An Infrastructure for Reactive Information Environments

Rudi Belotti, Corsin Decurtins, Michael Grossniklaus, and Moira C. Norrie

Institute for Information Systems, ETH Zurich, 8092 Zurich, Switzerland
{belotti, decurtins, grossniklaus, norrie}@inf.ethz.ch

Abstract. We introduce the concept of reactive information environments and a general infrastructure for experimentation with such systems. Its asynchronous state-based processing model is described along with the architectural requirements and main components of our infrastructure. These include a general context engine coupled together with a web publishing platform. An application for a public news service is used to motivate the requirements, explain the processing model and show how an application is implemented using the platform.

1 Introduction

Developments in mobile and ubiquitous computing have led to increasing demands for information systems that can adapt to users and their situations. Tools and technologies to support concepts such as multi-channel access, adaptive interfaces, context-awareness and multi-modal interaction abound. Yet it seems that each of these represents only one step along a path leading us away from traditional information system architectures based on the simple request-response model of processing towards a notion of *reactive information environments* that react to many forms of stimuli and deliver information in various formats and places. Processing will no longer be based purely on the synchronous request-response model, but rather on asynchronous state-based models.

We present the architectural requirements of such systems and a general infrastructure developed for experimentation with reactive information environments. To motivate the requirements of such a system and explain its operation, we use an example of a community news and awareness service that adapts the set of articles on public displays to the group of users currently present in the environment. Since the users do not interact with the system in a command-based way, the system itself has to decide which tasks to execute at what time and the most appropriate information to be displayed. It needs to keep track of the environment and the users within it, as well as news updates. A general context engine coupled with a content publishing system is responsible for delivering the right information to one or more channels as and when required.

We begin in Sect. 2 with a look at how this notion of reactive information environments has evolved and relates to various developments in information system technologies. In Sect. 3, we present the architecture and main components of the system. Sect. 4 describes the context engine and Sect. 5 content

publication. Details of the control flow of the entire system are presented in Sect. 6. Concluding remarks are given in Sect. 7.

2 Background

Since the outset of information systems, the demands made on these systems have changed constantly. The requirement for structured data once led from file-based storage to the first hierarchical databases and early path-based query languages. Demands for more flexibility and performance resulted in the relational model together with its well-defined algebra and the necessity for more advanced data models gave rise to the entity-relationship (ER) model and its many variants. Although specialist application domains such as Computer-Aided Design (CAD) have motivated the development of object-oriented database management systems (OODBMS), relational databases are still considered the state-of-the-art in contemporary information systems.

In the midst of this extensive evolution, however, at least one aspect has stayed the same. Interaction with a database has until recently been based purely on the paradigm of request and response, or, client and server, respectively. With the advent of the World Wide Web, new requirements needed to be considered that also affected the manner in which clients interacted with databases. Early approaches of supporting web-based access to databases were witnessed in the OODBMS O_2 [1]. Other reactions to this new requirement of Web access were also found in well-known commercial systems such as Oracle [2] with the first version of Oracle Web Server released in 1995. Although the importance of the web for information systems was accepted early on, the Internet has continued to spawn new demands ever since.

Probably the most important of these demands has been the requirement to manage ever larger web-sites within a database. Several research projects have acknowledged this requirement and provided valuable solutions to it. The Strudel [3] project approached the problem with StruQL a query language that allows the declarative specification of web-sites. Building on database modelling techniques, the WebML [4] project was among the first to extend traditional data models for the creation of data-intensive web-sites. Their long lasting endeavour in the field of model-based approaches has led to a collection of models and tools that together form a CASE suite for web engineering. The Hera [5] project also takes a model-driven approach. However, the Hera approach is not based on a traditional data model but rather on technologies and standards that have been developed by the World Wide Web Consortium (W3C), such as XML, RDF [6] and OWL [7]. Finally, projects such as UML-based Web Engineering (UWE) [8] and W2000 [9] introduced powerful modelling languages based on UML that allow the specification of both the data and behaviour of a web information system. In the meantime, industry has also been aware of the challenges and lived up to them by developing a multitude of content management systems.

While most of these approaches focused on web-sites represented in HTML, the Internet has moved on to become a platform that uses several presentation

channels to deliver information. Most of the projects mentioned above reacted to that development and at some point integrated support for multi-channel access. Adapting to different presentation channels is only one form of adaptation and over time other dimensions of adaptation have gained importance and a general notion of context-awareness emerged. This has led to numerous projects seeking to provide a uniform way of expressing the state of a client when issuing a request to a database server and allowing the information system to react to that state. While still retaining the basic architecture of request and response, the response in these systems is influenced by context that may be decoupled from the request. The same query asked at various points in time may return completely different results as the context of the client or the server changes.

Various components, frameworks and toolkits for the acquisition, management and provision of context information have been proposed, e.g. Context Toolkit [10]. An infrastructure that aims more at the provision of an integrated platform is Gaia [11] which uses a virtual machine metaphor and abstracts all kinds of distributed and mobile devices for input, output, storage, processing etc. into a meta-operating system. Applications are based on an extended version of a Model-View-Control (MVC) approach. The components of the application are mobile and mapped dynamically to devices and resources that meet the requirements specified by the application at runtime. In the area of information systems WebML has been extended to support the notion of context [12]. In [13], we describe how context and content information can be integrated to build a highly adaptive content management system.

Ubiquitous computing is another domain that calls for highly context-aware and distributed architectures. Instead of synchronous request and response, there is a demand for systems that autonomously deliver information asynchronously depending on the context of the environment. Research projects such as the Multichannel Adaptive Information Systems (MAIS) project [14] address these new requirements with the development of infrastructures that can adapt to constantly changing requirements, execution contexts and user needs. The MAIS project has produced a number of results in the domains of service registries, personal databases and adaptive user interfaces, to name only a few. We also believe that reactive information environments are the foundation for supporting ubiquitous and mobile applications.

To illustrate the requirements of such systems and the solution that we propose, we will use the example of AwareNews, a context-aware and ambient news visualisation system. The aim of the application is to promote community awareness by integrating community news into a global news service. The news displayed depends on the set of user currently present and their profiles, with the goal of encouraging communication as well as keeping users informed.

3 Architecture

The SOPHIE platform is a research project that aims at the integration and consolidation of the various proposals for adaptive, multi-channel and context-

Fig. 1. SOPHIE Architecture Overview

aware information systems into a generic platform for reactive information environments. We put special emphasis on reactivity and proactivity of the information environment as this is quite a novel concept in the area of web-based information systems. In this section, we present the overall architecture of the SOPHIE platform along with its core components and their integration.

Figure 1 shows an overview of the architecture. On the client side, the system consists of various user interface components, such as web browsers, voice engines or mobile devices. On the server side, the SOPHIE platform integrates a context engine, a content publishing framework and an application database. The context engine is backed up by various sensors (hardware and software) that provide contextual information about the physical and virtual environment. These sensors can be on the client side, for instance mounted on a mobile device, or embedded in the environment, such as for example a light barrier.

The server components have all been implemented using object-oriented database technologies since notions of modularity, extensibility and the integration of behaviour were central to the system. Further, both system and application metadata are represented as objects, enabling a common shared metamodel for the components. The OODBMS comes with a set of predefined models and metamodels to aid application development, e.g. the user model. These can be extended and completed according to the requirements of the application. Since data and metadata are represented objects, information about system concepts such as types and associations can be extended and modified using the same techniques as for user data. In AwareNews, the application database has information about news articles and their classification into categories etc. The predefined user model was extended to include an AwareNews-specific user profile and preferences of the individual users such as their preferred languages and news categories.

The *context engine* is responsible for managing contextual information about all application entities, including the physical and virtual environment of all users

and the location and state of devices that deliver information to them. The information is acquired by *physical sensors* in the environment. For example, in AwareNews, we use physical sensors to detect the presence of people in a certain area of a room and to identify them. The messages on a display are then adapted according to the profiles of the users currently present. In addition to the physical sensors, software components and applications can also serve as *software sensors* providing contextual information about entities in the virtual environment of the application.

The application database and the context engine, together with the OODBMS on which they were implemented, were developed within our research group in previous projects and required only slight modifications to integrate them into the SOPHIE platform. However, the remaining server component, the content publishing framework, was based on a general web-publishing framework that required more significant modifications to adapt it to an asynchronous state-based model from a traditional request-response model.

The content publishing framework includes a content model with information about the publication of application data, as well as a publishing process based around the separation of content, structure, presentation and view. It already supported a lot of the features required for a reactive information environment such as multi-channel delivery and context-awareness. While the core of the original web publishing framework supported a very flexible and generic concept of request and response that enabled it to be used more or less in its original form, as described later in Sect. 5, some parts at the periphery of the system had to be adapted to support reactivity and the proactive delivery of content.

The client components were also greatly affected by the requirements of reactivity and proactivity. Traditionally, desktop web browsers have been used as thin clients for access to information systems. With the evolution towards multi-channel and multi-modal interfaces, other browser components such as voice engines, mobile phones and PDA browsers, and even paper interfaces [15], have been incorporated into information system architectures. However, these components are all based on the traditional request-response model and cannot be used in their current form for a reactive information environment. We therefore propose an additional, integrating component on the client side, the *client controller*. This component acts as an HTTP proxy for the client-side components and serves as a gateway for all of the communication between a client and the server. In addition to the proxy functionality, the client controller also serves as an HTTP server for callbacks from the server, enabling the standard client-server configuration to be reversed and the server side to make requests to the client side, for example, in order to push content to the user.

The actual implementation of the server components was based on OMS, an OODBMS which implements the OM model [16]. Whereas the details of the OM model and OMS system are not important here, we believe that the use of a semantically expressive object data model as the foundation for the infrastructure is important, especially with respect to the integration of the various components and services.

In summary, the architecture of the SOPHIE platform presented in this section provides the basis for a flexible, model-based implementation of reactive information environments. It incorporates components such as the context engine and the content publishing framework which have proven to be very powerful through their generality. It also provides a high degree of flexibility with respect to interactional control flows, supporting not only traditional request-response interaction, but also more reactive patterns such as the pushing of content based on internal events of the application.

4 Context Engine

The context engine provides abstractions and general mechanisms for the acquisition and management of context information which allows context information to be presented to the other components of the SOPHIE infrastructure in a uniform and generic way. In contrast to other context frameworks, such as the Context Toolkit [10], the SOPHIE context engine is based on an expressive data model, which structures the context information and provides semantics for the values. Being database-driven, the context engine can work directly with the system's knowledge base. It is composed of different layers that we describe below.

The *context acquisition* layer includes a general definition of sensors that acquire context information from the physical and virtual environment. The information from the sensors is processed by components of this layer and stored in so-called context elements. For example, a physical sensors can be used to recognise persons in an AwareNews room. When a new user is detected, a software sensor can decide on the language to be used to present information on public displays based on their preferred language or a common language if more than one users is present.

In the next layer, *context augmentation*, existing context elements are associated with application objects that are concerned with the current context. The context elements can also be augmented through accumulation or combination with other data sources by means of software sensors. Examples are the computation of average values, as in the case of temperatures (both over a history of values or values from multiple sensors) or the mapping of user identifiers such as RFID tags or identification badges [17] to actual user objects stored in a user database. This layer is responsible for the conversion of the data coming from the sensors to valuable information that is related to existing objects.

Context-aware applications can query the context model and thus have access to all of the context elements. Another possibility for accessing the context model is the *event notification* layer. An application can register for certain events on specific context elements. For example, the publishing platform of AwareNews can register for changes to language context in order to present the news in the best language for the users currently viewing the public display. The context engine will send an event to the publishing platform if this context element changes. The application entities that want to be notified of an event can provide a notification endpoint, for example, in the form of a web service.

Fig. 2. Context Model

Figure 2 depicts the relationships between the components of the context engine. Each component of the context engine is typed. The type definitions provide a general abstraction of the component and provide parameters that allow the instantiation of concrete components. For example, you can define a sensor type describing a class of person-recogniser sensors connected through the USB port. In this case, the USB port represents the parameter required at the time of definition of the real sensor. In order to connect a new sensor that recognises the current users in the room, you need to instantiate a new sensor object of the corresponding type by passing the appropriate parameters as the ID of the USB port to which the person recogniser is connected.

```

contextType ctx_users {
    users: set of user;
};

context users_in_lab: ctx_users describes awarenews:student_lab;
sensorDriver user_sensor(usb_port: integer): ctx_users;
sensor user_recogniser: user_sensor(3) provides users_in_lab;
contextType ctx_lang {
    language: string;
};

context lang: ctx_lang describes awarenews:display;
sensorDriver lang_sensor(users: ctx_users): ctx_lang;
sensor lang_selector: lang_sensor(users_in_lab) provides lang;
  
```

Fig. 3. Context Definition Statements

Figure 3 illustrates the definitions for the example of the person-recogniser and language detector using our Context Definition Language (CDL). This language provides a simple means of defining metadata for the context engine. First we define a *context type* `ctx_users`, which characterises the users currently in an AwareNews room. This context type can be used to instantiate a context describing the presence of users in a given room. The **context** declaration links the set of users to an application concept, here the room `awarenews:student_lab`. Finally,

sensors need to be defined to acquire the identity of the users. A sensor definition comprises both a declaration of the sensor type and an instantiation of a sensor. In our example, with the `sensorDriver` statement, we define an abstract class of user sensors that can be connected to arbitrary USB ports. Then, with the `sensor` statement, we create a concrete person-recogniser that is plugged into USB port 3 and provides data for the context element `users_in_lab`. We define another context type `ctx_lang` which characterises the AwareNews' display properties and instantiate a context `lang` that describes the language used to present the information. A sensor driver is then defined which takes the `users_in_lab` context and extrapolates the appropriate language for the current set of users. Finally, we define the sensor instance that provides the context `lang`. More details on the context engine of SOPHIE and its interworking with applications can be found in [13, 18].

When we design context-aware applications, we have to define clearly the basic application functionality and the context-aware functionality. There is one main difference between these two classes of functionality. The basic application features are always available and completely predictable since they make no use of context information. On the other hand, it is important to notice that context-aware functionality could be restricted if the relevant context information is not available at the time of a request. Moreover, context-aware functionality is dynamically adapted at runtime and could be difficult to predict. This unpredictability could also lead to different responses to the same user's request because of changes in context. Considering the special properties of the context-aware functionality, we must therefore classify and handle it in a special way.

The contribution of the context engine in the field of reactive information environments resides in offering a general model for context information and also a standard component that allows the information system to manage the context and adapt to it accordingly. The adaptation mechanism is implemented in the application. The notification mechanism of the context engine is implemented using traditional database methods and triggers, combined with filters to ensure notification of only relevant changes.

5 Content Publication

Delivering content to the applications built on the SOPHIE platform is the task of the integrated content publishing system displayed in Fig. 1. For SOPHIE, we use our own content management system XCM [19] which is also based on the OM model and thus shares the same semantics as the application database and context engine. The content publishing system provides support to applications requiring features typical of content management such as user management, workflows, personalisation, multi-channel delivery and multi-format objects. At the core of the system is the clear separation of the notions of content, structure, presentation and view.

Fig. 4. Publishing Process

The process of publishing information illustrated in Fig. 4 is defined by the following course of action. First, the system selects the appropriate content requested by the client. As the same content often needs to be delivered in different variations, the system supports “multi-variant objects” [20], which are content objects with multiple context-dependent variants. These content variants are of the same type and are described in the system with a set of characteristics. Each characteristic specifies a content dimension and a value for that dimension e.g. dimension **language** and value **english**. The set of dimensions is specified by the application. Using the information from context engine and request, the content publishing system determines the most appropriate variant by matching this information to the characteristics of the variants.

Personalisation is achieved through the concept of view objects which define the properties of content objects to be published. As with views in relational systems, these can be used to aggregate content semantically linked together by references. The personalised content objects are incorporated into a tree-like hierarchy using the metadata stored in the structure objects and based on a component and container model. A component corresponds to a content object forming a leaf of the structure tree and containers can contain components or other container objects. Finally, the layout stored in presentation objects is applied to the structure tree, thereby producing the final representation. We use XML as a natural representation of the structure tree and hence XSLT as the language used by the presentation objects.

As with content, all other basic concepts of the system—view, structure and presentation—can be annotated and described by characteristics. Hence views, structure and presentation can be customised for different users and devices as well as content. The strength in this approach lies in the fact that our system does not specify what annotations have to be made. It only provides the capabilities to store and manage characteristics and match them according to the current context. By allowing for very fine-grained descriptions of data and metadata, our approach is also able to reduce complexity and redundancy as only the information that really varies from one context to another has to be managed. These requirements are not met by current systems as they mostly focus on multi-lingual aspects only, ignoring other useful dimensions to describe objects completely.

As described in [13], our system uses the context engine not only as a client, but also acts as a software sensor that delivers context information back to the context engine. The content management system is able to provide valuable context information, such as what the user is browsing and how busy they are. Storing this context information, gained from profiling the user's interaction with the system, back into the context component allows other applications that share the same context engine to profit from this information.

Traditionally, content publishing systems react to explicit requests only. Reactive information environments such as the SOPHIE platform break with this tradition and move towards asynchronous interaction. We solved this problem by allowing clients to register with the content publishing system for content that is pushed from the server to the client. As mentioned before, the client controller that manages communication on behalf of the client is itself an HTTP server. Hence, if the context engine triggers the content publishing system in the case of a particular event, it can communicate with all registered client controllers and send updated content to these clients. The existing flexibility of the system meant that few modifications were necessary to move away from the traditional paradigm of request and response and integrate the system as a component in a reactive information environment.

6 Control Flow

Traditionally, interaction with web-based information systems has been based on a simple request-response model at both the level of user interaction and the technical level where interaction is matched directly to the HTTP protocol. The user request corresponds to an HTTP request and the response of the system to an HTTP response. Thus, we can say that interaction is based on a serial dialogue between the user, represented by a client application such as a web browser and the application. This corresponds to a very simple control flow model in which actions are always initiated by the user on the client side and the application on the server side only passively reacts to these actions. The integration of multi-channelling or context-awareness into web information systems has not changed this basic interaction model. Content might be delivered through multiple channels, but the interaction model is still based on request-response. Usually context information is only used passively, i.e. it is used to alter and adapt content and services of the information system but does not initiate any actions in the system. Figure 5 shows a simplified sequence diagram of a request-response based on interaction with a context-aware content publishing system.

Proactivity is an important requirement for ubiquitous and mobile systems. Applications have to be able to initialise an interaction or dialogue and push information to the user solely upon internal conditions of the system and without an explicit request from the user. These internal conditions can be based on application data, publication data or contextual information. For example, in AwareNews, the creation of a breaking news article in the application database can trigger the publication of this article to all users. Contextual information

Fig. 5. Request-response based content publishing

such as the location of a user can also be used to trigger the publication of information. A user entering a room can trigger the display of some information on an ambient screen. Statistical information and access patterns could be used to trigger actions. For example, if a particular story in the AwareNews system is requested frequently within a certain time period, it becomes a “hot story” and will be pushed to interested users. Figure 6 shows the sequence diagram of content publishing initiated by the context engine and pushed to the user interface without any interaction by the user.

The OODBMS provides conventional database triggers on various operations on objects, such as creation, update or deletion. When the trigger is fired, the registered handler can execute arbitrary functionality in the system. Time triggers are also supported and these can be configured to fire at a specific point in time. Both types of triggers can be used to initialise arbitrary operations in the system, such as an interaction with a user. Further the Context Engine also provides the concept of *context events*, as described in Sect. 4. Context events can occur upon modification of context elements in the engine. They are propagated to all notification listeners that have been registered for the specific events. These notification listeners can in turn be used to trigger any functionality of the system based on the contextual information.

On the user interface and interaction level, the request-response model is thus extended to a model where responses can also be sent without a preceding request. An interaction of user and application, represented through client and server, respectively, can be initialised not only by the user, but also by the

Fig. 6. Context-triggered content pushing

application. On the technical level, we still use the request-response based HTTP protocol for communication between the distributed components of the system as we did not want to abandon what has become a defacto standard. However, whereas HTTP request and interaction requests were congruent in traditional web-based information systems, we now have to make a clear distinction between the two. HTTP requests and responses still have to come in pairs, but one interaction response, for instance the pushing of some content to the client, could be mapped to an HTTP request-response pair that originates at the server instead of the client or even multiple HTTP request-response pairs.

For the pushing of information to the client, we have can use the following technique. The client, represented by the gateway component mentioned in Sect. 3, also acts as an HTTP server and provides a callback for this mechanism. The exact address of the callback is managed by the server side as contextual information. If the server side now wants to push content to the client side, it compiles a URL that can later be used by the client to request this content. In the case of the breaking news example, this might be a simple article request URL with the particular identifier of the news article. The server now makes an HTTP request to the callback that corresponds to the user who is supposed to receive the content. This information can be looked up in the context engine. The content URL that was compiled previously is passed as a parameter of the HTTP request. The client replies to the HTTP request with an empty response and, as a side-effect, makes an HTTP request for the URL that it has just received from the server and receives the correct response with the breaking

news article content. For obvious reasons, this callback has to be protected by state-of-the-art authentication and authorisation mechanisms.

The advantage of this approach is that it requires minimal changes to an existing web-based information system with a traditional request-response approach. The system only has to be extended with the callback mechanism on the client side as well as the content publishing trigger on the server side. The disadvantage is that we need two HTTP calls to publish the content. Alternatively, the server can also send the content for the client directly in the first HTTP request. The client again replies with an empty HTTP response, but the second HTTP request from the client to the server is unnecessary. As a side-effect to the callback, the client does not have to initiate a new HTTP request, but can instead present the content directly to the user.

7 Conclusions

We have presented a platform that supports the development of information systems that can react to various forms of events in the user environment and control a variety of information channels. The platform was implemented through the integration of existing components for context-awareness and content publishing and we describe how these can be coupled with an application database and client controllers. We believe that this work represents an important step towards the development of general infrastructures for the support of ubiquitous and mobile information systems.

In contrast to a number of projects with similar aims in the field of ubiquitous computing, we adopted a metamodel-driven approach to the development of our system. As a result, our system is more flexible than other frameworks in terms of generality and extensibility and also has greater support for component reuse. In addition, it ensures greater semantic consistency due to the underlying semantic metamodels.

References

1. Bancilhon, F., Delobel, C., Kanellakis, P.C., eds.: Building an Object-Oriented Database System, The Story of O2. Morgan Kaufmann (1992)
2. Oracle Corporation: Company Web Site (1999-2005) (<http://www.oracle.com/>).
3. Fernandez, M.F., Florescu, D., Kang, J., Levy, A.Y., Suciu, D.: STRUDEL: A web-site management system. In: Proc. Intl. ACM SIGMOD Conference on Management of Data, Tucson, Arizona, USA (1997) 549–552
4. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann (2002)
5. Houben, G.J., Barna, P., Frasincar, F., Vdovjak, R.: Hera: Development of semantic web information systems. In: Proc. Intl. Conference on Web Engineering (ICWE), Oviedo, Spain (2003) 529–538
6. World Wide Web Consortium (W3C): Resource Description Framework (RDF) (2005) (<http://www.w3.org/RDF/>).

7. World Wide Web Consortium (W3C): Web Ontology Language (OWL) (2005) (<http://www.w3.org/2004/OWL/>).
8. Koch, N., Kraus, A.: The expressive power of UML-based web engineering. In: Proc. Intl. Workshop on Web-oriented Software Technology (IWWOST), Málaga, Spain (2002) 105–119
9. Baresi, L., Garzotto, F., Paolini, P.: Extending UML for modeling web applications. In: Proc. Intl. Hawaii Conference on System Sciences (HICSS), Big Island, Hawaii, USA (2001)
10. Dey, A.K., Abowd, G.D., Salber, D.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction* (2002) 97–166
11. Román, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R.H., Nahrstedt, K.: Gaia: A middleware infrastructure for active spaces. *IEEE Pervasive Computing* **1** (2002) 74–83
12. Ceri, S., Daniel, F., Matera, M.: Extending WebML for modeling multi-channel context-aware web applications. In: Proceedings of MMIS'2003, International Workshop on Multichannel and Mobile Information Systems, Rome, Italy (2003) 34–43
13. Belotti, R., Decurtins, C., Grossniklaus, M., Norrie, M.C., Palinginis, A.: Interplay of content and context. *Journal of Web Engineering* **4** (2005)
14. Pernici, B., The MAIS Team: Mais: Multichannel adaptive information systems. In: Proc. Intl. Conference on Web Information Systems Engineering (WISE), Rome, Italy (2003) 303–308
15. Norrie, M.C., Signer, B.: Switching over to paper: A new web channel. In: Proc. Intl. Conference on Web Information Systems Engineering (WISE), Rome, Italy (2003)
16. Norrie, M.C.: An extended entity-relationship approach to data management in object-oriented systems. In: Proceedings of ER'93, 12th International Conference on the Entity-Relationship Approach, Arlington, USA (1993)
17. Want, R., Hopper, A., Falcão, V., Gibbons, J.: The Active Badge location system. Technical Report 92.1, Olivetti Research Ltd. (ORL), 24a Trumpington Street, Cambridge CB2 1QA (1992)
18. Belotti, R., Decurtins, C., Grossniklaus, M., Norrie, M.C., Palinginis, A.: Modelling context for information environments. In: Proc. Intl. Workshop on Ubiquitous Mobile Information and Collaboration Systems (UMICS), CAISE, Riga, Latvia (2004)
19. Grossniklaus, M., Norrie, M.C.: Information concepts for content management. In: Proc. Intl. Workshop on Data Semantics in Web Information Systems (DASWIS), WISE, Singapore, Republic of Singapore (2002)
20. Norrie, M.C., Palinginis, A.: Versions for context dependent information services. In: Conference on Cooperative Information Systems (CoopIS), Catania-Sicily, Italy (2003)