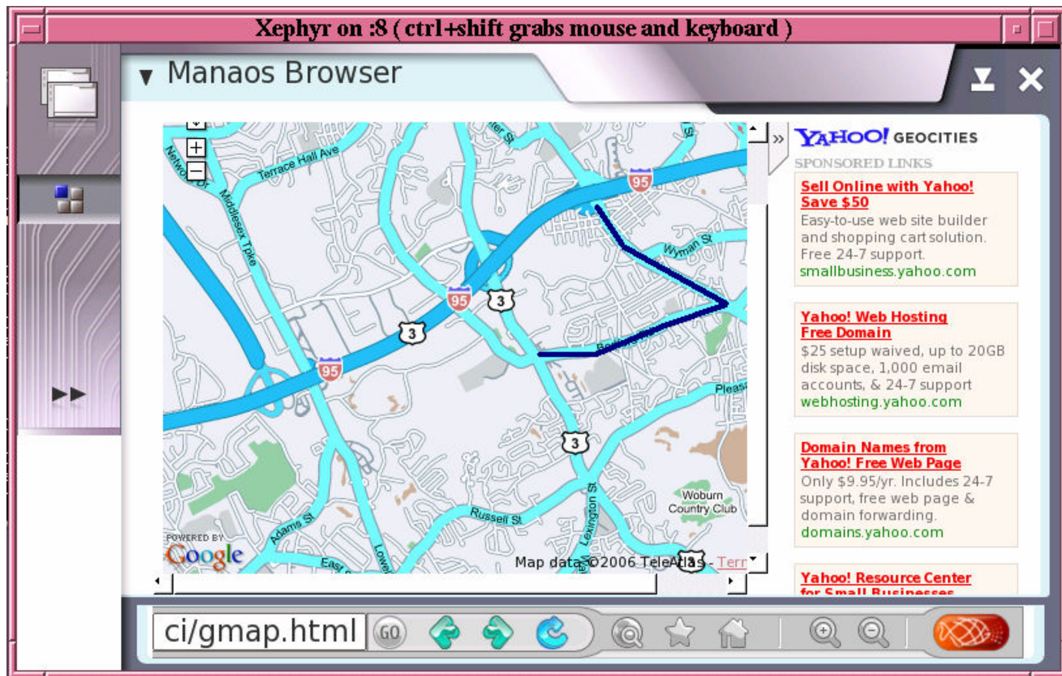

Figure 8: Manaos Browser for Nokia 770 (Mozilla Minimo) running mashup Google Map application.

The third implementation was made on Symbian 3.0 platform and Series 60 UI version 9. The Nokia S60 Open Source Browser was extended with the DCI platform. The location node obtained location data directly from a GPS device. The implementation was done for the Enterprise phone E60 coupled with the Nokia GPS module LD-1W. The implementation was carried out as part of extending an internal research platform and hence, it is not available for public use yet.

The DCI implementation forms one part of framework implementation and was tested with a Google Map application described in Chapter 11. The implementation was demonstrated inside Nokia and W3C as part of specification demonstration. The demonstration feedback has been positive and future extensions that are needed for full DCI framework implementation were approved. Almost all functionalities of DCI specification was implemented except the functionality to add proprietary search algorithms (for full DCI specification, please refer Appendix A). The implementation uses a static ontology where property representations have been statically modeled. DCI is meant to support dynamic addition of property nodes but the implementation currently does not support this. The data providers are statically linked to the model and

hence no provision for dynamic session establishment was provided. The next stage of framework implementation will have support for local and remote data provision with support for dynamic sessions. As such, provider interfaces are not standardized and hence a provider implementation was not needed for DCI demonstration. In our next DCI implementation, we aim at providing support for dynamic reference to a static ontology, support for dynamic providers, metadata interfaces and access policy support. Our immediate plans for the implementation include a full demonstration of DCI for candidate recommendation process of W3C, and conduct user tests utilizing more dynamic applications that are based on location properties and sensors that come embedded with current generation smart phones.

9. Context Provision

Providing intuitive access methods for delivery context only forms one part of the picture. There is also a need for an infrastructure that supports and provides such services and data. Context provision refers to the mode of providing context data through provider services. Context data providers can reside locally on the device, or they can be network based and the data itself can be static or dynamic. The ideal infrastructure should provide a clear separation between the various layers: data provisioning and service utilization logic. Moreover, considering the amount of disparate systems and technologies present today, it is imperative that all levels such as semantic and communication levels interoperate so that economy of scale can be achieved. Wherever possible, industry standards should be widely adopted so that existing systems can be reused to a large extent with minimal modifications.

The following sections in this chapter presents two context provisioning models that have been internally developed within Nokia. The first, called CREDO, developed by Dana Pavel et al., within Nokia Research Center is a SIP-event based framework that uses Ontology concepts for service description, discovery and provisioning. The second model is a conceptual framework developed by me that uses an Agent-based framework for context provisioning. Since the provisioning model is conceptual, I used a metamodel framework for describing the various entities involved. The two provisioning models are different. CREDO uses standard protocols and ontology based service discovery for context provisioning. CREDO has been implemented and also integrated with the DCI model. The agent-based model is a conceptual framework that is yet to be implemented and tested.

9.1 CREDO: A SIP-event based framework for context provisioning

CREDO is a context provisioning framework that has been internally developed within Nokia Research Center by Dana Pavel and Dirk Trossen [2006]. The context provision framework relies heavily on Session Initiation Protocol (SIP) [Rosenberg, 2002] and SIP-Events [Roach, 2002] work done within IETF. SIP enables separation of user identifier (URI) from endpoint identifier (IP) address, enabling application layer mobility of devices. SIP provides a separate signalling channel (and session setup) from the actual data channel. SIP has been chosen by virtually all mobile standardization bodies for future Internet multimedia services. The most important extension to the basic SIP framework is SIP-Events, which uses SIP for creating an event delivery framework for the Internet.

The specific semantic of SIP events is not specified in SIP-Events framework [Roach, 2002]. The semantics are supposed to be defined in separate standardization

documents, specifying for instance, behaviour of network entities, format of state information and rate limitations for notifications on state changes. An example for such specific event description is the presence event, describing the current presence state of a user in the Internet. SIP-Events therefore provide a very powerful tool to implement delivery of any event over the Internet. For describing the semantics of data provisioning services, we use W3C's OWL-S (Semantic Markup for Web Services). As described by Martin et al. [2004], OWL-S is a submission to W3C that specifies an OWL-based semantic description of a service. Using OWL-S enables the automation of service discovery, service invocation, service execution monitoring, and service (de-) composition. OWL-S provides means for service ontologies definition, allowing for describing services on a semantic level.

As part of our efforts to provide a fully integrated context access framework, I integrated DCI as a mechanism for context access to consumer applications with the CREDO framework. CREDO provides dynamic context data that is highly distributed to the DCI framework running on a client device. Figure 9 shows the context access framework using CREDO distributed context provision architecture.

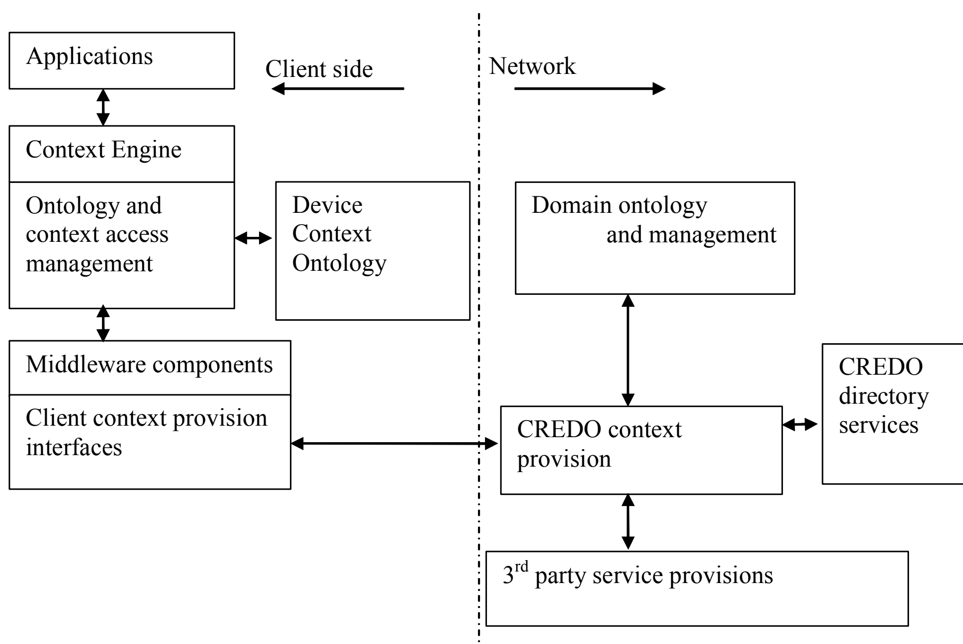

Figure 9: Context access framework using CREDO provision services [Sathish, Pavel and Trossen, 2006].

The context provider communicates with the client through the client context provision interfaces. A CREDO client API set integrated with the client context provision interfaces connects to the CREDO server. The Context Engine component (shown in Figure 9) provides the necessary interfaces for context access to consumer

applications. The context interface (in our work) is the DCI API. The ontology and context access management module is responsible for providing security and integrity of the context representation model (see Section 3.2 for more details). The client context provision interfaces provide coupling between provider specific interfaces and context representation model. The CREDO provisioning model is described in more detail below.

Figure 10 shows a high-level view of the CREDO architecture. In CREDO, each distributed element is based on a *middleware*, providing a common platform for context-aware applications. This middleware implements common functionality for discovery and provisioning of context information to the different entities with ontology and access authorization support.

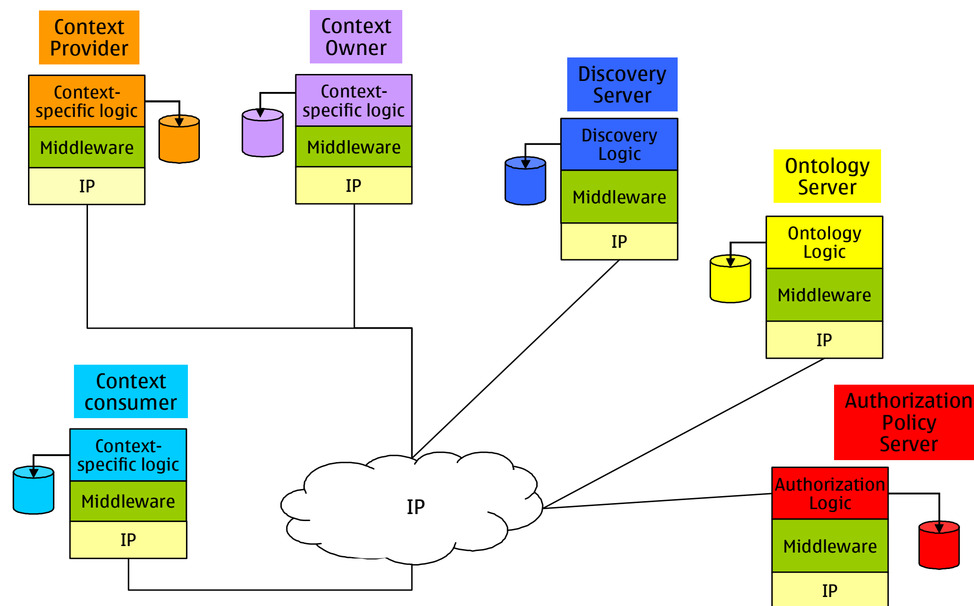

Figure 10: CREDO context provisioning system architecture [Pavel and Trossen, 2006].

Within each element in the architecture, *component specific functionality* should be implemented in addition to the common middleware (shown in green boxes). The element specific functionality is outside the scope of the platform and can implement proprietary and differentiating functionality, such as reasoning. The logic will use the common middleware for implementing its functionality, wherever possible. Since a full explanation of CREDO is outside the scope of this thesis, a brief explanation of the various components is given in the following sections.

9.1.1 Context consumers, providers and owners

The main components of the architecture are the context owners, providers and consumers. While this is not a new concept, it is not quite usual for using a service oriented view to model a context-aware system. Also, context owners and providers are considered separate since they do not necessarily have to be the same in most deployments. A context provider can simply be an intermediary for context provisioning (e.g., a Location provider) or an aggregator, combining data from various sources (e.g., a Meeting provider). Note that the topmost context consumer in the architecture would be the application logic that makes use of the obtained context information for its particular use case. Within our context access framework, shown in Figure 9, the topmost context consumer constitutes the context provider, delivering information directly to the DCI-based client. For each piece of context information provided or received, it is likely that there is need for some context-specific *recognition* and *reasoning* to process the actual information. Such specific logic could serve as a differentiating element in a (context) service offering of particular providers, e.g., through the quality of the reasoning method. Part of the context-specific logic is also the realization of *aggregation* functionality. Aggregation is the process of collecting information from various context providers, processing it and offering some derived information further to other consumers. These hierarchies of context providers are built through an inter-play of discovery, aggregation, acquisition, and ontology functionalities provided by the middleware.

9.1.2 Authorization policy component

This component authorizes the transfer of data from context providers to context consumers. It allows owners of the context to have control over their information. The middleware for this component provides generic functionality to manage and retrieve access policies for certain pieces of context information. These access policies are used in the actual context provisioning to ensure proper access rights for each subscription before granting the subscription eventually. However, the access policy could specify that access is supposed to be granted at the time of subscription. For this, the component middleware provides additional functionality, e.g., through some HTTP-based web forms.

9.1.3 Discovery Component

This component provides functionality to discover context sources within the system. It is important to note that while we describe it as a single component, there could actually

be fully distributed federations of discovery servers working together in providing the required functionality. At the middleware level, this component should insure a uniform system-wide discovery of context sources. This is achieved through providing a subscription-based mechanism, which allows for discovering but also subscribing to the future availability of context information. This discovery or availability request itself uses pointers to ontologies to allow for defining one’s own context ontologies.

9.1.4 Ontology Component

While most of the ontologies used in our solution could be directly addressed by URLs from their respective locations, certain ontologies would be local to the context-aware system. For example, in the current implementation (as will also be discussed later), own ontologies are created for representing the relationships between our middleware components. It can also be envisioned that certain other ontologies would be kept local within the particular deployment, e.g., within enterprises. The middleware part of this component should ensure proper access to existing ontologies, as well as other operations with ontologies that a certain consumer might require. It is to be noted that the ontology component does not have to be mapped onto a single entity but can instead use federations of ontology servers. As part of the ontology logic, functionalities like ontology maintenance (such as storage, verification, merging, mapping, and others.), proper format adjustment of the ontology, and reasoning logic for selecting an appropriate ontology can be envisioned.

For more details on CREDO framework and implementation, please refer to work by Pavel and Trossen [2006].

9.2 CREDO and DCI

A first level integration of DCI and CREDO has been completed. A major requirement in providing a full fledged context access mechanism is that the framework has to inter-operate with different types of provisioning systems. There are different modes of providing data services to the context model such as direct integration locally, distributed protocol services such as SIP based, web services based, other proprietary mechanisms etc. Standardized interfaces would help alleviate the problem to a certain extent but in certain cases, more proprietary integration modes may be warranted.

Another dimension to the problem is instigation of session establishment. There would be cases where the context service framework performs discovery of services (due to application requirements) while there would be cases where the provider pushes their service to the context model. We have adopted a transition approach where the first step is to provide adequate nodes in the DCI tree for CREDO provisioning service.

The DCI implementation does not employ discovery services (in this version) and a tight coupling between the node interfaces and the CREDO client API has been done. Our first level experiments with providing weather data (pressure, humidity) and location (through CellID) have been done and more property types are expected to be added in the near future.

The next step is to provide client initiated session establishment through use of SIP addressing for CREDO service and SIP proxy. The preferred solution for catering of different service providers require using the DCI provider interface and coupling with different protocol stacks. This requires defining a clear structure for metadata interface of DCI nodes so that applications can specify session establishment modes and procedures. Since this has not been developed, we opted for a more direct integration at this stage. We are also planning to develop additional capability for metadata interface which can accommodate application supplied initialization data that would be applicable for a particular session.

9.3 Agents and Context

The following sections describe an agent-based context provisioning model that can be interfaced with the DCI based adaptation architecture.

Agent-based computing is a new paradigm analogous with the metaphor of computing as a social activity. An agent [FIPA-Agents] is an autonomous entity capable of interaction between intelligent and sometimes independent entities. In this sense, an agent can be defined as "*An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future*" [Stan and Graesser, 1996]. An agent is capable of adaptation and/or bringing about adaptation with respect to a changing environment. An agent thus behaves in a metaphorical manner to the way humans interact within their social context. Agent technology can thus be termed as a disruptive approach to early computing paradigms where autonomy, coalitions and ecosystems did not make much sense.

Several categorizations have been put forward for agents. One such categorization classifies agents into [Agentlink, 2006]:

- mobile agents,
- interface agents,
- collaborative agents,
- information agents,
- reactive agents,
- hybrid agents, and
- heterogeneous agents

In this context, priority is given to the roles that agents may play within a particular context rather than a typed classification as one described above. However agents are assumed to possess the following attributes [Agentlink, 2006]:

- autonomy,
- mobile,
- proactive,
- goal-oriented,
- collaborative,
- communicative,
- adaptivity, and
- temporal continuity.

9.3.1 Operation Framework for Agent-Based Context Provision

A high level operation framework where an agent-based context provisioning service could work alongside a client-side supporting system is shown in Figure 11.

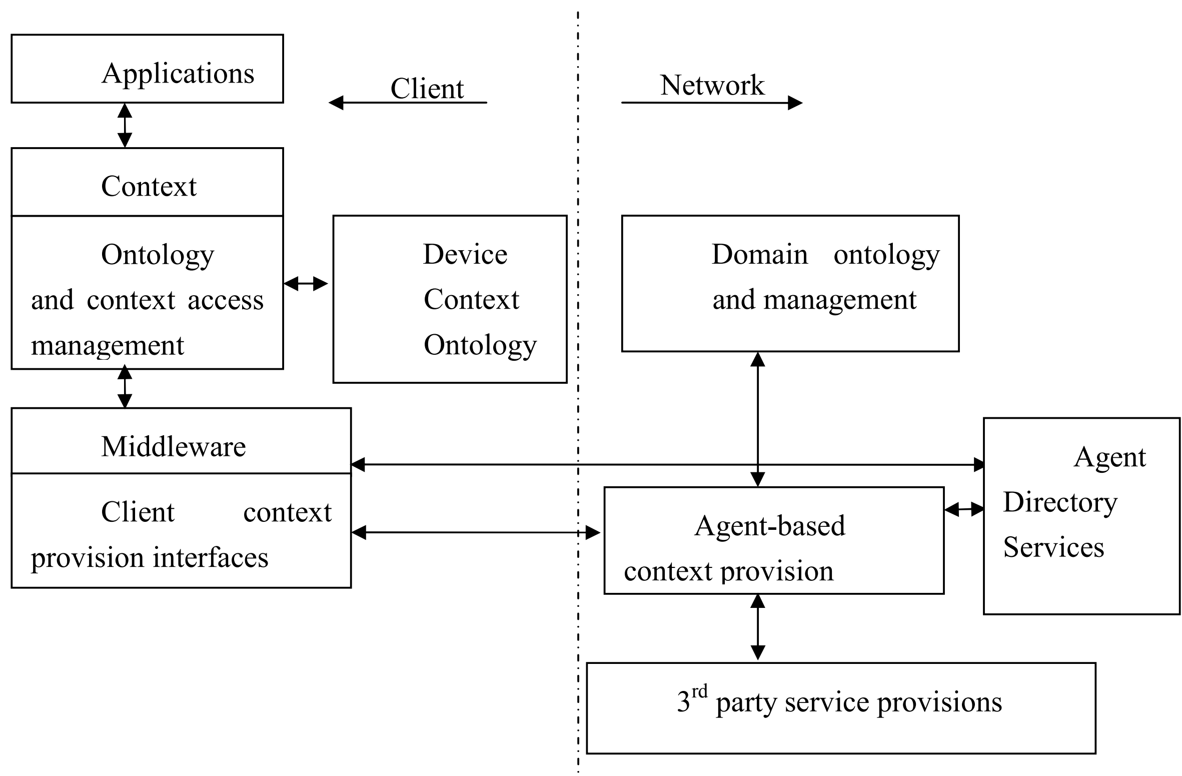

Figure 11: Agent-based context provisioning framework

Figure 11 shows an agent-based context provisioning framework. Applications access the device context through a context consumer service (such as Delivery Context Interface). The device context ontology component maintains the ontology that the context consumer part has to conform to. The ontology and context access management component manages access to context consumer side by referring the device context ontology component and maintaining the concepts and integrity of context representation. Suppose the context consumer side exposes the device properties as a tree structure (DCI) with pre-defined hierarchies with the context schema (Device Context Ontology). The ontology and context access management component provides access to the device context tree to a context provider by checking the ontology and making sure all integrity and security constraints are satisfied by the requesting context provider. The client context provision interfaces represent the provider API part and related components of the context access mechanism. This would consist of a provider management component, the provider API part and sets of protocol stacks that may be required by the different external context provider services (example: SIP, Web Services etc).

The context services are discovered and in certain cases, even provided by autonomous components (agents). This is shown as the agent-based context provision component on the network side. The agent-based services utilize the domain ontology services to understand and process domain specific context provisioning. Agents would also rely on third party services for raw or abstract context data. There would be different types of agents providing specific services. These are explained in further sections. The agent directory services provide white (for finding a specific service provider) and yellow page (for finding services based on service description) services for service registration and discovery.

The following sections describe two metamodels that have been developed for agent based service provision. The agent family metamodel provides a modeling platform for describing the different types of agents that may be present in a domain specific context framework. The agent component metamodel provides a description of the internal constituents of an agent and a model (instance of a metamodel) can choose which components may be needed for implementing a particular type of agent.

9.3.2 Agent Context Provisioning Model

The Agent Family metamodel defines a methodology that can be used to describe a framework corresponding to a particular domain where different types of agents interact with other entities. The main entities that an agent can interact with are the client, registration entities, data provision entities, subscription entities as well as other agents. The agent family metamodel for context provision is shown in Figure 12. The

metamodel has been developed using the GOPRR metamodeling language. A brief introduction to GOPRR language is given in Appendix D.

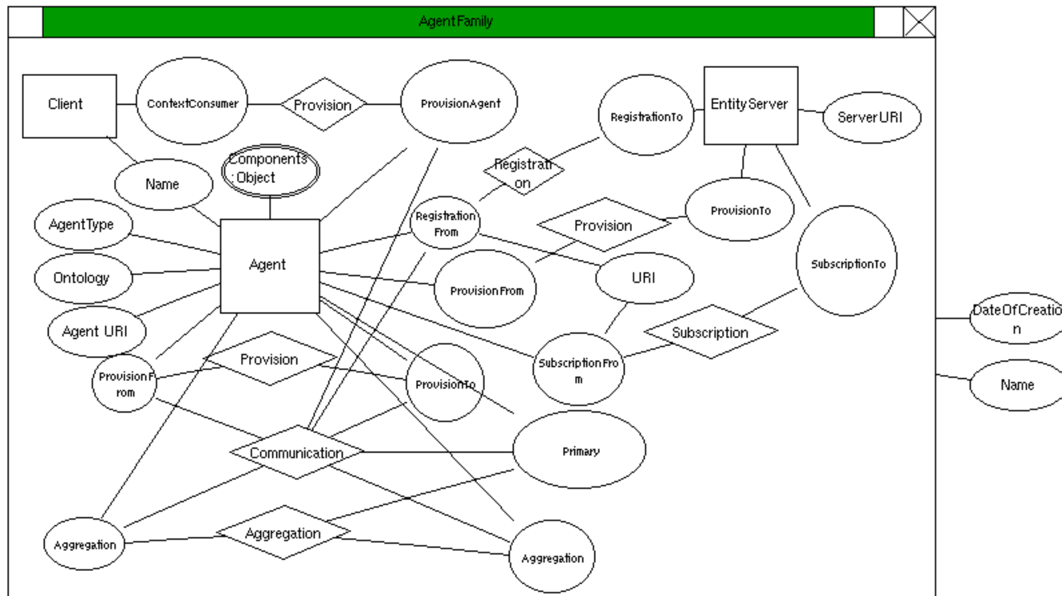

Figure 12: The Agent family metamodel.

The metamodel follows the notation that each object takes on different roles (shown as circles) that are connected to other roles through a relationship (diamond symbol). All entities (objects, roles, relationships, and graph) can have properties that are denoted as ellipses.

There can be many different kinds of agents available. Some of them are the following:

- Context Interface Access Agents: These are agents that have a one-to-one contact with the context access model (client) that applications use for context consumption.
- Context Aggregator Agents: These are agents that are capable of aggregating context data from multiple agents to provide a higher abstraction context.
- Primary Provider Agents: these agents interact directly with context provisioning mechanisms such as web services, location information provider etc and provide context data in an “agent readable” form.
- Service Discovery Agents: These agents are capable of discovering new services. These are optional agents as this capability can also be integrated with each agent.
- Communication translation agents: These are agents that can translate communication protocols between other agents. There would be agents that talk

other languages and the services of such agents can be subscribed to that act as a proxy/translation between those agents.

It is to be noted that even though there are specific roles assigned for the agents, each agent can also perform other agent functions partly or fully. These additional functionalities or those that form part of an agent implementation are represented as a collection of object properties for each agent. It is also to be noted that not all agent functions described earlier have been modeled for this task.

An agent framework also consists of other entities as well that aid in context provisioning. Prominent among them are the clients that use the agent platform to subscribe for context data, the different types of agents and the server entities such as registration server, subscription servers, provisioning servers etc. In addition, the agents can also communicate with each other. Based on this, a metamodel was created that describes an agent framework comprising all the different entities. We call this graph Agent family denoting the families of entities that can constitute a framework.

The main objects are the following:

- Client System: This is the client to the agent platform system. The Context Representation Role lists the required context data in its model that has to be provided or provisioned through the agent platform. This is also synonymous to the agent client platform. The CRS can talk to one or more agents simultaneously. The client system has a unique name as its property.
- Agent Object: The agent object is the main logical object for the system. Agents can work in many roles as mentioned earlier. Agents in each role have their own associated set of properties and relationships. An agent will communicate with other entities or agents depending on their role. The agent can have a decomposition relationship to a module diagram.
- Entity Server objects: There are mainly two roles for this object. One is to act as a registration server. The other is to act as context provider services. There could be other roles but within the current domain model, only these two roles are considered. The entity servers have unique URI that identifies them.

The agent object takes the roles of different agents described above. The **agent objects** have the following properties:

- Agent URI – a unique string value that identifies the agent,
- Agent Type – this is a string value that identifies the type of an agent i.e. the functions or role the agent would perform,
- Components – this is represented as a collection of objects and represents the functional components that make up the agent, and
- Agent Ontology – this property describes the services that the agent provides. This is a much more specific description of the agent's function than just categorizing as is denoted by the Agent Type property.

Each object performs different roles depending on the context within which the object operates. The major roles for the Agent Family metamodel objects are listed in the following section.

9.3.2.1 Role for Client System

Context Representation: This role provides a representation for the context data that the agent brings in. Agents can bring in all types of context data some of which can be aggregation of others. In order to maintain the relations between different context data and to have certain taxonomical classification between data, we decided to represent the CR role as a tree where each of the nodes would represent a context data.

9.3.2.2 Roles for Agent

Provider: This is the role of the agent when it interfaces with the context representation module. The provider role is bound to the CR role through the “Context provision” relationship.

Primary: This role is for primary context providers. Primary providers get “raw” data from direct sources, can perform some sort of semantic abstraction and sends them to other agents. A primary agent gets data only from one single source i.e. either a server entity or another agent.

Aggregator: The role of an aggregator is to gather context data from other agents, aggregate them and form an output of higher abstraction. The aggregator takes in input from one or more agents and provides a single output. The aggregator has a set of aggregation rules that the developer uses to write down semantics that aggregate context data.

Registration: The role of registration agent is to perform registration services at the registration directory services so that agents can discover each other.

9.3.2.3 Role for entity servers

Registration Server: This server is responsible for accepting, managing and maintaining a repository of agent ontology and identifiers. Agents use the registration server to register agent locations, the services that they offer etc. Agents can query the registration server for agents offering particular services. The registration server can also perform other complicated services such as mapping between ontological queries but this is outside the scope of the current report.

Context Provider: Such servers provide agents with actual context data related to a client. Agents contact their servers and communicate using the context provider

communication language. The data that agents get from such services would be used by other agents such as aggregators or direct providers to feed data onto the context representation mechanism.

In addition, all agents share some common functional modules:

Registration module: This module is used by all agents to perform registration function with a registry server. The registration protocol is registration server specific and can use some popular protocol like Session Initiation Protocol (SIP) along with an ontology describing agent functionality such as OWL-S.

A subscription module: The methods employed would be agent behavior specific.

Subscriber message generator (SM): The subscriber message generator is a module that includes an Agent Communication Language (ACL) parser and the ACL response generator. The subscriber module is linked to the agent logic module

The agent logic module: This is the core of the agent. The ALM performs logical decisions based on messages parsed by the SM. The agent logic module depends on others such as behavior module, intention module etc but these are currently outside scope of this report.

The transport module: The transport module is responsible for generating the final message (along with transport parameters) that provides a wrapping for the ACL message. The transport module is linked to the agent logic module which in turn is connected to agent SM module

9.3.3 Agent Component Model

An agent component model has been developed using the GOPRR metamodeling language. The agent component model describes the components that make up an agent. This is different from the agent family model that shows the different roles an agent would play within a context provisioning framework. The AgentComponent graph maintains an explosion relation with the AgentFamily model. A brief introduction to GOPRR language is given in Appendix D. The Agent Component metamodel is shown in Figure 13.

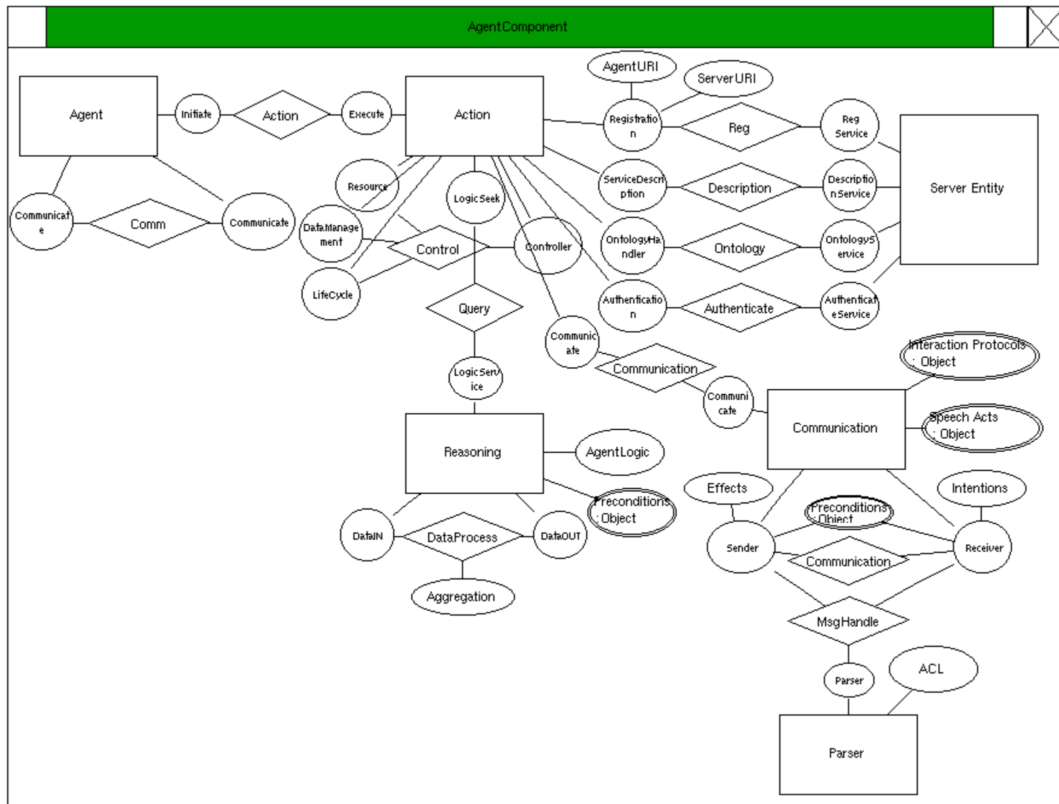

Figure 13: The Agent Component Metamodel.

An agent component metamodel is shown in Figure 13. The metamodel shows the different internal components that constitute an agent. An agent model would form an instance of a subset of the components described in the metamodel. The model shown above is a graphical metamodel intended to convey certain functional components that an agent can have. It is to be noted that there can be several different models for agents and the metamodel in Figure 13 has been developed for illustration purpose only.

The agent component metamodel in Figure 13 has four objects (shown as rectangles): Agent object, Action object, Reasoning object, Communication object and a Parser object. The metamodel also shows a Server Entity object denoting server side services that an agent might utilize for different purposes. The agent object denotes the agent interface that interacts with the rest of the agent components. The agent object takes two major roles: as a communicator for communicating with other agents and initiator of actions to be performed by the action object representing the agent functionalities. The agent object in its initiator role maintains an action relationship with the execute role of the action object. The action object is responsible for carrying out the various functions of the agent. The main roles that action object assumes (for the metamodel in Figure 13) are:

- Execute: executing instructions passed by the agent object through an action relationship.
- Resource: The action object manages the resources needed by the agent in this role.
- Data management: The data management object manages agent related data through a control relation with the controller role.
- Lifecycle: The action object manages the lifecycle of the agent through this role. The control is carried out through the controller role.
- Logic Seek: The agent logic is sought from the reasoning object through the logic seek role. This enables the action object to decide what action needs to be performed. The logic seek role is connected to the logic service role of the reasoning object through a query relationship.
- Controller: The controller role acts as the controller for each major role such as resource, data management and lifecycle. Each major role could have its own separate controller but a single controller is used here to simplify the metamodel. The controller is related to its connected roles through a control relationship.
- Registration: This is one of the actions that occur between an external entity and an agent. Agents register themselves with a white page service such as a registration server. The role has two properties, an agent URI for identifying the agent and a server URI which is the URI for the registration server.
- Service Description: The agent provides its service description through the service description role. The service description can be provided through standard description formats such as OWL-S. The service description can be combined with the registration service but it has been separated here to highlight the functional differences.
- Ontology handler: The ontology handler role understands the ontology for the domain and feeds to the rest of the agent components through the action object. The ontology will be managed by external services and the ontology handler maintains an ontology relation with the ontology services in the network.
- Communicate: The action object communicates with the communicate object through this role. The role is related through the communication relationship.
- Authentication: Agents need to be authenticated in order to provide trusted services to service requestors. The authentication can be provided by external services (authentication services) with the authentication role of the action object through an “authenticate” relationship.

The reasoning object handles the agent logic. For handling the agent logic, the main role the object represents is the agent logic role responsible for taking decisions on behalf of the agent. The reasoning object has an object collection property – beliefs that represent the agent beliefs about the environment. The belief property would be

continually updated as new information is processed by the agent. The object processes new information through the *data in* role with a data process (representing processing of information) relationship with a *data out* role. The *data process* role has an aggregation property determining whether the processing relation has to aggregate data from multiple agents in order to perform a *data out* role.

The communication object handles all communications for the agent and also feeds data to the reasoning object through the action object. This can also be done directly but has been so designed so as to allow the action object to be the central controller. The communication object is responsible for formulating the message packet in an appropriate agent communication language using domain specific speech act protocols. The communication object has several property collections. The interaction protocols object property provides templates for communication. The speech acts object represent speech act related logic and understanding. The communication object acts in sender and receiver roles between agents. The sender role has an *effects* property denoting the sender's beliefs of the effects the communication can cause on the receiver while the receiver has an *intentions* property denoting the intentions of the sender with the communication. The sender and the receiver roles share a *precondition* property collection representing domain specific knowledge and preconditions needed before the communication has to happen.

The communication object packages the intended communication in the appropriate format as well as parses a received message through the parser object. The sender and receiver roles interact with the parser role through a message handle relationship. The parser object has an ACL property denoting the communication language specific syntax and semantics needed for building the message packet.

The server object is an external entity provider agent specific service such as white and yellow page service. Agents register with the registration service, discover other agents through the discovery service; obtain domain specific concepts through the ontology service and authentication of other agents and services through an authentication service. There could be other services for agents but since the model is intended to provide an insight into agent components, the services has been addressed at a general level.

9.4 Ontology for Context Domain

Agents are autonomous entities that collaborate with other agents/entities to perform certain tasks within a particular context or environment. In order for agents to understand each other, they should talk the same vocabulary and understand the concepts within the domain. Ontology is a shared vocabulary and agreed upon meanings

to describe a subject domain. Thus agents or systems that communicate with each other should share a common ontology.

For a fully functional context-aware framework, the consumers and providers have to be in synch with the domain concepts. For an agent based context provisioning system, the span of ontology understanding is much wider. Agents work in inter-domain ranges and the ontology has to convey information across domains. Management of such ontology can be complex as outlined in Chapter 2. Besides, intelligent provisioning mechanisms have to deal with multiple ontologies that may be applicable to a particular context. These can vary between the types of service that is requested. There would be context ontologies, agent ontology (communication concepts), service ontologies and application ontologies that can be in play. The domain of working can also change based on the type of service that is being requested. An example would be an automated network (access) selection mechanism that can be automated based on application and user requirements. Context can play a significant role in such systems but now, the ontology of access providers also play a primary part. Different players would have to develop their own ontologies with a requirement that a single reasoning engine can process and understand the concepts being described. Example delivery context ontology developed using OWL is shown in Appendix E.

10. Dynamic Applications

Dynamic or adaptive applications perform real-time adaptation based on user, system and environment (context) data. Adaptive applications are aware of current user context and perform adaptation accordingly. Adaptation here refers to service adaptation (see Chapter 2) that is performed at run time. The other adaptation contexts are content adaptation and presentation adaptation. Frameworks themselves can also adapt depending on user context, for example, during dynamic discovery of a new modality within a multimodal framework (that can also be termed service adaptation). Here, a simple application adaptation based on location data is presented. The application is shown in Figure 14.

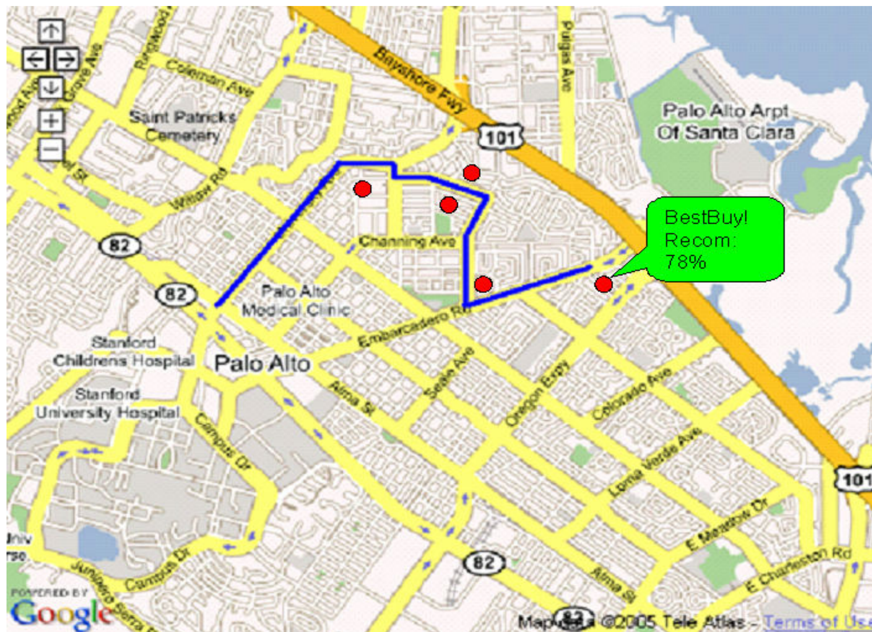

Figure 14: Mash-up Google map application showing user movement along the map. This can be combined with shopping application (shown as recommendation in map) thus combining various features together (not implemented).

The DCI-based context aware logic was first demonstrated with a Google Map application running on a Firefox browser provided with a DCI extension. The application used JavaScript based DCI context access to obtain GPS coordinates to plot a user's travel path on a map. As the user moves, the Google Map application plots the user course through use of polylines (shown in blue in Figure 14). A handle to the GPS node in the DCI tree was first obtained after traversing from the root node. An event handler was then attached to this node that listened for "***dci_prop_change***" event signifying a property value change. The handler was invoked whenever the GPS value

changed and the new value was read by the handler. The new value was then used as the current coordinates and plotted using the polylines feature provided by Google Map API.

The application shown in Figure 14 can be further enhanced by combining multiple data sources to create a true mash-up application. The CREDO framework aims to provide distributed context data such as location, time, activity and other information to be used by adaptive platforms for customized services. Such data sources can be efficiently utilized and combined with services such as a shopping assistant. The user interface itself can change reflecting the user's current context. The scenario is shown in Figure 15.

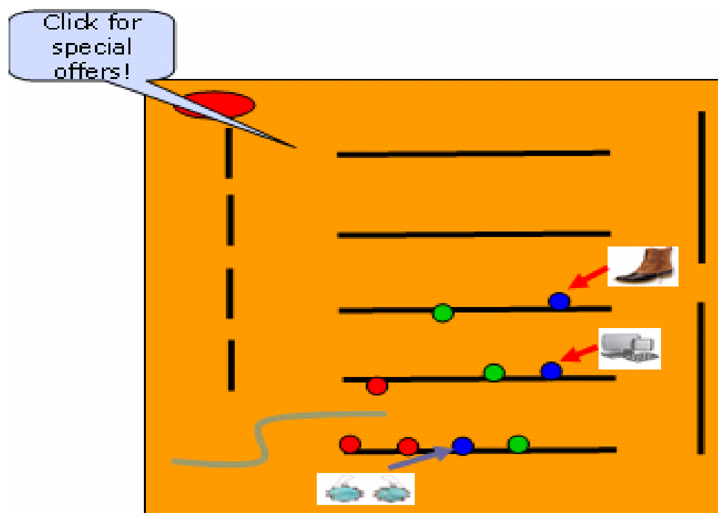

Figure 15: Extended shopping assistant application. The Google Map UI changes to location map of store as the user enters. The recommendations within the shops are highlighted.

Based on time information and user activity (such as leaving the office, getting into the car), shopping is highlighted as the next "to-do" activity. As the user starts driving, the drive path is highlighted on the map using polylines. The application then starts recommending shops along the user's way, complemented based on preferences and items to buy in the shopping list (not shown in Figure 14). The user stops at one shop and goes inside when the location information changes from GPS coordinates to some location format supported by the shop. The map application also changes to that of the shop. The DCI based application now uses location data from a new location node (ideally it uses a top level parent location node where the changes get reflected). The user course inside the shop is now plotted and the shop starts recommending items to the user. The items on the user's shopping list are highlighted at their respective location on the map as well as related recommendations, previous purchases, offers etc.

Further enhancements to the application include traffic information showing "recommended path" to user's destination (taking calendar data and appointment information), shopping recommendation for upcoming birthday, ceremony etc, coordinated shopping between family members, distributed shopping recommendation based on offers on items in shopping list.

The JavaScript code for the application shown in Figure 14 is shown in Example 1. The script provided here attaches an event handler (function ***addPoints ()***) to the location node in DCI tree. The event handler gets notified of location value changes (when ***dci_property_change*** event gets fired) and consequently updates the page by plotting the new coordinates as polylines. The application script does not provide the shopping recommendation service shown in Figure 15.

```
<script type="text/javascript">
  var map = new GMap(document.getElementById("map"));
  map.addControl(new GSmallMapControl());
  var loc =
document.getFeature("org.w3c.dci","1.0").firstChild.nextSibling.nextSibling;
  var xLoc = loc.firstChild;
  var yLoc = loc.firstChild.nextSibling;
  map.centerAndZoom(new GPoint(xLoc.value, yLoc.value));
  var coords = [];
  function addPoints(){
  coords.push(new GPoint(xLoc.value, yLoc.value));
  map.addOverlay(new GPolyline(coords));
  }
  loc.addEventListener("dci_property_change",addPoints,true);
  </script>
```

Example 1: JavaScript application code for adaptive location application.

Another adaptive application implemented was a simple browser based dynamic device configuration viewer using JavaScript. The script iterates through the DCI tree creating text boxes in a hierarchical manner on the web page based on the nodes that are present in the DCI tree. If there is a value attribute, the application will add a handler where the value change will be reflected in the corresponding text box. If new nodes are created and added, the change will be dynamically reflected through creation of corresponding text boxes and name of property.

11. Conclusion

The world is seeing an explosion of new mobile devices that vary widely in their characteristics and capabilities. There are wide variations in connectivity mechanisms, interaction mechanisms, sensors, operating systems, application platforms, applications and service offerings, hardware etc. With mobile Internet becoming a norm, Internet based services such as web pages and services have to cater to each of such devices. In order for mobile web to succeed, the single most determinant factor is that users should be able to enjoy the same desktop user experience on a mobile device. Interaction mechanisms on mobile and small devices are limited. The key factor then would be to leverage the strengths of such devices, namely mobility, connectivity options and user proximity. Thus to get accepted, services have to adapt based on device characteristics and usage situations.

In addition, they have to leverage user context, system and environment conditions utilizing new sources of data (so called secondary modalities) that will enable creation of new services. Differentiation comes from the fact that this would not be possible through standard desktop access. For applications to utilize user, system and environment data there has to be a standardized and intuitive mechanism for accessing such data especially since the focus here is the World Wide Web. To date, there is no such mechanism available that would allow generalized access to device properties other than some proprietary methods for a limited set of properties. The work presented in this thesis aims to address this issue by developing an access mechanism that utilizes current technologies, is intuitive, easily deployable and most importantly, standardized.

A framework for delivery and device context has been presented in this work. The framework addresses access mechanisms for consumers and providers of context data with support for ontologies, security and access policies. The framework provides support for tightly coupled “framework-aware” applications such as serializing delivery context for content adaptation. Even though the framework is targeted at mobile web applications, it can be generalized to accommodate any type of consumer application through integration with mobile middleware.

Our approach is based on W3C’s Delivery Context: Interface (DCI) for accessing static and dynamic system and environment properties. DCI is based on W3C’s Document Object Model (DOM) which provides a tree representation of elements in a document (the web page) with methods for traversing and manipulating this tree. DCI thus provides a tree representation of system and environment properties arranged in a hierarchy in accordance with some standard ontology. DCI also uses the DOM event model for notification of changes to the tree. Properties can use the event mechanism for notifying value changes amongst others. The context service framework, along with

consumer API, defines additional components that are needed for providing an end-to-end service platform. The DCI provider API defines a generic API through which services that provide data can get access to DCI tree. The provider part works in conjunction with an access management module that determines whether to grant access to a requesting provider. The function of access management within the framework is to control both security issues as well as determining where in the DCI tree a new provider seeking entry should be allocated a place. This would be worked out subject to an ontology that lists the vocabulary and relations of properties. The DCI tree would be an instance of the ontology. A security policy based on categorization of providers and consumers have been developed. The solution mainly addresses authorization for provider access to the model and access control. The serializer module provides an API called Dynamic Device Profile (DDp) that can be used by calling applications (used by scripts embedded within markup) to serialize whole or part of DCI tree for adaptation by a content server or proxy. The serialization can take place during a session (different from current models where content adaptation takes place only before the page gets loaded) supporting dynamic adaptation. Such adaptation can cater for changes in underlying topology especially where mobile communication is concerned. The changes can be in network connection types, available modalities such as in a multimodal framework or services with which dynamic sessions can be established. Adding a serializer to a browser will add additional load on the browser and effects of such serializers on mobile browsers especially need to be investigated. The DDp does not mandate any particular serialization format. One approach would be to embed DDp content within the web page and another approach would be to transfer DDp content during HTTP requests as a separate payload between the client and content server. The latter approach has the advantage that DDp serialization can happen before the content is downloaded to client.

The framework is designed to support browser applications in particular. Hence it needs to be de-coupled from application specific constructs in order to be used as a generic mechanism for data share. Our approach is to model data properties through an ontology that describes any property within the DCI representation. Applications are free to interpret the descriptions within the application context and interpretations of the same property can vary between applications. This raises certain usability problems as there are no limitations on the number of applications that can listen for a property value change. Thus, multi-application disambiguation needs to be supported and addressed in a future revision of the framework.

Providing access to context data forms only one part of the picture. Context data can be locally resident or distributed. The locally resident providers can directly plug in to the framework (being DCI aware) while the remote services can be found through directory services. For remote services, there should be intuitive ways for providing

service description, service discovery and standardized mechanisms for delivery of provider data. Towards this end, we provide descriptions of two mechanisms. The first provisioning model called CREDO uses a SIP-event based mechanism for service provisioning (see CREDO, Section 9.1). Here, the SIP-event specification is extended to support transport of context data from remote data providers. The mechanism uses ontology based description of service providers with an infrastructure supporting discovery logic, provisioning and authentication services. The CREDO model requires the use of a middleware that provides basic support for SIP-event based communication and discovery. CREDO is an ideal candidate when the framework can support only one model for context provisioning. Heterogeneity of access technologies can be addressed by interfacing each access model behind a CREDO cloud. All communication between consumers and providers happen through CREDO. In reality, this may not be the case and devices can support multiple communication stacks such as Web Services or Universal Plug and Play (UPnP). The specific communication model would depend on capability limitations of client devices. The second is a conceptual framework that uses an agent-based mechanism for service provisioning. The agent-based provisioning model has not been implemented and only a metamodel has been developed. Hence, a detailed analysis of the model is difficult without implementation experience and user trials.

More standardization efforts are needed for the described framework to be fully deployable. The future work includes contributing and working with partners within W3C and OMA for providing a standard ontology that would be applicable for DCI. We shall be looking at compromises between server side and client side adaptation facilitating distributed and runtime adaptation for content and presentation. Work would also be needed in creating a fool-proof security policy and access control but the solution may also be proprietary and platform dependent. We also need a standard mechanism for representing metadata for properties as well as cater for different types of dynamic values that each property can expose. On the distributed front, we will be looking at managing distributed DCI trees, especially within Multidevice scenarios and support for remote DOM protocols. On the provider front, we shall be looking at new context data sources, abstraction and aggregation logic, federation of context sources, mediated services, and optimal provision mechanisms that can be widely deployable providing economies of scale.

A natural progression from providing an adaptive platform would be to look at specific user interaction issues within smart space environments. Smart space is a multi-user multi-device dynamic interaction environment that is aware of its physical environment working on top of heterogeneous radio technologies/software distribution platforms. It should be possible to define a uniform user interface for smart space applications utilizing the underlying context representation. The user interface can list

interaction widgets available to user and applicable within the current application context. The adaptation platform is also a perfect candidate providing abstractions between top level applications and heterogeneous platforms providing data provisioning. The abstraction can mask underlying protocol implementations and complexities through its unified interface. A separate protocol specific translation mechanism can translate between application requirements, protocol semantics and syntax. One way to approach this would be to model protocol semantics through ontologies, thereby supporting translations. This is part of our future research. In the framework described, emphasis has been placed on communication from providers to consumers and not the other way around. We can envisage several scenarios where consumers would want to communicate to providers enabling fine grained control. Even though the framework does not exclude the functionality, the specifics are yet to be worked out.

Finally, any platform developed would not succeed unless backed by compelling and usable applications. Towards this end, we would be looking at next generation adaptive applications that can leverage new data sources opening up the full potential of mobility. This should enable new interaction modes and new genre of applications ultimately bringing value to the user.

References

[Abowd, Dey, Brown, Davies, Smith and Steggles, 1999] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith and Pete Steggles, Towards a better understanding of context and context-awareness source. In: *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, 1999, 304-307.

[Agentlink, 2006] Agent Technology: Computing as interaction, a roadmap for agent-based computing. Available at: http://www.agentlink.org/roadmap/al3rm.pdf

[Barnett, Bodell, Raggett and Wahbe, 2006] Jim Barnett, Mike Bodell, Dave Ragett and Andrew Wahbe, Multimodal architecture and interfaces, W3C working draft April 2006. Available at: http://www.w3.org/TR/mmi-arch/

[Biegel and Cahill, 2004] Gregory Biegel, and Vinny Cahill, A framework for developing mobile, context-aware applications. In: *Proceedings of Second IEEE Annual Conference on Pervasive Computing and Communications, PERCOM*, 2004, 361.

[Bos, Celik, Hickson and Wium Lie, 2006] Bert Bos, Tantek Celik, Ian Hickson and Hakon Wium Lie, Cascading Style Sheets (CSS 2.1), W3C working draft, November 2006. Available at: http://www.w3.org/TR/CSS21/

[Brezillon, 2003] Brezillon, P., Context Dynamic and Explanation in Contextual Graphs. In: Modeling and using context (CONTEXT-03), LNAI **2680**, *Springer Verlag* p.94-106.

[Brickley and Guha, 2004] Dan Brickley and R.V. Guha, RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation, April 2004. Available at: http://www.w3.org/TR/rdf-schema/

[Brinkkemper, 1990] Brinkkemper, S., Formalisation of information systems modeling. *Thesis publishers, The Netherlands*, 1990.

[Brown, Bovey and Chen, 1997] P.G. Brown, J.D. Bovey and X.Chen, Context-aware applications: From the laboratory to the marketplace. In: *IEEE Personal Communications, 4(5)*:58-64, October 1997.

[Bulterman, Grassel, Koivisto, Layaida, Michel, Mullender, and Zucker, SMIL 2005] Dick Bulterman, Guido Grassel, Antti Koivisto, Nabil Layaida, Theirry Michel, Sjoerd Mullender and Daniel Zucker, Synchronized Multimedia Integration Language (SMIL 2.1), W3C Recommendation December 2005. Available at: http://www.w3.org/TR/2005/REC-SMIL2-20051213/

[CCPP-exchange] CC/PP exchange protocol based on HTTP Extension Framework, W3C Note 24 June 1999. Available at: http://www.w3.org/TR/NOTE-CCPPexchange

[Chen, Finin and Joshi, 2003] Harry Chen, Tim Finn and Anupam Joshi, An ontology for context-aware pervasive computing environments. In: *The Knowledge Engineering Review*, archive volume **18**(3), September 2003.

[Clark and DeRose, 1999] James Clark and Steve DeRose, XML Path Language (XPath 1.0), W3C Recommendation, November 1999. Available at: http://www.w3.org/TR/xpath

[Conneg] IETF Content Negotiation Working Group concluded in October 2000. Available at: http://www.ietf.org/html.charters/OLD/conneg-charter.html

[CSS2] Cascading Style Sheets, level 2 CSS2 Specification, W3C Recommendation 12 May 1998. Available at: http://www.w3.org/TR/REC-CSS2

[DAML + OIL] DARPA Agent Markup Language and Ontology Interface Language specification. Accessible at: http://www.w3.org/TR/daml+oil-reference

[Dey and Abowd, 2000] Anint.K. Dey, Gregory Abowd, Towards a Better Understanding of Context and Context-Awareness. In: *Proceedings of workshop on The What, Who, Where, When and How of Context Awareness*, *Conference of Human Factors in Computing Systems (CHI 2000)*, April 2000.

[Dey and Sohn, 2003] Anint K. Dey and Timothy Sohn, Supporting end user programming of context-aware applications. In: *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications*, 2004. PerCom 2004, 361-165.

[Dey, 2001] Anint K. Dey, Understanding and Using Context. In: *Personal and Ubiquitous Computing Journal*, Vol **5**, No 1, 2001 pp 4-7.

[Dublin Core MI] The Dublin Core Metadata Initiative forum. Accessible at: http://dublincore.org/

[FIPA-Agents] Foundation for Intelligent Physical Agents (FIPA), IEEE computer society standards organization for promoting agent-based technology. Accessible at: http://www.fipa.org/

[FOAF] Friend Of A Friend project. Accessible at: http://www.foaf-project.org/

[Gimson, Sathish and Lewis, DCO 2006] Roger Gimson, Sailesh Sathish and Rhys Lewis, Delivery Context Overview (DCO) for Device Independence, W3C Working Group Note, March 2006. Available at: http://www.w3.org/TR/di-dco/

[Gu, Wang, Pung and Zhang, 2004] 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*, California, USA, 2004, 270-275.

[Held, Buchholz and Schill, 2002] Albert Held, Sven Buchholz and Alexander Schill, Modeling of context information for pervasive computing applications. In: *Proceedings of 6th World multiconference on Systemics, Cybernetics and Informatics (SCI)*, Orlando, Florida, July 2002.

[Hitchins, Hollan and Norman, 1986] Hutchins, E. L., Hollan, J. D. and Norman, D. A., Direct manipulation interfaces. In: *Norman and Draper (Eds.), User Centered System Design: New perspectives on human-computer interaction. Hillsdale*, NJ: Erlbaum.

[HTTP 1.1] Hypertext Transfer Protocol -- HTTP/1.1, IETF RFC-2616 June 1999. Available at: http://www.w3.org/Protocols/rfc2616/rfc2616.html

[HTTPex] An HTTP Extension Framework, IETF RFC-2774 February 2000. Available at: http://www.ietf.org/rfc/rfc2774.txt

[HTTPneg] HTTP Negotiation algorithm, Tim Berners-Lee 1992. Available at: http://www.w3.org/Protocols/HTTP/Negotiation.html

[ICAP Forum] The Internet Content Adaptation Protocol forum.
Accessible at: http://www.i-cap.org/home.html

[Ishii and Ulmer, 1997] Ishii, H. and Ulmer, B., Tangible bits: Towards seamless interface between people, bits and atoms. In: *Proceedings of ACM conference of Human Factors in Computing systems*, CHI'97 (Atlanta, GA), 234-241.

[Khedr and Karmouch, 2005] Mohamed Khedr and Ahmed Karmouch, ACAI: Agent-based context-aware infrastructure for spontaneous applications, 2005. In: *Journal of Network and Computer Applications*, pages 19-44, 2005.

[Klyne, Reynolds, Woodrow, Ohto, Hjelm, Butler and Tran 2004] Graham Klyne, Franklin Reynolds, Chris Woodrow, Hidetaka Ohto, Johan Hjelm, Mark. H. Butler, and Luu Tran, Composite Capabilities/Preference Profiles (CC/PP): Structure and Vocabularies 1.0, W3C Recommendation January 2004. Available at: http://www.w3.org/TR/CCPP-struct-vocab/

[Laamanen, 2002] Heimo Laamanen, Lecture notes on FIPA – Agents meet the semantic web? Sonera Corporation. In: Lecture notes at Helsinki University of Technology.
Available at: http://www.cs.helsinki.fi/u/eahyvone/xmlfinland2002/laamanen.pdf

[Lemlouma and Layaida, 2003] Tayeb Lemlouma and Nabil Layaida, Media resources adaptation for limited devices. In: *Proceedings of the 7th ICCC/IFIP International Conference on Electronic Publishing*, Universidade do Minho, Portugal 25-28 June 2003.

[Li, 2002] Weihua Li, Intelligent information agent with ontology on the semantic web. In: *Proceedings of the 4th World Congress on Intelligent Control and Automation*, 2002, 1501-1504, Shanghai, China.

[Li, Wu and Yang, 2005] Li Li, Baolin Wu and Yun Yang, Agent-based ontology integration for ontology-based applications. In: *Proceedings of the Australasian Ontology Workshop (AOW)*, 2005, Sydney, Australia.

[Mackay, 1990] Mackay, W., Patterns of sharing customizable software. In: *Proceedings of ACM conference in Computer Supported Cooperative Work* (CSCW'90), 1990, 209-221, Los Angeles, CA.

[Manaos Browser, 2006] Manaos Internet Browser for Nokia 770. Available at: http://extindt01.indt.org/10le/manaos/screenshots.html

[Martin et al., 2004] David Martin, Ora Lassila, Mark Burstein, Jerry Hobbs, Drew McDermott, Sheila Mcllrath, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katya Sycara, OWL-S Semantic Markup for Web Services, W3C member submission, November 2004. Available at: http://www.w3.org/Submission/OWL-S/

[MetaEdit+, MetaCase] Domain-specific modeling with MetaEdit+. Available at: http://www.metacase.com/

[Metamodeling] What is Metamodeling? @ metamodel.com. Available at: http://www.metamodel.com/staticpages/index.php?page=20021010231056977

[Moran and Dourish, 2001] Moran, T., and Dourish, P., Context-aware computing. In: *Special issue of human-computer interaction*, **16** (2-4), 2001.

[MPEG-21] MPEG-21 Overview (v.5), ISO/IEC JTC1/SC29/WG11/N5231 October 2002. Available at: http://www.chiariglione.org/mpeg/standards/mpeg-21/mpeg-21.htm

[Nokia LD-3W GPS] The Nokia LD-3W GPS module. Accessible at: http://www.nokiausa.com/accessories/item_details/1,8994,product:LD-3W,00.html

[OMA WAP Specification] Open Mobile Alliance (OMA) Wireless Application Protocol (WAP) Specification. Available at: http://www.wapforum.org/what/technical.htm

[OMA] The Open Mobile Alliance (OMA) Home. Accessible at: http://www.openmobilealliance.org/

[OMA-DPE] DPE-Work Item Definition, elaborated by OMA-BAC-UAProf group, revised 2 August 2005. Available at (member access): http://member.openmobilealliance.org/ftp/Public_documents/BAC/UAPROF/2005/OMA-UAPROF-2005-0010R05-Device-Profile-Evolution-WID.zip

[OMG IDL] The Object Management Group (OMG) Interface Definition Language (IDL) home page. Accessible at: http://www.omg.org/gettingstarted/omg_idl.htm

[Ontology] Ontology Definition: Source CMS Wiki. Accessible at: http://www.cmswiki.com/tiki-index.php?page=Ontology

[OWL] Web Ontology Language@W3C. Accessible at: http://www.w3.org/2004/OWL/

[Passani and Trasatti, WURFL] Luca Passani and Andrea Trasatti, Wireless Universal Resource File (WURFL). Available at: http://wurfl.sourceforge.net/

[Pavel and Trossen, 2006] Dana Pavel and Dirk Trossen, Context provisioning in future service environments. In: *Proceedings of International Multi-Conference on Computing in the Global Information Technology* (ICCGI), August 2006, Bucharest, Romania.

[Pixley, DOM Level 2 Events 2000] Tom Pixley, Document Object Model (DOM) Level 2 Events Specification, W3C Recommendation November, 2000. Available at: http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113/

[Ranganathan and Cambell, 2003] Ranganathan, A., and Campbell, R. H., An infrastructure for context awareness based on first order logic. In: *Proceedings of Personal and Ubiquitous Computing*, **7**(6), p 353-364.

[Roach, 2002] A. Roach, Session Initiation Protocol (SIP) – Specific Event Notification, RFC 3265, Internet Society, June 2002.

[Rodden, Cheverst, Davies and Dix, 1998] Rodden.T, K. Cheverst, N. Davies and A. Dix, Exploiting context in HCI design for mobile systems. In: *Proceedings of Workshop on Human Computer Interaction with Mobile Devices*, Glasgow 1998, pp 12-17.

[Rosenberg, 2002] J. Rosenberg, SIP: Session Initiation Protocol, RFC 3261, Internet Society, June 2002.

[Roy, 1999] Roy M. Turner, A model of explicit context representation and use for intelligent agents. In: *Proceedings of Second International and Interdisciplinary Conference on Modeling and Using Context*, 1999, 375-388.

[Ryan, Pascoe and Morse, 1997] Ryan, N., Pascoe, J., and Morse, D., Enhanced reality fieldwork: the context-aware archaeological assistant. In: *Gaffney, Leusen and Exxon (eds.), Computer Applications in Archaeology,* 1997.

[Sathish and Pettay, 2006] Sailesh Sathish and Olli Pettay, Delivery context access for mobile browsing. In: *Proceedings of International Multi-Conference on Computing in the Global Information Technology (ICCGI)*, August 2006, Bucharest, Romania.

[Sathish, Pavel and Trossen, 2006] Sailesh Sathish, Dana Pavel and Dirk Trossen, Context service framework for the mobile Internet. In: *Proceedings of International Workshop on System Support for Future Mobile Computing Applications (FUMCA2006)*, in conjunction with *Eighth international conference on Ubiquitous Computing (UbiComp)*, September 2006, Irvine, California.

[Sathish, 2007] Sailesh Sathish, Using declarative models for multi-device smart space environments. Position paper for W3C workshop on *Declarative Models of Distributed Web Applications*, June 2007, Dublin, Ireland.

[Sathish and Di-Flora, 2007] Sailesh Sathish and Cristiano Di-Flora, Supporting smart space infrastructures: A dynamic context-model composition framework.

Submitted to *International Workshop on Context Aware Multimedia Systems and Applications*, part of Mobimedia 2007, Nafpaktos, Greece, August 2007.

[Schilit, Adams and Want, 1994] Bill Schilit, Norman Adams and Roy Want, Context-aware computing applications. In: *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications*, 1994, 85-90.

[Stan and Graesser, 1996] Stan Franklin and Art Graesser, Is it an agent or just a program? A Taxonomy for autonomous agents. In: *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, Springer-Verlag, 1996.

[Suchman, 1987] Suchman, L., Plans and situated actions: The problem of human-machine communication. Published by: Cambridge University Press, 1987.

[TCN] Transparent Content Negotiation in HTTP, IETF RFC-2295 March 1998. Available at: http://www.ietf.org/rfc/rfc2295.txt

[Tolvanen, 2000] Juha-Pekka Tolvanen, GOPRR metamodeling tutorial, January 2000. Available at: http://www.cs.jyu.fi/~jpt/ME2000/Me07/index.htm

[UAProf] OMA User Agent Profile. Available at: http://www.openmobilealliance.org/release_program/uap_v20.html

[Van Gigch, 1991] John P. Van Gigch, Book: System design modeling and metamodeling (The Language of Science). Publisher: Springer 1991.

[W3C DOM 2004] World Wide Web Consortium (W3C) Document Object Model (DOM) Level 3, April 2004. Available at: http://www.w3.org/DOM/DOMTR#dom3

[WAP2] Wireless Application Protocol 2.0 specifications, WAP Forum 31 July 2001. Available at: http://www.wapforum.org/what/technical.htm

[Waters, Sathish, Hosn, and Ragett, W3C DCI 2006] Keith Waters, Sailesh Sathish, Rafah Hosn, and Dave Ragett, Delivery Context: Interfaces (DCI) Accessing Static and Dynamic Properties, World Wide Web Consortium Candidate Recommendation October 2006. Available at: http://www.w3.org/TR/DPF/

[Weiser, 1991] Mark Weiser, The computer for the 21st Century. In: *Scientific American* **265**(3), p 94-104.

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

[Xerces DOM] Apache Foundation Xerces C++ Parser. Available at: http://xml.apache.org/xerces-c/

[XPCOM] Cross-Platform Component Object Model for Mozilla. Available at: http://www.mozilla.org/projects/xpcom/

Appendix A

Delivery Context: Interfaces (DCI) Accessing Static and Dynamic Properties Interface Definition (IDL) is shown below. DCI is a W3C Candidate Recommendation as of October 2006.

```
#include "dom.idl"
#include "events.idl"
#pragma prefix "dom.w3c.org"
module dci
{
 // NodeType as an addition to list in DOM3 Core
 const unsigned short DCIPROPERTY_NODE      = 15;
 const unsigned short DCICOMPONENT_NODE     = 16;
 typedef dom::DOMString DOMString;
 typedef dom::Node Node;
 typedef dom::NodeList NodeList;
 typedef events::EventTarget EventTarget
 interface DCIComponent : DCIProperty
 {
  readonly attribute DOMString version;
 };
 interface DCIPropertyFilter
 {
   boolean acceptProperty(in DCIProperty property)
                raises DCIException;
 } ;
 interface DCIProperty : Node
 {
   attribute any value;
   // raises(DCIException) on setting
   // raises(DCIException) on retrieval

   readonly attribute DOMString valueType;
   attribute DOMString propertyType;
   readonly attribute boolean readOnly;
   // used for direct association of metadata
   readonly attribute any DCIMetadataInterfaceType;
   readonly attribute any DCIMetadataInterface;
```

```
    // a pair of convenience functions that save
    // having to explicitly walk the propertytree
    NodeList searchProperty(
            in DOMString namespaceURI,
            in DOMString propertyName,
            in DCIPropertyFilter dciPropertyFilter,
            in boolean deep)
          raises(DCIException);
    boolean hasProperty(
            in DOMString namespaceURI,
            in DOMString propertyName,
            in boolean deep);
  };
```

Appendix B

The Delivery Context: Interfaces Provider API (internally developed within Nokia) is given below:

1. DOMString setDCISession (in DOMString xPathExpr, in DOMString propertyName, in DOMString namespaceURI, in DOMString propertyType, in DOMString valueType, in DOMString value, in DOMString metadataType, in DOMString metadata);

This method is called to start a session with the DCI component. The method returns a unique ID that identifies the property during further calls. If this method is called with all parameter values NULL, it is allocated a new node without attaching to any particular place in DCI tree. The return value will be NULL if the allocation was unsuccessful.

The parameters are:

DOMString XPathExpr: The XPath expression that determines where to add the allocated node to.

DOMString propertyName: The name of the property

DOMString nameSpaceURI: The namespace URI for this property

DOMString propertyType: The type definition for this property.

DOMString valueType: The string identifier for the value type.

DOMString value: The initial value for this property

DOMString metadataType: The metadata type for this property

DOMString metadata: The metadata value for this property

All of the above parameters are optional. The allocated node properties will depend on the parameters passed here.

2. DOMString getLocation (in DOMString propertyName, in DOMString namespaceURI);

This method searches the DCI tree for a particular property based on the name and/or namespace URI. This method returns the XPath expression for the property. A subsequent method call can use this expression to add a new property as a child property of this node etc. The parameters are:

DOMString propertyName: The name of the property to search for.

DOMString nameSpaceURI: The namespace URI of the property to search for.

3. Boolean hasProperty (in DOMString propertyName, in DOMString namespaceURI);

This method searches the DCI tree to check if a particular property is present. Providers can use this method to search if an entry already exists or if a child node can be added etc. This returns a Boolean value. The parameters are:

DOMString propertyName: The name of the property to search for.

DOMString nameSpaceURI: The namespace URI of the property to search for.

4. DCIProperty addProperty (in DOMString propertyID, in DOMString XPathExpr, in DOMString propertyName, in DOMString namespaceURI, in DOMString propertyType, in DOMString valueType, in DOMString value, in DOMString metadataType, in DOMString metadata);

This method adds an allocated node to the DCI tree. Before calling this method, the provider should have called ***setDCISession*** and provided with an ID. If ***setDCISession*** was called with valid parameters, calling the addProperty can change the attributes for the already added node. The parameters are:

DOMString propertyID: The allocated propertyID from setDCISession call

DOMString XPathExpr: The XPath expression that determines where to add the allocated node to.

DOMString propertyName: The name of the property

DOMString nameSpaceURI: The namespace URI for this property

DOMString propertyType: The type definition for this property.

DOMString valueType: The string identifier for the value type.

DOMString value: The initial value for this property

DOMString metadataType: The metadata type for this property

DOMString metadata: The metadataa value for this property

Once addProperty is called, the DSM is consulted to check whether the calling property has rights to be allocated a new node in the tree. If successful, a node pointer is returned. Otherwise, NULL is returned.

5. Void removeProperty (in DOMString propertyID);

This method removes an existing property from the DCI tree. The parameters are:

DOMString propertyID: The property ID allocated to the provider of this property.

6. ***Void setPropertyValue (in DOMString propertyID, in DOMString valueType, in DOMString value);***

This method is called to set the value of a property node. The parameters are:

DOMString propertyID: The property ID allocated to the provider of this property

DOMString valueType: The string identifier for the value type

DOMString value: The value for this property.

7. DOMString getPropertyMetaData (in DOMString propertyID);

This method is called to get the metadata that was set for the property in the DCI tree. This method can be used by providers to check the previous metadata and change if needed. The parameters are:

DOMString propertyID: The property ID allocated to the provider of this property

8. Void setPropertyMetaData (in DOMString propertyID);

This method is used to set the metadata for a property in the DCI tree. Any previous metadata entry would be overridden by this call. The parameters are:

DOMString propertyID: The property ID allocated to the provider of this property

9. Void setNameSpacePrefix (in DOMString propertyID, in DOMString nameSpaceURI, in DOMString prefix);

This method is used to set a namespace prefix for a namespace URI so that this prefix can be used in XPath expressions that the provider uses. This eliminates the need for a namespace resolver. The namespace prefix is only valid for the provider and is identified based on the propertyID. Prefixes have to be set before calling any method that uses namespace prefixes.

The parameters are:

DOMString propertyID: The property ID allocated to the provider of this property

DOMString nameSpaceURI: The namespace URI for which the prefix is to be set

DOMString prefix: The prefix to set for nameSpaceURI.

Appendix C

The Dynamic Device Profile IDL is shown below:

```
#include dom.idl
#include dci.idl

Module DDP {

//The serialize interface is used to serialize the properties that are present in the list
//parameter. The serializer that the method serialize would use would be the active
//serializer. This returns the serialized profile

Interface serializer {
DOMString serialize (DCIPropertyList list); //returns NULL if there is an error
};

//This filter iterates through the DCI tree and determines what all nodes need to be
//added to be serialized. The filter implementation is provided by the application
Interface DDpFilter {
Boolean includeProperty (in DCIProperty property);
};

//The Serialization list interface is used to add or remove a property node to the
//serialization list
Interface DDpSerializationList extends DCIPropertyList {
AppendProperty (in DCIProperty);
RemoveProperty (in DCIProperty);
};

//Response handler: This interface is responsible for handling a response from the server
//once a dynamic profile has been sent. This is provided by the application. There can be
//default behaviour that is implementation dependent if this interface is not provided by
//application

Interface DDPResponseHandler {
Void handleServerResponse (in DOMStringURI uriResponse);
};
```

```
//The Dynamic Device Profile interface
        Interface DDp {

//The list of serializers available
        Readonly attribute DOMStringList serializers;

//Activate a particular serializer for this session
        ActivateDDpSerializer (DOMString slString);
//get the active serializer – returns the identifier of the active serializer
        DOMString getActiveSerializer ();

//This method is called to set an application defined serializer. An identifier needs to be
//provided that identifies the serializer

        SetDDpSerializer (in DOMString identifier, in Serializer NewSerializer);

//This method returns a serializer object on passing the identifier string
        Serializer getDDpSerializer (DOMString identifier);

//This method is used to remove a serializer from the serializer list. This method raises
//an exception if the calling application does not have the right to remove a serializer
        RemoveDDpSerializer (in DOMString identifier) raises DDpException;

//Attach an application defined filter to the current serializer
        DOMString serializeWithFilter (in DDpFilter filter);

//Create an empty serialization list
        DDpSerializationList createSerializationList ();

//This method is called to serialize the list by calling the current active serializer
        DOMString serializePropertyList (in DDpSerializationList list);

//This method is called to submit the device profile to the server identified through the
//URI. The method parameter determines the type of protocol used for submission. The
//behaviour of the return value would depend upon the protocol. This is a
//asynchronous method that immediately returns. The response from the server will be
//handled by the response handler set by the application – see "setResponseHandler"
//method
```

```
        SubmitDDp (in DOMString method, in DOMString uri, in DOMString
ddpString);


//The response handler is set by the application that will be called when the server sends
//back a response. The handler will then decide based on the response as to what to do.
//If there is none specified, then there will be some default behavior that is
//implementation dependent or it can be assumed to be HTTP by default

        SetResponseHandler (DDPResponseHandler handler);
        };

//The dynamic device profile exceptions
        DDPExceptions {
        DEFAULT_SERIALIZER_REMOVE_EXCEPTION = 1;
        }
        }
```

Appendix D

Brief Introduction to GOPRR Metamodeling Language

This section briefly describes the GOPRR metamodeling language. For detailed information, readers are asked to refer to [Tolvanen, 2000], which forms the basis of this introduction.

Computer Aided Software Engineering (CASE) is the use of software tools for development and maintenance of software. CASE tools are generally associated with analysis and design of software even though CASE applies across the spectrum in software development from initial conception to final product testing. A metamodel is a conceptual model of an Information System Design (ISD) method (Brinkkemper 1990). Metamodels work at one level of abstraction and logic higher than a modeling method (Van Gigch 1991). In short, metamodels are used to construct standard modeling languages applicable to a particular method. Figure 16 (taken from Brinkkemper 1990) illustrates the relationship between metamodels and modeling.

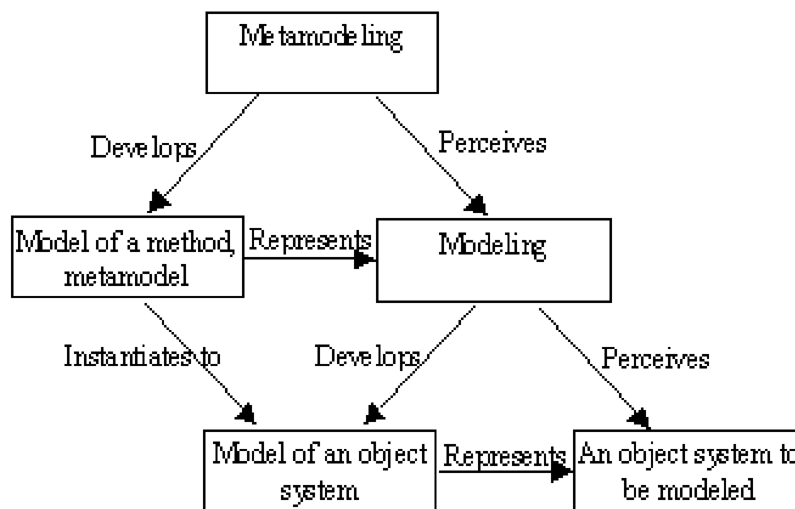

Figure 16: Metamodeling and Modeling

Metamodeling deals with designing the conceptual structure and notation of the method. The resulting metamodel captures information about the concepts, constraints, rules and representation forms used in modeling techniques. The metamodel that is created is then used when modeling a particular system to which that model can be applied. The model defines the concepts and relations that go into defining the system. A metamodel

language is used to make the metamodel. The metamodel (model of a method) represents the modeling concept applicable to the domain and an instance of the metamodel becomes the model of the system to be designed. GOPRR stands for Graph, Object, Properties, Relationship and Role and is a metamodeling language. MetaEdit+ is a CASE tool [MetaCase] that employs the GOPRR metamodeling language for creating metamodels. For the thesis, MetaEdit+ has been used in creating the metamodel. GOPRR provides a set of concepts such as collection of objects, roles, relationships and bindings describing how these objects are connected. The following section briefly describes the concepts used in GOPRR [Tolvanen, 2000]:

- Graph: Graph represents a collection of relationships, object, and role types, and bindings describing how these are connected. A graph type usually denotes a modeling technique such as class diagram or state diagram. A graph is represented by a window symbol.
- Object: An object describes a thing that can exist on its own i.e. independent of relationships and roles. Object names are typically nouns and are part of a graph. Objects can be characterized with properties. An object is represented by a rectangle.
- Properties: Properties describe characteristics of instances of other types. Property type names are typically nouns or adverbs. Each property type has a basic data type or a collection of base data types. A property is represented by an ellipse (collection by a double ellipse).
- Relationship: A relationship can exist between objects. It connects objects through roles. Semantically, relationships are usually verbs, but relationship type names are sometimes nouns or adverbs. A relationship is represented by a diamond symbol.
- Role: A role specifies how an object participates in a relationship. Semantically, roles are adverbs. Role type names are often prepositional phrases or verbs. A role is represented by a circle.
- Inclusion: An inclusion relationship can exist between a graph type and its components. Inclusion is used to combine all the main components of a technique. Inclusion is many-to-many, so that the same type can belong to many graph types.
- Binding: A binding binds participation (role) with a relationship (composition). Each role in a binding is characterized with a cardinality constraint.
- Explosion: An object, relationship or role can be linked to one or more graphs via an explosion. Explosion is typically used between different graph types.
- Decomposition: An object can be decomposed into a new graph. This feature is known as functional decomposition in data flow diagrams, or leveling of graphs

to form a hierarchy. A single decomposition is allowed for each object instance, and it applies in all graphs containing that object.

The process of GOPRR metamodeling involves the following process:

1. Identify techniques of the method
2. Identify object types
3. Define properties for each object types
4. Identify relationship types between object types
5. Define role types and connections in identified relationships
6. Inspect properties allocated to object types, relationship types and role types
7. Define role cardinalities and constraints
8. Define identifying properties for object types, role types and relationship types
9. Define the representational part of the method
10. Define method connections and global properties
11. Define method reports and analyze the metamodel

Appendix E

Ontology for Context domain

A mock ontology for dynamic delivery context is described here. The ontology has been created in OWL using Stanford University Protégé tool. The OWL ontology has been exported to RDFS. The ontology is by no means complete and particular attributes, values and metadata for properties have not been modeled. A naming convention of properties is followed where child property names are preceded with abbreviated parent node names. This was done so that it would be easier to identify particular categories to which each child property belongs.

The hierarchical model of the ontology is shown below:

- owl:Thing
    - Device
        - Associations_DEV

          Documentation: The Associations property lists the devices that are associated with the current device. This is useful in a multi-device scenario where sessions can be dynamically established. The Associations property is a child node of Device property node. Association has sibling relation with Type property.
        - Type_DEV

          Documentation: Type property describes the type of device. It can take value from one of the enumerated device types. Type property is a child node of Device property. Type has sibling Associations node.
    - Extras

      Documentation: The Extras property would contain manufacturer, user, and service specific ontology that are defined outside the current ontology scope. Extras belong to the superset of first level topology classes that fall under DCIComponent.
    - Hardware
        - Battery_HW

          Documentation: Battery property is a child of Hardware parent node. It is also the first child of Hardware node. Battery has a sibling property relationship to the super class that is the union of Memory_HW, CPU_HW and Sensors_HW nodes.
        - CPU_HW

          Documentation: CPU property is a child of Hardware parent node. CPU has a sibling property relationship to the superclass that is the union of Battery_HW, Memory_HW and Sensors_HW nodes.
        - Memory_HW

          Documentation: Memory property is a child of Hardware parent node. Memory has a sibling property relationship to the super class that is the union of Battery_HW, CPU_HW and Sensors_HW nodes.
        - Sensors_HW

Documentation: Sensors property is a child of Hardware parent node. Sensors have a sibling property relationship to the super class that is the union of Battery_HW, CPU_HW and Memory_HW nodes. In this ontology, Sensors is also the lastchild of Hardware.

- Accelerometer_SNS

  Documentation: Accelerometer sensor provides acceleration data with respect to movement of the device.

- Humidity_SNS

  Documentation: Humidity measurement of the environment that is applicable.

- Pressure_SNS

  Documentation: Pressure measurement of the environment that is applicable.

- Temperature_SNS

  Documentation: Temperature measurement of the environment that is applicable.

- Network
  - ActiveConnections_NW

    Documentation: This describes the current active connections that the device has.

  - AvailableConnections_NW

    Documentation: This describes the available connections (does not necessarily have to be active) that the device has. The connections need not be active.

  - Signal_NW

    Documentation: This describes the available connections (does not necessarily have to be active) that the device has. The connections need not be active.

    - Bandwidth_SIG

      Documentation: The bandwidth of the current active network.

    - Strength_SIG

      Documentation: Signal strength of the active network.

- Software
  - Browsers_SW

    Documentation: This property describes the browsers that the device has.

  - Java_SW

    Documentation: The Java platform supported by the device.

  - Middleware_SW

    Documentation: The middleware component and version of the device.

  - OS_SW

    Documentation: Information about operating system.

  - Other_SW

    Documentation: Other software that the device has.

- UserData
  - Location_UD
    - A-GPS_LOC

Documentation: Assisted-GPS (A-GPS). GPS receivers are connected to network at various known geographical points to provide additional location and fix. Reduces GPS search time from minutes to seconds. This allows for weaker signals than non-asssisted GPS and is available independent of network type. Accuracy (5-30M)

- AFLT_LOC

Documentation: Advanced Forward Link Trilateration is a CDMA based triangulation method. It uses IS 801 messaging and is commonly used as a hybrid with A-GPS. Accuracy is 50 - 200 M.

- AOA_LOC

Documentation: AOA denotes Angle of Arrival (for GSM networks) and can be used standalone or in hybrid to compute triangulation data.

- Cell-ID_LOC

Documentation: This is the most basic wireless location technology. The serving cell is used to locate the user. The latitude and longitude of cell gives the location. The accuracy is limited because the user can be anywhere within the coverage area of the cell. Rural areas suffer more than urban areas due to the coverage done by a single cell. It can be improved by combining with other technologies such as Timing Advance (TA). Accuracy is about 100M - 3Km and can be available with all types of networks.

- CellID-TA_LOC

Documentation: In GSM networks, Cell-ID can be combined with Timing Advance (TA) which measures handset range from the base station, including whether or not the handset is connected to the nearest cell and/or Received Signal Level which measures average signal strength. This is applicable to GSM networks.

- EFLT_LOC

Documentation: Enhanced Forward Link Trilateration is a CDMA method for triangulation that uses a 1 chip resolution reporting. It handles legacy handsets without any changes in handsets. It is typically used as a backup for non AFLT/A-GPS phones. Accuracy is 250 - 350 M.

- EOTD_LOC

Documentation: Enhanced Observed Time Difference is a network technology that calculates a user's position by triangulating the differing arrival times of network synchronization data regularly transmitted between handset and base station. It can provide accuracy to tens to metres. E-OTD is available for GSM networks. In E-OTD, the processing is done at the mobile terminal. Accuracy is (50 - 200 M).

- GPS_LOC

Documentation: Standard GPS device. GPS is usually resident on device. This is available independent of network type. Accuracy is about (5 - 30 M).

- Other_LOC

Documentation: Other location technologies not considered here.

- Sensor_LOC

Documentation: Sensor based location technologies are prevalent now. This type of location technologies is mostly proprietary.

- TDOA_LOC

Documentation: Time Difference Of Arrival is similar in concept to E-OTD in that it also uses triangulation and time delay, but the processing is carried on the network side. This applies to GSM networks.

- Presence_UD

Documentation: Presence information as a child node under UserData.

- UserInterface
  - Audio_UI
    - Input_AUD

    Documentation: The audio input capabilities of the device.

    - Output_AUD

    Documentation: The audio output capabilities of the device. This can also have networked audio devices as children or siblings.

  - Camera_UI

  Documentation: The camera User Interface available with the device.

    - Still_CAM

    Documentation: Still picture capabilities of the device.

    - Video_CAM

    Documentation: Video capabilities of the device.

  - Other_UI

  Documentation: Other UI properties.

  - Screen_UI

  Documentation: The device screen properties.

    - Brightness_SCR

    Documentation: Current brightness level of the screen.

    - Orientation_SCR

    Documentation: Current orientation of the screen.

  - Touch_UI

  Documentation: Touch modality capabilites of the device.

    - Gesture_TCH

    Documentation: Gesture engine supported by the device.

    - Haptic_TCH

    Documentation: Haptic UI capabilities of the device including kinaesthetic capabilities if any.

    - Tactile_TCH

    Documentation: Tactile features supported by the device. In some devices, tactile properties can also form child property of haptic property.

Delivery Context Ontology in RDFS: OWL ontology exported as RDFS in Protégé

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE rdf:RDF (View Source for full doctype...)>
- <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdf_="http://protege.stanford.edu/rdf" xmlns:a="http://protege.stanford.edu/system#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfA-GPS_LOC" rdfs:label="A-GPS_LOC">
<rdfs:comment>Assisted-GPS (A-GPS). GPS receivers are connected to network at various known geographical points to provide additional location and fix. Reduces GPS search time from minutes to seconds. Allows for weaker signals than non-assisted GPS. Available independent of network type. Accuracy (5-30M)</rdfs:comment>
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfLocation_UD" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfAFLT_LOC" rdfs:label="AFLT_LOC">
<rdfs:comment>Advanced Forward Link Trilateration is a CDMA based triangulation method. It uses IS 801 messaging and is commonly used as a hybrid with A-GPS. Accuracy is 50 - 200 M.</rdfs:comment>
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfLocation_UD" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfAOA_LOC" rdfs:label="AOA_LOC">
<rdfs:comment>AOA denotes Angle of Arrival (for GSM networks) and can be used standalone or in hybrid to compute triangulation data.</rdfs:comment>
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfLocation_UD" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfAccelerometer_SNS" rdfs:label="Accelerometer_SNS">
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfSensors_HW" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfActiveConnections_NW" rdfs:label="ActiveConnections_NW">
<rdfs:comment>This describes the current active connections that the device has</rdfs:comment>

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfNetwork" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfAssociations_DEV"
rdfs:label="Associations_DEV">
<rdfs:comment>The Associations property lists the devices that are associated
with the current device. This is useful in a multi-device scenario where sessions can be
dynamically established. The Associations property is a child node of Device property
node. Association has sibling relation with Type property.</rdfs:comment>
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfDevice" />
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdf__:A187" />
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdf__:A191" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfAudio_UI"
rdfs:label="Audio_UI">
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfUserInterface" />
</a:owl_class>
- <a:owl_class
rdf:about="http://protege.stanford.edu/rdfAvailableConnections_NW"
rdfs:label="AvailableConnections_NW">
<rdfs:comment>This describes the available connections (does not necessarily
have to be active) that the device has. The connections need not be
active.</rdfs:comment>
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfNetwork" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfBandwidth_SIG"
rdfs:comment="The bandwidth of the current active network"
rdfs:label="Bandwidth_SIG">
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfSignal_NW" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfBattery_HW"
rdfs:label="Battery_HW">
<rdfs:comment>Battery property is a child of Hardware parent node. It is also
the first child of Hardware node. Battery has a sibling property relationship to the
superclass that is the union of Memory_HW, CPU_HW and Sensors_HW
nodes.</rdfs:comment>
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfHardware" />
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdf__:A140" />
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdf__:A143" />
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdf__:A146" />

</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfBrightness_SCR" rdfs:comment="Current brightness level of the screen" rdfs:label="Brightness_SCR">
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfScreen_UI" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfBrowsers_SW" rdfs:comment="This property describes the browsers that the device has." rdfs:label="Browsers_SW">
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfSoftware" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfCPU_HW" rdfs:label="CPU_HW">
<rdfs:comment>CPU property is a child of Hardware parent node. CPU has a sibling property relationship to the superclass that is the union of Battery_HW, Memory_HW and Sensors_HW nodes.</rdfs:comment>
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfHardware" />
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdf__:A236" />
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdf__:A239" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfCamera_UI" rdfs:comment="The camera UI." rdfs:label="Camera_UI">
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfUserInterface" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfCell-ID_LOC" rdfs:label="Cell-ID_LOC">
<rdfs:comment>This is the most basic wireless location technology. The serving cell is used to locate the user. The latitude and longitude of cell gives the location. The accuracy is limited because the user can be anywhere within the coverage area of the cell. Rural areas suffer more than urban areas due to the coverage done by a single cell. It can be improved by combining with other technologies such as Timing Advance (TA). Accuracy (100M - 3Km). Can be available with all types of networks.</rdfs:comment>
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfLocation_UD" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfCellID-TA_LOC" rdfs:label="CellID-TA_LOC">
<rdfs:comment>In GSM networks, Cell-ID can be combined with Timing Advance (TA) which measures handset range from the base station, including whether

or not the handset is connected to the nearest cell and/or Received Signal Level which measures average signal strength. Applicable to GSM networks.</rdfs:comment>
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfLocation_UD" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfDevice" rdfs:label="Device">
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdf__:A418" />
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/system#owl_thing" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfEFLT_LOC" rdfs:label="EFLT_LOC">
<rdfs:comment>Enhanced Forward Link Trilateration is a CDMA method for triangulation that uses a 1 chip resolution reporting. It handles legacy handsets without any changes in handsets. It is typically used as a backup for non AFLT/A-GPS phones. Accuracy is 250 - 350 M.</rdfs:comment>
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfLocation_UD" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfEOTD_LOC" rdfs:label="EOTD_LOC">
<rdfs:comment>Enhanced Observed Time Difference is a network technology that calculates a users position by triangulating the differing arrival times of network synchronization data regularly transmitted between handset and base station. It can provide accuracy to tens to meters. E-OTD is available for GSM networks. In E-OTD, the processing is done at the mobile terminal. Accuracy (50 - 200 M).</rdfs:comment>
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfLocation_UD" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfExtras" rdfs:label="Extras">
<rdfs:comment>The Extras property would contain manufacturer, user, service specific ontology that are defined outside the current ontology scope. Extras belong to the superset of first level topology classes that fall under DCIComponent.</rdfs:comment>
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdf__:A355" />
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/system#owl_thing" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfGPS_LOC" rdfs:label="GPS_LOC">

<rdfs:comment>Standard GPS device. Resident on device. Available independent of network type. Accuracy (5 - 30 M).</rdfs:comment>
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfLocation_UD" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfGesture_TCH" rdfs:comment="Gesture engine supported by the device" rdfs:label="Gesture_TCH">
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfTouch_UI" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfHaptic_TCH" rdfs:label="Haptic_TCH">
<rdfs:comment>Haptic UI capabilities of the device including kinesthetic capabilities if any.</rdfs:comment>
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfTouch_UI" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfHardware" rdfs:label="Hardware">
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdf__:A381" />
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/system#owl_thing" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfHumidity_SNS" rdfs:label="Humidity_SNS">
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfSensors_HW" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfInput_AUD" rdfs:comment="The audio input capabilities of the device" rdfs:label="Input_AUD">
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfAudio_UI" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfJava_SW" rdfs:comment="The Java platform supported by the device" rdfs:label="Java_SW">
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfSoftware" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfLocation_UD" rdfs:label="Location_UD">
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfUserData" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfMemory_HW" rdfs:label="Memory_HW">

<rdfs:comment>Memory property is a child of Hardware parent node. Memory has a sibling property relationship to the superclass that is the union of Battery_HW, CPU_HW and Sensors_HW nodes.</rdfs:comment>

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfHardware" />

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdf__:A79" />

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdf__:A83" />

</a:owl_class>

- <a:owl_class rdf:about="http://protege.stanford.edu/rdfMiddleware_SW" rdfs:comment="The middleware component and version of the device" rdfs:label="Middleware_SW">

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfSoftware" />

</a:owl_class>

- <a:owl_class rdf:about="http://protege.stanford.edu/rdfNetwork" rdfs:label="Network">

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdf__:A456" />

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/system#owl_thing" />

</a:owl_class>

- <a:owl_class rdf:about="http://protege.stanford.edu/rdfOS_SW" rdfs:comment="Information about operating system." rdfs:label="OS_SW">

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfSoftware" />

</a:owl_class>

- <a:owl_class rdf:about="http://protege.stanford.edu/rdfOrientation_SCR" rdfs:comment="Current orientation of the screen." rdfs:label="Orientation_SCR">

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfScreen_UI" />

</a:owl_class>

- <a:owl_class rdf:about="http://protege.stanford.edu/rdfOther_LOC" rdfs:label="Other_LOC">

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfLocation_UD" />

</a:owl_class>

- <a:owl_class rdf:about="http://protege.stanford.edu/rdfOther_SW" rdfs:comment="Other software that the device has." rdfs:label="Other_SW">

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfSoftware" />

</a:owl_class>

- <a:owl_class rdf:about="http://protege.stanford.edu/rdfOther_UI" rdfs:comment="Other UI properties" rdfs:label="Other_UI">

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfUserInterface" />

</a:owl_class>

- <a:owl_class rdf:about="http://protege.stanford.edu/rdfOutput_AUD" rdfs:label="Output_AUD">

<rdfs:comment>The audio output capabilities of the device. This can also have networked audio devices as children or siblings.</rdfs:comment>

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfAudio_UI" />

</a:owl_class>

- <a:owl_class rdf:about="http://protege.stanford.edu/rdfPresence_UD" rdfs:label="Presence_UD">

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfUserData" />

</a:owl_class>

- <a:owl_class rdf:about="http://protege.stanford.edu/rdfPressure_SNS" rdfs:label="Pressure_SNS">

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfSensors_HW" />

</a:owl_class>

- <a:owl_class rdf:about="http://protege.stanford.edu/rdfScreen_UI" rdfs:comment="The device hardware screen properties." rdfs:label="Screen_UI">

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfUserInterface" />

</a:owl_class>

- <a:owl_class rdf:about="http://protege.stanford.edu/rdfSensor_LOC" rdfs:label="Sensor_LOC">

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfLocation_UD" />

</a:owl_class>

- <a:owl_class rdf:about="http://protege.stanford.edu/rdfSensors_HW" rdfs:label="Sensors_HW">

<rdfs:comment>Sensors property is a child of Hardware parent node. Sensors have a sibling property relationship to the superclass that is the union of Battery_HW, CPU_HW and Memory_HW nodes. In this ontology, Sensors is also the lastchild of Hardware.</rdfs:comment>

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfHardware" />

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdf__:A201" />

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdf__:A213" />

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdf__:A216" />

</a:owl_class>

- <a:owl_class rdf:about="http://protege.stanford.edu/rdfSignal_NW" rdfs:label="Signal_NW">

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfNetwork" />

</a:owl_class>

- <a:owl_class rdf:about="http://protege.stanford.edu/rdfSoftware" rdfs:label="Software">

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdf__:A59" />
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/system#owl_thing" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfStill_CAM" rdfs:comment="Still picture capabilities of the device" rdfs:label="Still_CAM">
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfCamera_UI" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfStrength_SIG" rdfs:comment="Signal strength of the active network" rdfs:label="Strength_SIG">
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfSignal_NW" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfTDOA_LOC" rdfs:label="TDOA_LOC">
<rdfs:comment>Time Difference Of Arrival is similar in concept to E-OTD in that it also uses triangulation and time delay, but the processing is carried on the network side. Applies to GSM.</rdfs:comment>
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfLocation_UD" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfTactile_TCH" rdfs:label="Tactile_TCH">
<rdfs:comment>Tactile features supported by the device. In some devices, tactile properties can also form child property of haptic property.</rdfs:comment>
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfTouch_UI" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfTemperature_SNS" rdfs:label="Temperature_SNS">
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfSensors_HW" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfTouch_UI" rdfs:comment="Touch modality capabilities of the device" rdfs:label="Touch_UI">
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfUserInterface" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfType_DEV" rdfs:label="Type_DEV">
<rdfs:comment>Type property describes the type of device. It can take value from one of the enumerated device types. Type property is a child node of Device property. Type has sibling Associations node.</rdfs:comment>
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfDevice" />

<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdf__:A30" />
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdf__:A34" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfUserData" rdfs:label="UserData">
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdf__:A278" />
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/system#owl_thing" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfUserInterface" rdfs:label="UserInterface">
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdf__:A171" />
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/system#owl_thing" />
</a:owl_class>
- <a:owl_class rdf:about="http://protege.stanford.edu/rdfVideo_CAM" rdfs:comment="Video capabilities of the device" rdfs:label="Video_CAM">
<rdfs:subClassOf rdf:resource="http://protege.stanford.edu/rdfCamera_UI" />
</a:owl_class>
<a:owl_hasvaluerestriction rdf:about="http://protege.stanford.edu/rdf__:A140" rdfs:label="isFirstChildOf _ Hardware" />
<a:owl_allvaluesfromrestriction rdf:about="http://protege.stanford.edu/rdf__:A143" rdfs:label="isChildOf Hardware" />
<a:owl_allvaluesfromrestriction rdf:about="http://protege.stanford.edu/rdf__:A146" rdfs:label="isSiblingOf (CPU_HW ⊔ Memory_HW ⊔ Sensors_HW)" />
<a:owl_unionclass rdf:about="http://protege.stanford.edu/rdf__:A148" rdfs:label="CPU_HW ⊔ Memory_HW ⊔ Sensors_HW" />
- <a:owl_somevaluesfromrestriction rdf:about="http://protege.stanford.edu/rdf__:A171">
<rdfs:label> isSiblingOf (Device ⊔ Extras ⊔ Hardware ⊔ Network ⊔ Software ⊔ UserData)</rdfs:label>
</a:owl_somevaluesfromrestriction>
<a:owl_unionclass rdf:about="http://protege.stanford.edu/rdf__:A173" rdfs:label="Device ⊔ Extras ⊔ Hardware ⊔ Network ⊔ Software ⊔ UserData" />
<a:owl_somevaluesfromrestriction rdf:about="http://protege.stanford.edu/rdf__:A187" rdfs:label="isSiblingOf Type_DEV" />

<a:owl_somevaluesfromrestriction
rdf:about="http://protege.stanford.edu/rdf__:A191" rdfs:label="isChildOf Device" />
<a:owl_allvaluesfromrestriction
rdf:about="http://protege.stanford.edu/rdf__:A201" rdfs:label="isSiblingOf
(Memory_HW ⊔ CPU_HW ⊔ Battery_HW)" />
<a:owl_unionclass rdf:about="http://protege.stanford.edu/rdf__:A203"
rdfs:label="Memory_HW ⊔ CPU_HW ⊔ Battery_HW" />
<a:owl_hasvaluerestriction rdf:about="http://protege.stanford.edu/rdf__:A213"
rdfs:label="isLastChildOf Hardware" />
<a:owl_allvaluesfromrestriction
rdf:about="http://protege.stanford.edu/rdf__:A216" rdfs:label="isChildOf Hardware" />
<a:owl_allvaluesfromrestriction
rdf:about="http://protege.stanford.edu/rdf__:A236" rdfs:label="isChildOf Hardware" />
<a:owl_allvaluesfromrestriction
rdf:about="http://protege.stanford.edu/rdf__:A239" rdfs:label="isSiblingOf
(Battery_HW ⊔ Memory_HW ⊔ Sensors_HW)" />
<a:owl_unionclass rdf:about="http://protege.stanford.edu/rdf__:A240"
rdfs:label="Battery_HW ⊔ Memory_HW ⊔ Sensors_HW" />
- <a:owl_somevaluesfromrestriction
rdf:about="http://protege.stanford.edu/rdf__:A278">
<rdfs:label>isSiblingOf (Device ⊔ Extras ⊔ Hardware ⊔ Network ⊔ Software
⊔ UserInterface)</rdfs:label>
</a:owl_somevaluesfromrestriction>
- <a:owl_unionclass rdf:about="http://protege.stanford.edu/rdf__:A280">
<rdfs:label>Device ⊔ Extras ⊔ Hardware ⊔ Network ⊔ Software ⊔
UserInterface</rdfs:label>
</a:owl_unionclass>
<a:owl_somevaluesfromrestriction
rdf:about="http://protege.stanford.edu/rdf__:A30" rdfs:label="isSiblingOf
Associations_DEV" />
<a:owl_somevaluesfromrestriction
rdf:about="http://protege.stanford.edu/rdf__:A34" rdfs:label="isChildOf Device" />
- <a:owl_somevaluesfromrestriction
rdf:about="http://protege.stanford.edu/rdf__:A355">
<rdfs:label>isSiblingOf (Device ⊔ Hardware ⊔ Network ⊔ Software ⊔
UserData ⊔ UserInterface)</rdfs:label>
</a:owl_somevaluesfromrestriction>
- <a:owl_unionclass rdf:about="http://protege.stanford.edu/rdf__:A356">

<rdfs:label>Device ⊔ Hardware ⊔ Network ⊔ Software ⊔ UserData ⊔ UserInterface</rdfs:label>
</a:owl_unionclass>
- <a:owl_somevaluesfromrestriction rdf:about="http://protege.stanford.edu/rdf__:A381">
<rdfs:label> isSiblingOf (Device ⊔ Extras ⊔ Network ⊔ Software ⊔ UserData ⊔ UserInterface)</rdfs:label>
</a:owl_somevaluesfromrestriction>
- <a:owl_unionclass rdf:about="http://protege.stanford.edu/rdf__:A383">
<rdfs:label>Device ⊔ Extras ⊔ Network ⊔ Software ⊔ UserData ⊔ UserInterface</rdfs:label>
</a:owl_unionclass>
- <a:owl_somevaluesfromrestriction rdf:about="http://protege.stanford.edu/rdf__:A418">
<rdfs:label> isSiblingOf (Hardware ⊔ Network ⊔ Software ⊔ UserData ⊔ UserInterface ⊔ Extras)</rdfs:label>
</a:owl_somevaluesfromrestriction>
- <a:owl_unionclass rdf:about="http://protege.stanford.edu/rdf__:A420">
<rdfs:label>Hardware ⊔ Network ⊔ Software ⊔ UserData ⊔ UserInterface ⊔ Extras</rdfs:label>
</a:owl_unionclass>
- <a:owl_somevaluesfromrestriction rdf:about="http://protege.stanford.edu/rdf__:A456">
<rdfs:label> isSiblingOf (Device ⊔ Extras ⊔ Hardware ⊔ Software ⊔ UserData ⊔ UserInterface)</rdfs:label>
</a:owl_somevaluesfromrestriction>
- <a:owl_unionclass rdf:about="http://protege.stanford.edu/rdf__:A458">
<rdfs:label>Device ⊔ Extras ⊔ Hardware ⊔ Software ⊔ UserData ⊔ UserInterface</rdfs:label>
</a:owl_unionclass>
- <a:owl_somevaluesfromrestriction rdf:about="http://protege.stanford.edu/rdf__:A59">
<rdfs:label> isSiblingOf (Device ⊔ Extras ⊔ Hardware ⊔ Network ⊔ UserData ⊔ UserInterface)</rdfs:label>
</a:owl_somevaluesfromrestriction>
- <a:owl_unionclass rdf:about="http://protege.stanford.edu/rdf__:A61">
<rdfs:label>Device ⊔ Extras ⊔ Hardware ⊔ Network ⊔ UserData ⊔ UserInterface</rdfs:label>
</a:owl_unionclass>

<a:owl_allvaluesfromrestriction
rdf:about="http://protege.stanford.edu/rdf__:A79" rdfs:label="isChildOf Hardware" />
<a:owl_allvaluesfromrestriction
rdf:about="http://protege.stanford.edu/rdf__:A83" rdfs:label="isSiblingOf (Battery_HW
⊔ CPU_HW ⊔ Sensors_HW)" />
<a:owl_unionclass rdf:about="http://protege.stanford.edu/rdf__:A84"
rdfs:label="Battery_HW ⊔ CPU_HW ⊔ Sensors_HW" />
- <a:owl_objectproperty rdf:about="http://protege.stanford.edu/rdfhasColor"
rdfs:label="hasColor">
<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />
</a:owl_objectproperty>
- <a:owl_objectproperty rdf:about="http://protege.stanford.edu/rdfhasHeight"
rdfs:label="hasHeight">
<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />
</a:owl_objectproperty>
- <a:owl_objectproperty rdf:about="http://protege.stanford.edu/rdfhasResolution"
rdfs:label="hasResolution">
<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />
</a:owl_objectproperty>
- <a:owl_objectproperty
rdf:about="http://protege.stanford.edu/rdfhasUAProfLink"
rdfs:label="hasUAProfLink">
<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />
</a:owl_objectproperty>
- <a:owl_objectproperty rdf:about="http://protege.stanford.edu/rdfhasValue"
rdfs:label="hasValue">
<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />
</a:owl_objectproperty>
- <a:owl_objectproperty rdf:about="http://protege.stanford.edu/rdfhasVersionNo"
rdfs:label="hasVersionNo">
<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />
</a:owl_objectproperty>
- <a:owl_objectproperty rdf:about="http://protege.stanford.edu/rdfhasWidth"
rdfs:label="hasWidth">
<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />
</a:owl_objectproperty>
- <a:owl_objectproperty rdf:about="http://protege.stanford.edu/rdfisAnscestorOf"
rdfs:label="isAnscestorOf">
<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />

</a:owl_objectproperty>
- <a:owl_objectproperty rdf:about="http://protege.stanford.edu/rdfisChildOf" rdfs:label="isChildOf">
<rdfs:subPropertyOf rdf:resource="http://protege.stanford.edu/rdfisDescendantOf" />
<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />
</a:owl_objectproperty>
- <a:owl_objectproperty rdf:about="http://protege.stanford.edu/rdfisDescendantOf" rdfs:label="isDescendantOf">
<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />
</a:owl_objectproperty>
- <a:owl_objectproperty rdf:about="http://protege.stanford.edu/rdfisDissimilarTo" rdfs:label="isDissimilarTo">
<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />
</a:owl_objectproperty>
- <a:owl_objectproperty rdf:about="http://protege.stanford.edu/rdfisEquivalentTo" rdfs:label="isEquivalentTo">
<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />
</a:owl_objectproperty>
- <a:owl_objectproperty rdf:about="http://protege.stanford.edu/rdfisFirstChildOf" rdfs:label="isFirstChildOf">
<rdfs:subPropertyOf rdf:resource="http://protege.stanford.edu/rdfisChildOf" />
<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />
</a:owl_objectproperty>
- <a:owl_objectproperty rdf:about="http://protege.stanford.edu/rdfisImmediateSiblingOf" rdfs:label="isImmediateSiblingOf">
<rdfs:subPropertyOf rdf:resource="http://protege.stanford.edu/rdfisSiblingOf" />
<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />
</a:owl_objectproperty>
- <a:owl_objectproperty rdf:about="http://protege.stanford.edu/rdfisLastChildOf" rdfs:label="isLastChildOf">
<rdfs:subPropertyOf rdf:resource="http://protege.stanford.edu/rdfisChildOf" />
<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />
</a:owl_objectproperty>
- <a:owl_objectproperty rdf:about="http://protege.stanford.edu/rdfisManufacturedBy" rdfs:label="isManufacturedBy">

<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />

</a:owl_objectproperty>

- <a:owl_objectproperty rdf:about="http://protege.stanford.edu/rdfisOfType" rdfs:label="isOfType">

<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />

</a:owl_objectproperty>

- <a:owl_objectproperty rdf:about="http://protege.stanford.edu/rdfisOwnedBy" rdfs:label="isOwnedBy">

<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />

</a:owl_objectproperty>

- <a:owl_objectproperty rdf:about="http://protege.stanford.edu/rdfisParentOf" rdfs:label="isParentOf">

<rdfs:subPropertyOf rdf:resource="http://protege.stanford.edu/rdfisAnscestorOf" />

<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />

</a:owl_objectproperty>

- <a:owl_objectproperty rdf:about="http://protege.stanford.edu/rdfisSiblingOf" rdfs:label="isSiblingOf">

<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />

</a:owl_objectproperty>

- <a:owl_objectproperty rdf:about="http://protege.stanford.edu/rdfisSimilarTo" rdfs:label="isSimilarTo">

<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />

</a:owl_objectproperty>

- <a:owl_datatypeproperty rdf:about="http://protege.stanford.edu/rdfunits" rdfs:label="units">

<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />

</a:owl_datatypeproperty>

</rdf:RDF>