Context Service Framework for the Mobile Internet

Sailesh Sathish Dana Pavel* Dirk Trossen*
Nokia Research Center
Visiokatu 1, Tampere, Finland
**Itämerenkatu 11-13, Helsinki, Finland*
{sailesh.sathish, dana.pavel, dirk.trossen}@nokia.com

Abstract

Many context aware systems have been developed especially for mobile devices, where proximity to users allows for collecting more reliable context data. However, many solutions suffer from lack of interoperability, complexity and adoption difficulties. We describe a complete framework for context provision for mobile web applications that is based on technologies that have already been deployed as well as endorsed by standardization bodies. Our framework includes as the main parts a context representation model that is an extension of an ongoing standardization activity within World Wide Web Consortium (W3C) called Delivery Context Interfaces (DCI) targeted to consumer web applications, and a SIP-event based framework for context data provisioning. We further provide a proof-of-concept implementation of individual components along with application scenarios utilizing our proposed framework and our ongoing integration efforts for combining the two main framework parts.

1. Introduction

Adaptive applications are those that adapt based on context data that characterize a usage situation. The advent of mobile information access, coupled with increased capabilities of mobile devices such as cell phones and PDA's adds new dimensions for such adaptability. As such, mobility has opened up new prospects with devices expected to be with users at all times providing reliable cues on user intentions and usage environment. It is becoming imperative that context leverage be seen as part of normal application development driving the need for unified delivery context access methods. Context data can be leveraged for providing content adaptation in order to fit the current device, presentation adaptation (utilizing device capabilities and user preferences) and service adaptation that can happen either on the client device or through a networked service provider. Moreover, devices vary in characteristics such as screen size, keypad type, screen orientations etc. Writing web pages for each configuration is not an option. Adaptation is key and taking into consideration, the dynamic nature of user situations, applications during run time should be able to access system and environment data enabling such processes.

Providing intuitive access methods for context delivery only forms one part of the picture. There is also the need for an infrastructure that can provide for and support such kind of services. The main purpose of any service provider is to provide a clear separation between logically different layers, such as data provisioning and usage. Reusing common functionalities would enable a certain economy of scale rather than implementing every feature from scratch. Through this commonly agreed functionality, interoperability is enabled between services based on the infrastructure. Context information can reside anywhere. Such widespread distribution of information is likely to lead to use cases that will operate in several network domains, spread over different devices while involving several pieces of service logic. Interoperability at various levels, including semantic and communication levels, is a critical issue that can determine if a solution is deployable or not. As a consequence, it is desirable that any solution is as close as possible to existing and emerging standards so that adoption and wide scale deployment becomes feasible.

We describe our ongoing efforts to integrate two frameworks that were developed with respect to context access and provisioning services. Our context access framework uses an extended W3C's Delivery Context Interface (DCI) [4] mechanism, improving on our earlier work [4] with ontology based access control and management of device delivery context access. For the provisioning part, we created a context provisioning architecture [19], together with a middleware specification and implementation, allowing for context provisioning between providers and consumers for the Internet.

The paper is organized as follows. We review some of the related work in this area in Section 2. Section 3

Proceedings of the International Workshop on
System Support for Future Mobile Computing Applications (FUMCA'06)
0-7695-2729-9/06 $20.00 © 2006 IEEE

presents the client side of our context access platform while section 4 describes our context provisioning platform called CREDO. Section 5 outlines our implementation and integration efforts. An adaptive application that uses the context platform is described in section 6. Section 7 concludes the paper and provides a few thoughts on future work direction.

2. Related work

Context-access and adaptation technologies have been getting due attention with platform and services developments conducted by industry and academia. Earlier work by Schilit et al. [8] demonstrated the usage of context aware computing applications on mobile platforms. Biegel et al. [11] describe a framework for context aware application development based on sentient objects in ubiquitous environments. Khedr et al. [23] apply agent-based approaches for building context-aware platform that even spans its reach for context down to the network level. The work done in [13] describes using ontology based context models in intelligent environments. Work done by standardization bodies such as Open Mobile Alliance (OMA), W3C and IETF (Internet Engineering Task Force) has greatly influenced our approach. For delivery context access, there are already limited mechanisms in place. W3C's Cascading Style Sheet (CSS [17]) media queries choose a particular style sheet based on the media type (desktop, PDA etc) that is accessing the web page. Another standard, Synchronized Multimedia Integration Language (SMIL [16]) also offers limited support for checking system characteristics where dynamic values are provided by the runtime environment. OMA's User Agent Profile (UAProf [15]) describes device characteristics using UAProf vocabulary over RDF. WURFL [14] is another resource description mechanism. But, the properties that these profiles describe do not necessarily have to be static. The client device always holds the most updated information and a mechanism is needed for soliciting such information by the adaptation entity. For our context access framework, we rely on W3C's Delivery Context Interfaces (described in Section 3) that provides generic property interfaces as well as extensibility.

For context provision framework, we rely heavily on Session Initiation Protocol (SIP) [20] and SIP-Events [21] work done in IETF. SIP enables separation of user identifier (URI) from endpoint identifier (IP) address, enabling application layer mobility of devices. The most importance extension to the basic SIP framework is *SIP-Events* [21], which uses SIP for creating an event delivery framework for the Internet. The specific semantic of SIP events is not specified in [21]. The semantics are supposed to be defined in separate standardization documents, specifying for instance, behavior of network entities, format of state information and rate limitations for notifications on state changes. SIP-Events 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 [22] (Semantic Markup for Web Services).

3. Context access framework

Figure 1 shows our context access framework using the CREDO [19] distributed context provision architecture. CREDO (also referred to as context provider) is a SIP-event based extensible framework that provides context data grounded within available standards.

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 1) provides the necessary interfaces for context access to consumer applications. The context interfaces would be the DCI API, which is an ongoing standardization effort within W3C.

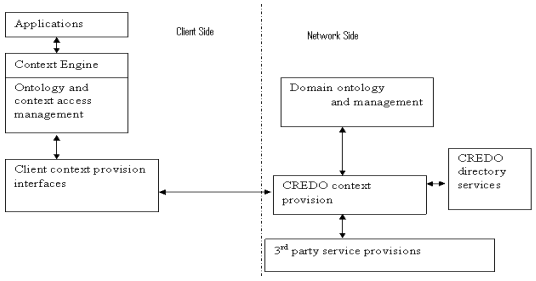


Figure 1. Context access framework using CREDO provision services

The ontology and context access management module is responsible for providing security and integrity of the context representation model. The client context provision interfaces provide the coupling between provider specific interfaces and context representation model. Section 3.1 describes the DCI model and context representation in more detail.

3.1 Delivery Context Interfaces (DCI): accessing static and dynamic properties

The Delivery Context Interfaces [1] is a new approach taken by the Device Independence Working Group (DIWG) of W3C as an access mechanism for device static and dynamic properties. DIWG advocates this approach as a complementary mechanism to their Composite Capability/Preference Profile (CC/PP) [6] model for server side content adaptation and the delivery context approach described in Delivery Context Overview (DCO) [3]. 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. Any instance of the dynamic DCI tree can be serialized using an appropriate CC/PP vocabulary (or other XML representation) for server side content adaptation. But, beyond content adaptation, DCI would be used by applications themselves to gather context data and provide application adaptation through simple access methods thereby reducing reliance on external services for providing the same information.

The W3C's Document Object Model (DOM) [2] is *a platform and language neutral interface that allows programs (scripts) to dynamically access and update content, structure and style of documents*. Scripts can use the DOM model to traverse and manipulate the document. DOM also supports an event system that involves an event propagation mechanism and handlers for listening and capturing events. 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 model among application developers as well as popular browser support. DCI provides an API for property access by extending the standard DOM interfaces while retaining the same event mechanism as DOM. DCI mandates the latest recommendation of DOM level [2] and DOM event [7] specifications.

In DCI, all properties (static and dynamic) are represented as *DCIProperty* nodes in a tree hierarchy where a *DCIComponent* forms the root node. The properties will be grouped under logical categories to form a hierarchical order. New properties are added under specific categories within the tree depending on the property type. DCI extends the DOM Node interfaces with methods for searching and checking for properties. Additional attributes have been defined including specifying metadata related to properties, attribute *propertyType* that can be used to define new property types so that the standard set of DCI interfaces can be extended. Nodes with the same name can be distinguished by their namespace attribute and metadata information. The conceptual structure of a section of the DCI tree is shown in Figure 2.

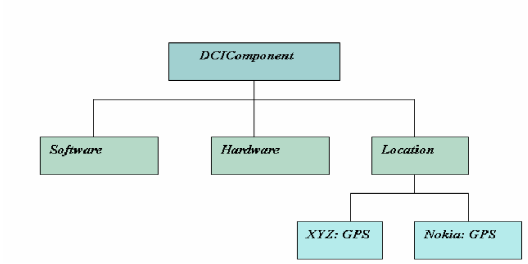


Figure 2. DCI conceptual structure. The location node has two GPS child nodes with two different namespaces.

DCI uses the DOM event [7] model for notification of dynamic value and structural changes. It follows the DOM capture, bubble and target event phases. Applications that implement the DOM *EventListener* interface can add listeners for events at any point in the event path for a target property. All event handlers registered for a particular event and propagation phase are invoked when the event is fired. The Delivery Context Interfaces specification has reached last call working draft status. It is expected to go into candidate recommendation status in second half of 2006.

3.2 Client-side Context Access Architecture

Figure 3 shows the client side context access architecture.

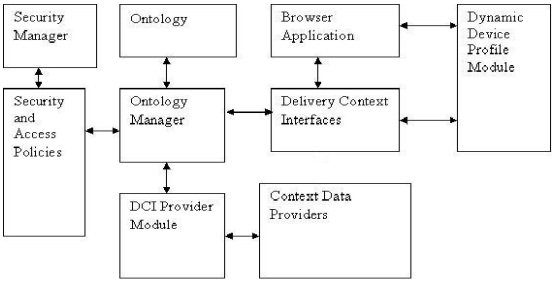


Figure 3. Client side context access architecture using ontology based mechanism for access and integrity management.

An ontology describes concepts used in a particular domain that are machine understandable along with relations among the concepts used. The architecture 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 etc) and the set of properties that would fit under each set. The ontology would be extensible in that the device manufacturer or management authority can extend the vocabulary based on new properties that would emerge.

Figure 3 is an architecture extension of client side context access framework shown in figure 1. The context access mechanism to consumer application is the DCI component in figure 3. The ontology and context access management module are split into ontology manager, ontology, security and access policies and security manager. The DCI Provider module provides a generic interface that can be linked to any data providers that may be locally resident or remote. The security and access policy module provides 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 who gets what type of access to where within the DCI tree. The ontology manager could also provide similar controls required for management of the ontology. The full functionality of security manager, details, format of security and access policies are under development and are outside scope of this paper.

In our proposed architecture, the context data providers seek access to the DCI tree through the DCI provider interface. The DCI provider interface (module) takes the property metadata (such as OWL-S 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, given the proper rights, the ontology manager checks the ontology itself 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 (with parent information etc.). The node pointer is then passed to the DCI provider interface which is then passed 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 when a consumer asks for that property. When a consumer attaches an event handler (the first event handler to that property), a subscription request is sent depending on the protocol supported by the provider. The property (that the provider provides) though, would have a node in the DCI tree with the metadata interface describing the services that can be subscribed to by the consumer. The frequency of data provided would depend on the type of property and mechanisms for controlling the dynamic nature of data are yet to be developed. The DCI provider module exposes the DCI provider API that has been internally developed within Nokia. 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. 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
- Set namespace prefixes for XPath [18] usage

The provider API supports usage of XPath expressions for addressing nodes in the DCI tree. XPath (XML Path Language) describes expressions that can be used to address parts of an XML document. It describes a way to address, locate, and process elements (including attributes, processing instructions etc). XPath uses an addressing syntax based on a logical path through the document hierarchy treated as a logically ordered 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 dynamic device profile module, shown in figure 3 supports an API at the client side for generating profiles showing client side capabilities. The dynamic profile is a snapshot of DCI tree at any point during a session serialized in an appropriate format that can aid server side content adaptation. To support this, a client side API has been developed that

can be used by calling applications for generating the profile. More information can be found in [4].

4. CREDO distributed context provisioning architecture

Figure 4 shows the high-level view of the CREDO distributed context provisioning system architecture. As can be seen, each distributed element of our architecture 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.

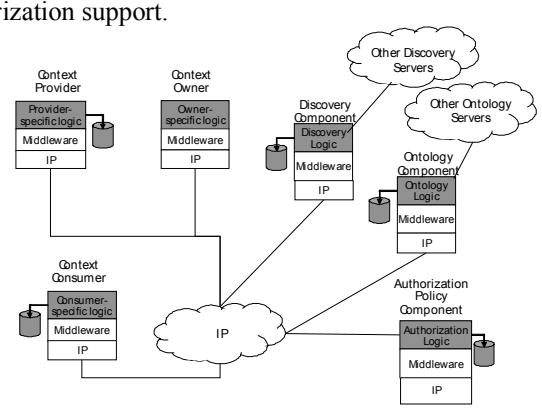


Figure 4. Context provisioning system architecture

Within each element in our architecture, *component-specific functionality* should be implemented in addition to the common middleware, shown as gray boxes in figure 4. Note that this element-specific functionality is outside the scope of the platform and can implement proprietary and differentiating functionality, such as reasoning. This logic, however, will use the common middleware for implementing its functionality, wherever possible. In the following, we briefly describe these components.

4.1 Context consumers, providers and owners

The main components of our 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 be simply 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 1, 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.

4.2 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 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.

4.3 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 own context ontologies.

4.4 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, as will also be discussed later, in the current implementation, 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 insure proper access to existing ontologies, plus other operations with ontologies that a certain consumer might require. Note 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 (storage, verification, merging, mapping, etc.), proper format adjustment of the ontology, and reasoning logic for selecting an appropriate ontology can be envisioned.

5. Proof of concept

This section briefly discusses our implementation efforts with DCI and CREDO.

5.1 DCI Implementation

A near full implementation of the DCI specification has been done. Our implementation provides a DCI extension to Mozilla Firefox browser. The implementation platform was Linux RedHat9 using C++. A direct integration was done using Firefox browser's extension mechanisms. A mock taxonomy for a first level hierarchy of property vocabulary including Software, Hardware and Location were created. A GPS property node was added under location node where a simulated value was provided that was updated every 300 milliseconds. Our ongoing integration work with CREDO platform is aimed at replacing simulated data with real data. Multiple streams of context data are expected that would be utilized by the application providing richer interactions. The DCI implementation was also ported to a Nokia 770 Linux tablet with the Manaos Browser [24] (Firefox for Maemo platform) for integration with CREDO framework. For reasons of space limitations, we refer to [4] for more detailed implementation information

5.2 CREDO Implementation

As key element in our proof-of-concept, we implemented the context provisioning platform middleware, shown Figure 4 as the common entity. It provides commonly used functionality across all components. Three major parts exist in the middleware, as shown in Figure 5, responsible for context provider discovery, context information provisioning and general provisioning of information within an event delivery paradigm. Each of the parts has been fully specified with dedicated interfaces. Further, the information within our system is modeled, based on our ontology-based design principle. Communication with the DCI module happens through the middleware API that interacts with the DCI provider API.

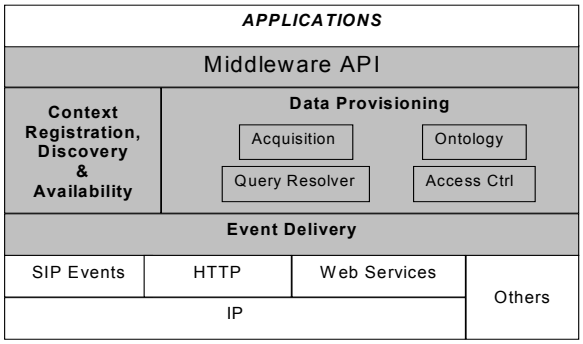


Figure 5. Middleware of the CREDO distributed context provisioning platform

For reasons of space limitations, we refer to [19] for more CREDO implementation information.

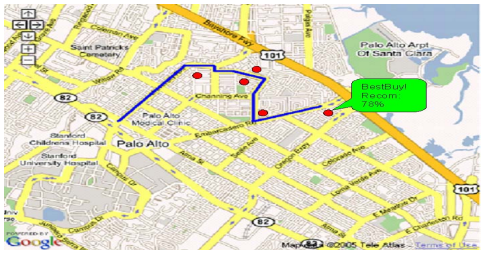
5.3 Integrating DCI and CREDO

The integration of DCI and CREDO is currently ongoing. 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 extend 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) as well as where the provider can also push 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 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) is ongoing 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 of catering for 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. We are also planning to develop additional capability for metadata interface that can accommodate application supplied initialization data that would be applicable for a particular session.

6. Applications

The DCI-based context aware logic was first demonstrated with a Google map mash-up application running on a DCI enabled browser. The application used JavaScript calls to attach event handlers to the DCI tree that listened for a "***dci_prop_change***" event for property value changes at the GPS node. The handler was invoked during coordinate changes which were then plotted as polylines (between the coordinates) on the Google map. Our new approach complements with the CREDO platform providing location, user activity and preference data.



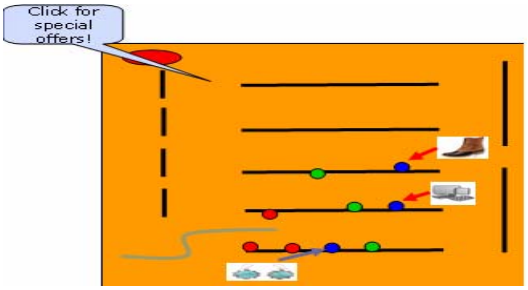


Figure 6. Adaptive map application. The map highlights user position and changes according to user context.

The scenario is shown in Figure 6. Based on time information and user activity (such as leaving the office, getting into car), shopping is highlighted as the next "to-do" activity. As the user start 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)). 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.

7. Conclusion

A context service framework for mobile web has been described. Our approach relies on existing and ongoing standardization activities as well as relevant industry practices in the realm of mobile web. For context consumers, we use an extended framework based on W3C's Delivery Context Interfaces providing additional features such as ontology based access management, serialization for server side adaptation and provider interfaces. For context data and service provisioning part, we use a SIP-event based framework as a data delivery mechanism coupled with ontology based mechanism for semantic interoperability of services.

Our future plans include completion of ongoing integration between DCI and CREDO frameworks, investigating uses of integrated and distributed

ontology for consumers and providers, access policies and control through management objects as well as improvements to the overall framework. We will be looking at new dynamic context sources, levels of abstractions and new adaptive applications that promise value to the user.

Acknowledgements

Sailesh Sathish thanks fellow editors of the DCI specification and members of Device Independence Working Group for their contributions and support for this work. We thank colleagues at Nokia for their implementation support and invaluable insights that made this paper possible.

8. References

[1] K.Waters, S. Sathish, R.Hosn, D.Ragett and M. Womer, " Delivery Context: Interface Accessing Static and Dynamic Properties", W3C last call working draft, November 2005, available at http://www.w3.org/TR/2005/WD-DPF-20051111/

[2] World Wide Web Consortium, "Document Object Model Technical Report:, available at http://www.w3.org/DOM/DOMTR

[3] R.Gimpson, R. Lewis, S. Sathish, "Delivery Context Overview", W3C note, available at http://www.w3.org/2001/di/Group/di-dco/di-dco-draft-20060111/

[4] S. Sathish, O. Pettay "Delivery Context Access for Mobile Browsing", Proceedings of IARIA, Bucharest, 2006

[5] D. Brickley, R.V. Guha. "Resource Description Format 1.0: RDF Schema", W3C recommendation, February 2004., available at http://www.w3.org/TR/rdf-schema/

[6] G. Klyne et al. "Composite Capability/Preference Profiles: Structure and Vocabularies 1.0", W3C recommendation, January 2004, available at http://www.w3.org/TR/CCPP-struct-vocab/

[7] T. Pixley, "Document Object Model Level 2 Events Specification", W3C recommendation, November 2000, available at http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113/

[8] Schilit, B., Adams, N., Want, R., "Context aware computing applications", IEEE workshop on mobile computing systems and applications, Dec 8-9, 1994

[9] Roy, M. "A model of explicit context representation and use for intelligent agents", *Modeling and using context: second international and interdisciplinary conference*, CONTEXT'99, Trento, Italy 1999

[10] A. Dey and G. Abowd, "Towards a better understanding of context and context-awareness", VU Technical Report GIT-GVU-99-22.1999

[11] G.Biegel, V.Cahill, "A framework for developing mobile, context-aware applications" in Proc, Second *IEEE Annual Conference on Pervasive Computing and Communications*, PERCOM, 2004

[12] A, Held, S. Buchholz and A Schill, "Modeling of context information for pervasive computing applications", Proc of 6th *World multiconference on systemics, cybernetics and informatics (SCI*), Orlando, FL, July 2002

[13] T. Gu et al., "An ontology-based context model in intelligent environments", Proc, *Communication Networks and Distributed Systems Modeling and Simulation Conf., Soc, for modeling and simulation intl's*, 2004

[14] L. Passini, A. Trassati, "Wireless Universal Resource File (WURFL)", available at http://wurfl.sourceforge.net/

[15] WAP User Agent Specification, available at http://www.wapforum.org/what/technical.htm

[16] D. Bulterman et al. "Synchronized Multimedia Integration Language (SMIL2.1)", W3C recommendation, December 2005, available at http://www.w3.org/TR/2005/REC-SMIL2-20051213/

[17] B. Bos et al. "Cascading Style Sheets (CSS 2.1)", W3C working note, available at http://www.w3.org/TR/CSS21/

[18] J. Clark et al. "XML Path Language (XPath 1.0)", W3C recommendation, November 1999, available at http://www.w3.org/TR/xpath

[19] D. Pavel, D. Trossen, "Context Provisioning in Future Service Environments", Proceedings of IARIA, Bucharest, 2006

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

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

[22] D. Martin, O. Lassila et al."OWL-S Semantic markup for web services" W3C member submission, available at http://www.w3.org/Submission/OWL-S/

[23] M. Khedr, A. Karmouch., "ACAI: Agent-based context-aware infrastructure for spontaneous applications", Journal of Network and Computer Applications, pages 19-44, 2005.

[24] Manaos Internet Browser: Available at http://extindt01.indt.org/10le/manaos/screenshots.html