mCAR: Software Framework Architecture for In-vehicle Pervasive Multimedia Services

Kamal Kumar Sharma, Hemant Sharma and A. K. Ramani

***Abstract*— The pervasive computing systems inside modern day automobiles are made up of hundreds of interconnected, often replaceable software components. In-vehicle multimedia components and applications are becoming complex artifacts due to advancement in technology and increased competition. There is a growing need for software platform to enable efficient deployment of multimedia services in automotive environment.**

This paper presents a software framework, *mCAR*, to support component based development of pervasive multimedia applications. The framework is scalable and configurable. The framework is aimed to run on an In-vehicle infotainment platform to support efficient management of available multimedia resources.

***Index Terms*— Context Aware, Pervasive Systems, In Vehicle Multimedia.**

I. INTRODUCTION

Many modern automobiles contain hundreds of embedded microcontrollers [1] interconnected via vehicle network. The automobile industry has seen a shift towards the use of more on-board technology and, is becoming increasingly software-dependant. From sophisticated navigation systems to computer-controlled driver assistance safety systems and in-car multimedia and entertainment, the amount of software written for cars is increasing rapidly.

Though embedding multiple software components is more cost-effective and facilitates more reuse than designing a central control software system, there is an associated cost in additional software complexity. Many components in these automobiles are designed to be replaceable to ease future maintenance and service extension of the vehicle. This means that a new component will often have a different feature-set to the component it replaces. Separate components need to be able to work together despite not always being aware of each other's capabilities. It is also likely that this modularity will give rise to a market for cheaper non-OEM components.

Kamal Kumar Sharma is Ph. D. student with Devi Ahilya University, Indore, INDIA. (e-mail: kamal.sharma74@rediffmail).

Hemant Sharma is Software Architect at Delphi Delco Electronics Europe GmbH, Bad Salzdetfurth, Germany. (e-mail: hemant.sharma @ delphi.com).

A. K. Ramani, is Professor at School of Computer Science, Devi Ahilya University, Indore, INDIA. (e-mail: ramani.scs@dauniv.ac.in).

Further, the proliferation of multimedia capable mobile devices such as novel multimedia enabled cellular phones has encouraged users inside a vehicle to consume multimedia content and services while on move. However, selecting and enabling the presentation of the most appropriate content for the given device is rather complicated due to the vast amount of multimedia data available and the heterogeneity of systems available. Appropriate software infrastructure is required and being developed to enable automatic discovery and efficient presentation of multimedia content. Context information can be used to guide the applications and systems in selecting the proper content and format for a given user (passenger or driver) at a certain time, place and under a specific context.

Recent advances in computer and Internet technology in the past decade have greatly shaped the evolution of in-vehicle ubiquitous multimedia environment. On-board infotainment system is able to access multimedia content from a CD changer, iPod or Bluetooth enabled Cellular Phone. However, there are still many challenges in this type of environment. One basic problem of this multimedia service model is the dynamic heterogeneity caused by the physical attributes of diversified computing devices and vehicle networks. Device and user mobility, another important problem of in-vehicle ubiquitous computing, posses a great challenge to the management of multimedia services.

The key contribution of this paper is the description of framework architecture for pervasive multimedia services that addresses the challenges mentioned above. It includes mechanisms to access multimedia content as part of contextual information. It is composed of core service components and augmented by several enhanced components that comprise a distributed service enabler space.

The rest of the paper is organized as follows: Section 2 provides an overview of related research. Section 3 outlines the design concepts for the framework. In section 4, the software architecture of the framework has been described. Section 5 introduces application model for the framework. Section 6 concludes the paper.

II. BACKGROUND

Driving is a complex process which requires adaptable interactions between three components: the driver, the car and the environment. Driving is successful or safe when the interactions between these three components are not faulty. Relevant information required to perform the main driving task are adapted (to the situation) and exchanged correctly, without loss between the three components.

Developing context-aware applications continues to present software engineering challenges. Well known solutions that assist with acquiring and processing context information from sensors have been proposed by Dey et al. [2], Schmidt et al. [3] and Chen and Kotz [4]. In addition, a variety of context management systems that maintain repositories of context information and provide sophisticated query facilities to context-aware applications have also recently appeared [5, 6]. These solutions focus on removing complex functionality from the applications and placing it within shared infrastructure, but do not place much emphasis on providing context modelling techniques that are natural to use for the application developer.

Several novel developments are related to context-awareness in general. Regarding context-aware multimedia systems and services, recent trends consider mostly the location and time as part of the context. Contextual information can be weaved into the process of Authoring, Annotation, Retrieval, Adaptation, Personalization and Delivery of Multimedia Content. As an example, location can be used as context information to deliver most appropriate multimedia information (such as a video, a multimedia presentation or a commercial) related to the current location of the user. Examples are location and context aware museums or tour guides such as [8], [9]. Location and time can also be used to annotate multimedia content or clips during their creation. Contextual information can then be used to tag and store the data as well as being used during the retrieval process. Several projects such as Meaning [10], Context Watcher [11], or Lifeblog [12] automatically associate such contextual metadata with media captured on camera phones. When it comes to presentation of the multimedia content, device characteristics and bandwidth information can be used as contextual information to adapt the multimedia content to the device and network capabilities, as e.g. done within the MPEG-21 framework and contextual extensions [13]. Regarding more interactive multimedia services such as IPTV or video on demand (VoD), contextual information can be used to personalize the content and much work needs to be done to correctly estimate the context for proper personalization in order to enrich the streamed content by adding more information (such as personalized advertisements) or adapting the presented information taking into account user context [14]. Finally, location information is also used as context source in pervasive multimedia games. Here, context service management models and components are used to manage game related events and delivery of related multimedia data [15]. However, we are not aware of any integrated platform that aims to deliver pervasive multimedia services in automotive environment.

III. Pervasive In-vehicle Multimedia

A. Overview

Innovative key factor for improving applications and services, comfort and safety, and vehicle management are represented by pervasive networked technologies in the automotive sector. This networking enables:

- Innovation of in-vehicle multimedia system and software architectures.
- Enhancement of the application and services offered to end users.

Fig.1 presents typical in-vehicle pervasive network along with the physical components and devices that could part of the network.

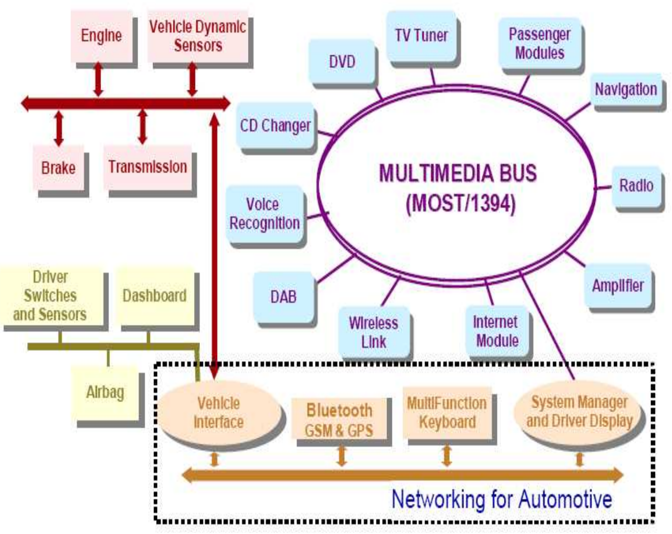

Figure 1: Pervasive Vehicle Network with Multimedia Devices.

The network contains a pure multimedia part, represented by *Multimedia Bus*, and a control and external communication part, represented by *Networking for Automotive*. Multimedia Bus represents the wired multimedia resources. External multimedia resources can form an ad hoc network via the communication interfaces of the vehicle network.

In order to exploit the services, provided by available multimedia resources, a software framework is necessary. This framework must follow the design concepts outlined in the next sub section.

B. *Design Concepts*

In order to address challenges described in section 1, the software architecture presented here incorporates the reflective techniques into its design, in a sense that the system can reason about and modify itself on a meta-object level. The architecture considers the following design concepts:

- *Open*: the term has two-fold meaning in the context of this system. First of all, the internal semantics of the system objects are exposed to the applications on top. Secondly, the system is open to the modification or replacement of its constructing components.
- *Component-based*: the system is decoupled into different sets of components, each having individual functionalities. As the building blocks of the whole architecture, they can be loaded or removed dynamically.
- *Active*: the system environment updates and application behavior changes, and has certain self administrative power to accomplish the system functionality.
- User-centric: we believe that user, not the application, will be the ultimate client of pervasive multimedia service delivery.

In the next section, we present the architecture of the multimedia framework based on these design concepts. We name the framework as *mCAR*.

IV. ARCHITECTURAL OVERVIEW

This section presents a high level overview of *mCAR*, the context aware multimedia framework architecture, describing briefly some of the design decisions taken for each part. The framework is divided in three main modules; the *mCAR* Kernel, the *Communication Subsystem* (CS) and *the Content Module* (CM). The user of *mCAR* provides a set of decision modules and multimedia protocol software to resolve the communication with the vehicle infrastructure.

The Kernel defines a set of software interfaces that are used by the decision modules for accessing both the CS and the CM. This module also provides a set of mechanisms for handling concurrent access to the data. The Kernel initializes the server and the decision modules. To maintain low coupling between the Framework and the decision modules, an extensible event system has been developed. This low coupling approach allows dynamically adding and removing listeners without changing the predefined internal behavior of the Framework.

The decision modules and their dependencies are specified in a configuration file that contains the id, the class that implements the decision module and their execution dependencies. This approach allows a declarative way for specifying the logic components used by the system. The execution dependencies are embedded as a list of decision modules that run before the current one is executed. This defines a direct acyclic graph of executing dependencies between all the decision modules. A topologic sort over this graph provides the execution order for each module. As previously stated, the modules are executed when an event occurs, but it is also possible that the decision module runs in parallel to the framework. The decision modules are loaded in runtime, allowing the system to replace or modify certain logic without stopping the execution of the multimedia application. The architecture and components that integrate a multimedia service implemented in the framework are presented in Fig. 2.

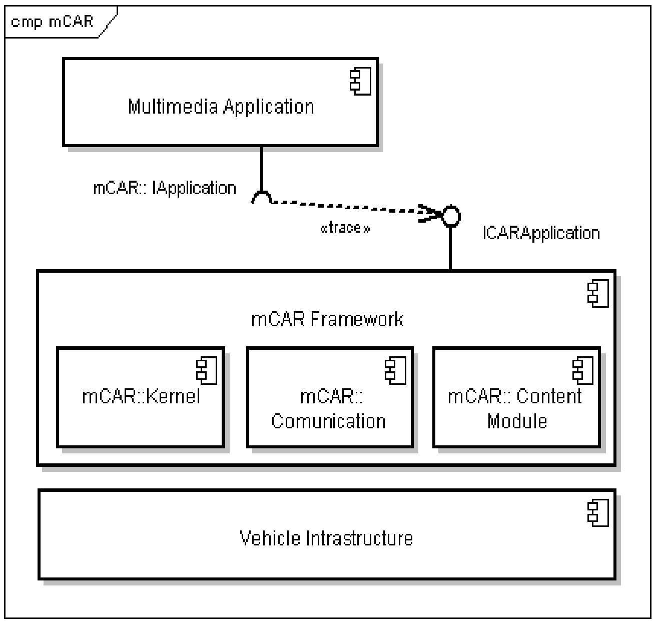

Figure 2: Framework Architecture

The *Communication System* implements asynchronous communication for the framework. It abstracts the network protocol and the network link (wired or wireless). Thus, the user of the framework defines the communication protocol used at the application level. Applications open a data connection with the framework and for each new connection established the CS creates a content worker thread for handling the communication with the specific multimedia resource. For sending data to a resource, each worker contains a local queue and the worker sends data messages stored in this queue to the specific resource. The worker stores data received from the resource in a general queue that is part of the CS.

The information of this queue is processed by certain number of dispatcher threads, each dispatcher pass the data message to the Multimedia Protocol component. The Multimedia Protocol component is loaded at the startup of Framework. This component processes the incoming messages of the resources and may propagate events to the decision modules (or other listeners) each time a new message arrives. Different types of events can be defined and propagated by the user through this component. The user application can provide different implementations of the *Multimedia Protocol* (only one at a time is used). The latter allows different application-level protocols to operate with the Framework [2] [3]. The internal composition of Communication System is presented in Fig. 3.

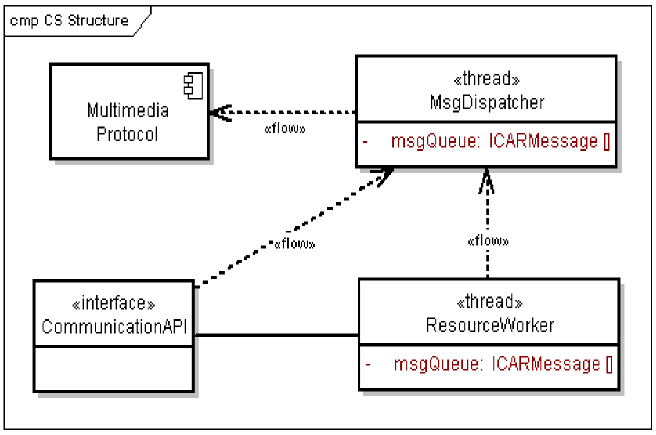

Figure 3: Communication System Structure

The proposed queue system, which is based on a producer-consumer model, makes possible to achieve certain level of asynchronous application - resource communication. It is also possible to distribute the work of the dispatchers and the workers under several resources, improving the scalability of the system.

V. Interface Specification

This section will introduce the class and interface designs of the Event System, Decision Modules and the Multimedia Protocol. The interfaces are described using UML [7]. The user of the framework provides some of these interfaces to implement different system behaviors. The framework is currently implemented in C++.

A. *Event System*

The event system used in the framework is defined using three interfaces, the *ICAREventProvider*, *ICAREventListener* and *ICAREvent*, the UML class diagram for these interfaces is presented in Fig. 4.

ICAREventListener instances are registered to listen events from a specific event source (*ICAREventProvider* instances). Each *ICAREventListener* instance provides the list of dependencies that should be executed before itself, defining a partial execution order for all the listeners. *ICAREventProvider* instances are active components that propagate certain *mCAR* or user defined events (*ICAREvent* instances). Event source instances can be linked in a chain of event propagation, using the appropriate method of *ICAREventProvider*. When an event is propagated, the following protocol steps should be respected by each event source:

1. Executes default action event for the current event source.
2. Executes the *listenEvent* method for all registered listeners of the current event source, respecting the predefined order.
3. Propagates the event to the parent of the current event source (repeat the process from step 1).

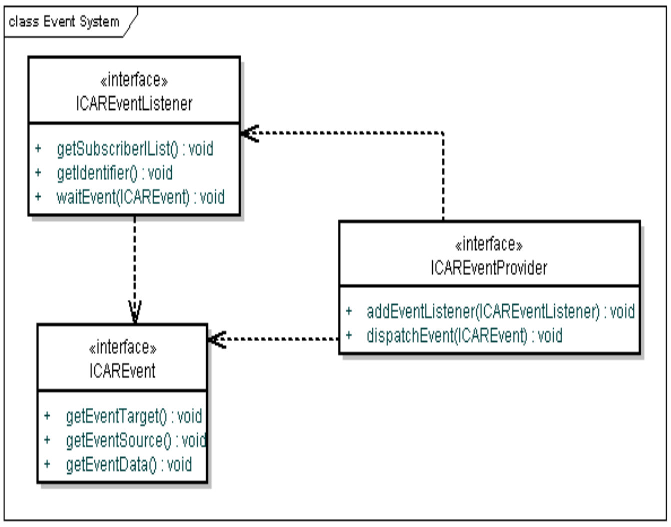
class Event System
«interface» ICAREventListener
+ getSubscriberlList() : void
+ getIdentifier() : void
+ waitEvent(ICAREvent) : void
«interface» ICAREventProvider
+ addEventListener(ICAREventListener) : void
+ dispatchEvent(ICAREvent) : void
«interface» ICAREvent
+ getEventTarget() : void
+ getEventSource() : void
+ getEventData() : void

Figure 4: Event System Interfaces

ICAREvent instances are the event objects that contain the event source target and some data related to the event. The event source target is the logic element of the framework that generates the event, for example, if a vehicle logged in event is propagated then the event source target is the resource identifier.

B. *Decision Modules*

The framework instantiates decision modules using a factory configurable by an XML configuration file. A default implementation for the factory is provided by the framework; this factory loads the XML information, resolves the dependencies and allows the runtime instantiation of each decision module. The UML class diagram for the decision modules is presented in Fig. 5.

Each declared decision module in the XML is an instance of

ICARDecisionModule interface. All the decision modules are initialized with the Kernel interface (API to the framework), including the decision module name and dependencies list (information declared in the XML). The *ICARDecisionModule* instance may run in parallel of the Framework (for example as a proxy to other legacy system), or be activated and executed only under certain events propagated.

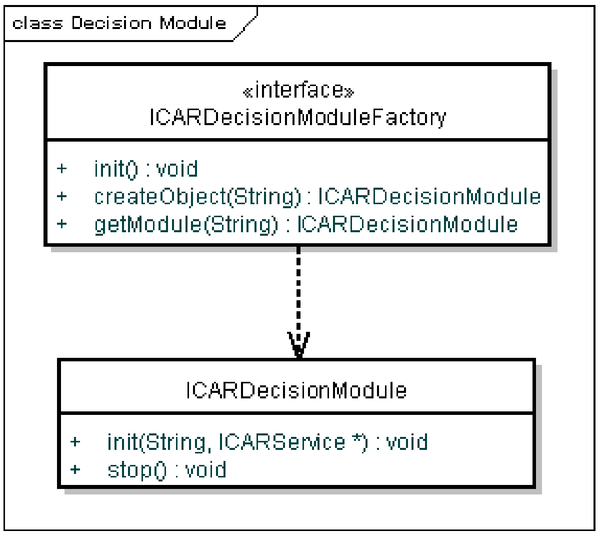
class Decision Module
«interface»
ICARDecisionModuleFactory
+ init() : void
+ createObject(String) : ICARDecisionModule
+ getModule(String) : ICARDecisionModule
ICARDecisionModule
+ init(String, ICARService *) : void
+ stop() : void

Figure 5: Decision Module Interfaces

C. *Multimedia Protocol*

The Multimedia Protocol component processes the communication messages received from the vehicles, propagating the corresponding events in each case. This component is divided in three interfaces; *ICARMessage*, *ICARMessageFactory* and *ICARMultimediaProtocol*.

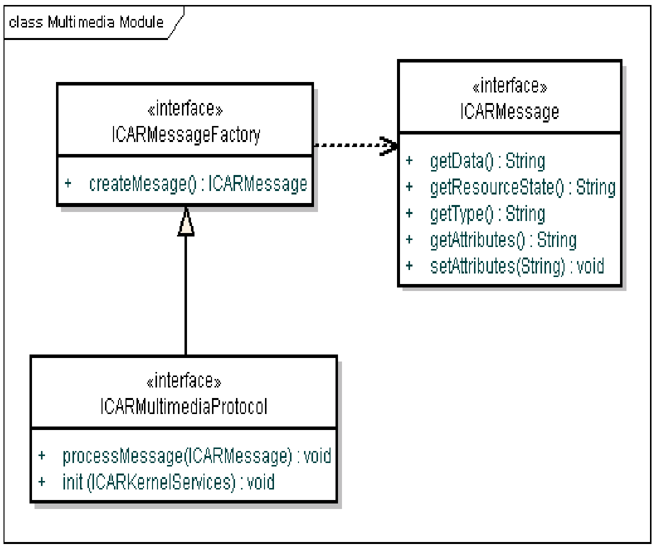
class Multimedia Module
«interface»
ICARMessageFactory
+ createMesage() : ICARMessage
«interface»
ICARMessage
+ getData() : String
+ getResourceState() : String
+ getType() : String
+ getAttributes() : String
+ setAttributes(String) : void
«interface»
ICARMultimediaProtocol
+ processMessage(ICARMessage) : void
+ init (ICARKernelServices) : void

Figure 6: Multimedia Protocol Interfaces

The *ICARMessage* interface represents a message to exchange with the resources. The implementation of this interface provides the methods marshal/unmarshal which receive Input and Output Streams respectively. These methods allows the message to be serialized and deserialized using different formats provided by the application, making transparent to workers and dispatchers, the real stream format exchanged over the network protocol. The *ICARMessageFactory* instance, provided as part of the Multimedia Protocol component by the application, will be the creator of the *ICARMessage* instances. The *ICARMultimediaProtocol* instance is responsible for processing the messages received from the resources, before different events may be propagated. A UML class diagram of the Multimedia Protocol interfaces is presented in Fig. 6.

VI. Application Model

The Application Model of the *mCAR* framework (Fig. 7) provides support for building and running pervasive multimedia applications on top of the framework kernel. The application model shall guide modeling of multimedia application components with the use of interfaces of component of the framework in an efficient manner.

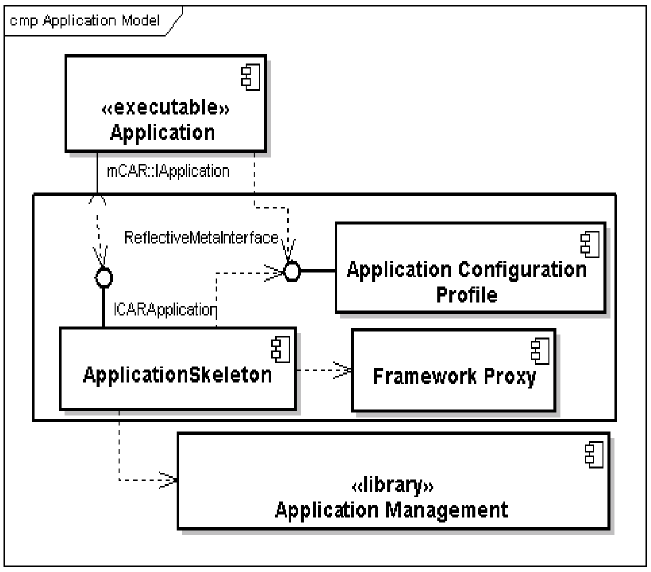

Figure 7: *mCAR* Application Model

The applications access the framework functionality through an *IApplication* interface. Each time an application is started, an *ApplicationSkeleton* is created to allow interactions between the framework and the application itself. In particular, application interfaces allow applications to request services from the underlying framework, and to access their own application configuration profile through a well-defined reflective meta-interface. The multimedia application is realized as composition of components based on the component model of the framework. The description component model is out of scope of this paper.

VII. Conclusion

In the paper, the software architecture of mCAR has been presented. The architecture is based on the design concepts outlined in the paper to meet the multimedia service deployment and management challenges.

This contribution also introduces the application model for development of multimedia applications on the top of mCAR. Applications shall use the provided interfaces of mCAR to use the framework services. The framework behavior is configurable via application specific configuration file. Specification of application specific decision module makes the framework open and scalable to accommodate support for wide range application and multimedia resources.

Component and interface optimization is the first objective of our future work. We anticipate two main directions:

- The use in framework dynamic context formation based on context configuration information.
- Performance analysis of multimedia protocol for different multimedia resources.

References

[1] R. Bannatyne, "Microcontrollers for the Automobile" .Micro Control Journal, 2004.

[2] A. K. Dey, D. Salber, and G. D. Abowd, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications", Human-Computer Interaction, 16(2-4):97–166, 2001.

[3] A. Schmidt, K. A. Aidoo, A. Takaluoma, U. Tuomela, K. Van Laerhoven, and W. Van de Velde, "Advanced interaction in context". In 1st International Symposium on Handheld and Ubiquitous Computing, volume 1707 of Lecture Notes in Computer Science, pages 89–101. Springer, 1999.

[4] G. Chen and D. Kotz., "Context aggregation and dissemination in ubiquitous computing systems". In 4th IEEE Workshop on Mobile Computing Systems and Applications, Callicoon, June 2002.

[5] G. Judd and P. Steenkiste, "Providing contextual information to pervasive computing applications", In 1st IEEE Conference on Pervasive Computing and Communications, pages 133–142, Fort Worth, March 2003.

[6] H. Lei, D. M. Sow, J. S. Davis, G. Banavar, and M. R. Ebling. The design and applications of a context service. ACMSIGMOBILE Mobile Computing and Communications Review, 6(4):45–55, October 2002.

[7] Object Management Group. Unified Modeling Language Specification v.2.0. www.uml.org, September 2003.

[8] Opermann, R., and Specht, M, "A Context-sensitive Nomadic Information System as an Exhibition Guide." Proc. of 2nd Intl. Symposium on Handheld and Ubiquitous Computing, Bristol, 2000.

[9] Keith Cheverst, Nigel Davies, Keith Mitchell and Adrian Friday, "Experiences of Developing and Deploying a Context-Aware Tourist Guide: The GUIDE Project", Proc. 6th annual international conference on Mobile computing and networking, pp. 20-31, 2000

[10] The Meaning Project http://meaning.3xi.org/

[11] Context Watcher Application http://portals.telin.nl/contextwatcher/

[12] Nokia Lifeblog 2.0 http://europe.nokia.com/nokia/

[13] Asadi, M.K. and Dufourd, J.-C , "Context-Aware Semantic Adaptation of Multimedia Presentations", , in Proc. IEEE Multimedia and Expo, Amsterdam, 2005

[14] Amit Thawani, Srividya Gopalan and Sridhar V., "Context Aware Personalized Ad Insertion in an Interactive TV Environment ", TV'04: the 4th Workshop on Personalization in Future TV - Methods, Technologies, Applications for Personalized TV, 2004

[15] David Linner, Fabian Kirsch, Ilja Radusch, Stephan Steglich, "Context-aware Multimedia Provisioning for Pervasive Games", Proc. 7th IEEE International Symposium on Multimedia (ISM'05), pp. 60-68, December 2005.