Journal of Web Engineering, Vol. 4, No. 1 (2005) 057–078

INTERPLAY OF CONTENT AND CONTEXT

RUDI BELOTTI, CORSIN DECURTINS, MICHAEL GROSSNIKLAUS
MOIRA C. NORRIE, ALEXIOS PALINGINIS
Institute for Information Systems, ETH Zurich
8092 Zurich, Switzerland
{*belotti,decurtins,grossniklaus,norrie,palinginis*}*@inf.ethz.ch*

Received October 1, 2004
Revised March 1, 2005

We examine the relationship between context engines and content management systems, showing by means of an example application how these should mutually interact with each other to ensure the timely delivery of relevant information. We show how a content management system can use context information to enrich its functionality and also how a general and abstract approach to content management can support context awareness. Information models of the general context engine and content management system that we have developed are presented, along with a description of how a symbiotic relationship of content and context can be achieved through the integration of these models.

Keywords: Context, Content Management, Information Modelling, Metadata

1 Introduction

Web technologies are no longer solely concerned with providing access to information and applications via desktop browsers. Nowadays, they are also widely adopted as the basis for a universal platform for mobile and ubiquitous information systems. Access via mobile devices requires a high-level of adaptation to cater for the varying, and often restrictive, characteristics of these devices as well as the situation and task at hand. Information must be carefully selected, filtered and transformed to meet the needs of a user in a constantly changing environment.

The notions of context and context-aware applications have become prevalent in the research community as a means of adapting information delivery to match the situation of the user. While a number of application-specific solutions and context-frameworks exist, there is still no agreement on a universal model of context and it is doubtful as to whether one can ever be reached that will satisfy all application developers. Instead, we believe that it is important to provide general context engines that can be coupled with existing applications to augment them with an application-specific notion of context.

At the same time, the globalisation of information access through web browsers and the proliferation of information services has led to increased demands for personalisation and customisation. Users want to be able access information of particular interest to them in the language of their choice. History may play an important role in determining both user preferences and also the context of access.

Emerging from classical information systems, content management systems have been developed to attain the goal of adequate information delivery. These systems extend the traditional functionalities of organising and accessing data with means to publish, present and deliver content. Originally developed for the administration of large websites, content management systems have evolved into general web publishing platforms responsible not only for content management but also its delivery to a range of devices and users.

Within our research group, we have developed both a general context engine and a content management system based on well-defined metamodels. In this paper, we describe how we have integrated these to produce a general context-aware content management system and discuss the interplay between content and context in such a system. The resulting system provides a general platform for the rapid development of context-aware information systems based on an application-specific model of context. As we will see, the interplay between content and context is not a client-server relationship, but rather a symbiosis where both counterparts profit from each other. To illustrate this, we use the example of a mobile tourist information system that we developed for visitors to the Edinburgh Festivals to explain the concepts behind the system and what is involved in developing an application.

In Sect. 2, we first present a brief overview of related work that has been done in the field of context-aware computing in general and then go on to specifically consider the area of content management systems. We then introduce the Edinburgh Festival demonstrator application in Sect. 3 in order that we can use it as an example to explain the concepts introduced in the paper. The general context engine at the core of our approach is described in Sect. 4. Our content management system is then described in Sect. 5 along with a discussion of how context influences the content management system. The opposite direction—how the content management system influences the context engine—is presented in Sect. 6. Concluding remarks and a discussion of future work are given in Sect. 7.

2 Context and Context-Aware Computing

The influence of user behaviour and, in general, the situation of the environment where an interaction is taking place has been identified as context and studied by different research communities including philosophy, linguistics and social sciences. For example, in everyday speech, we use context to determine the meaning of spoken phrases. How we interpret and respond to what someone is saying may be dependent on, not only linguistic context, but also other factors such as who is doing the speaking, where they are standing and what body gestures they are using.

In a similar way, how a system interprets and responds to a user request may be made to depend on, not only the explicit content of the request and application state, but also contextual information about the user and their environment. This has led to a new category of computer applications which are *context-aware* [1, 2, 3] and use implicit parameters that represent information about the state of the user and their environment, along with explicit request parameters and the current application state, to determine the response to a request. Thus, a context-aware application system may adapt to or learn from context [4].

Context-aware computing has received a lot of attention in the fields of ubiquitous and mobile computing where context is strongly associated with information extracted from the physical environment through sensor data. Spatial information such as location, orienta-

tion and speed or environmental information such as temperature, light and noise level are commonly sensed context properties for applications in these domains.

A variety of context-aware applications have been designed and implemented including office and meeting tools [5], tourist guides [6, 7] and a system to support archaeological fieldwork [8]. All those examples propose context models specifically tailored to the application domain in question and directly implemented in the application logic. Although each application has its own context model, they have many concepts and components in common. Some researchers therefore sought to provide a common infrastructure that would support reusability and rapid development. For example, the Context Toolkit developed at Georgia Institute of Technology [9] and the Context File System of the Gaia project at the University of Illinois at Urbana-Champaign [10, 11] are both efforts to provide general context frameworks appropriate for ubiquitous computing.

Based on such a framework, a context-engine can be created to acquire, manage and offer context-relevant information. Applications can then use the engine to gather context information which enriches their behaviour. Furthermore, the proposals offer an infrastructural solution to the architecture, based on either a transparent host/port distributed communication [9] or CORBA [10, 11]. Additionally, [9] separates the actual sensor from the abstract context which allows the decoupling of the context acquisition and the application using the context.

Although the functionality and infrastructure offered by such context frameworks simplifies the implementation of context-aware applications, they still lack a clear information and conceptual model that describes the context engine. Such a metamodel can be used as a reference model for context-engines, increasing the understanding of different approaches and enabling a comparison based on the concepts introduced in the metamodel. Context information exchange is also facilitated by using appropriate mappings to transform context-engine specific data to the metamodel. In [12], we present a general information model for context and, in the next section, we describe a context engine based on that model.

A general purpose context-engine can be used by any type of application. We mentioned above applications from the domains of ubiquitous and mobile computing where context is often used to model properties of the physical environment based on sensor data. But system responses could be made to depend on a whole host of factors about the user and their situation. For example, it might depend on the user's profile and preferences in terms of language capabilities, interests and level of experience. It might also depend on the device that they are using and the time of day. It might depend on the task at hand: For example, if the user is seeking information about train times, the appropriate response might depend on whether they are planning a trip or just want to check when they should meet friends at the station. History may also play an important role in helping to determine both the state and the desires of a user. For example, the history of access in a web browser determines what information has already been delivered and can also help identify the interests of a user and the task at hand.

The notion of context-awareness has therefore also appeared in other areas of computer science where there was a desire to adapt user interaction and possibly behaviour according to some model of context. In the field of web engineering, the first steps towards context-awareness were made by introducing adaptation concepts directly into hypertext models [13,

14]. Such Adaptive Hypermedia Systems (AHS) are based on various user characteristics presented in a user model. Most of the early examples focus on information collected from the user's click stream. In [15], user data, usage data and environment data are distinguished. An extensive discussion on the field of AHS and related work can be found in [16].

We should mention here that some authors and systems handle general user preferences such as colours, style and interests separately from context, considering them as part of a static user profile rather then a dynamic characterisation of an interaction situation. Alternatively, one can handle both in an integrated manner by considering a user profile as a static definition of part of the context state that is loaded initially and may then be updated dynamically if appropriate. What is important is to note that one cannot categorically state whether a specific characteristic of a user or an environment is or is not part of the context definition. Rather, the definition of context will always depend on the application and its specific context model.

As stated above, AHS systems typically employ a user model to capture the adaptive-relevant properties. After each interaction, the user model is updated and, based on adaptation rules, appropriate output is generated. Adaptation is applied to the basic concepts of hypermedia models, namely, component and link. Rules are defined to control the composition of components (fragments) or even alter link presentation. Approaches that use rule-based adaptation of hypertext are AHAM [17], a reference model for adaptive hypermedia and AHA! [18], an architecture for adaptive hypermedia that is based on AHAM.

Due to the fact that components encapsulate both notions of content and presentation in an indistinguishable manner, it is not possible to adapt only one of them in isolation. This is unfortunate if one considers that a major adaptation requirement of web applications is context-dependent presentations. When evaluating AHS, we are confronted with the situation found in other application-specific approaches. The user model is inspired and specified with the browser/server environment in mind. Thus, no general approach to context is taken with important properties such as context history, sensor generality, quality etc.

To provide a better solution, model-based approaches to web site development have recently been extended to support more general forms of adaptation [19] and context-awareness [20, 21, 22]. Based on strong and abstract information models, these approaches have the potential to exploit context-awareness in many dimensions, keeping the solutions as simple as possible. In [19], delivery channel characteristics can be used to influence the hypertext and navigation model of the original WebML model [23]. In this approach, no explicit context model is used. Instead, general data modelling techniques are made available to manage context information. Although the presentation cannot be adapted explicitly, it can be influenced by adapting similar content units bound to different presentation templates. A comprehensive survey of context and context-dependent customisation can be found in [24].

Our approach is based on extending the functionality of a database management system specifically to support web publishing. This involves managing not only application data, but also web document structures and presentations within the database. Web document structures are recursively defined in terms of web components which define containers for static and/or dynamic content, possibly generated from application data. Web components are associated with one or more template objects which define the presentation of content.

In earlier work, we developed the web publishing platform OMSwe [20] as an extension

of the object-oriented database management system OMS Pro [25]. In addition to the basic content and presentation management facilities outlined above, OMS Pro was extended with a special versioning mechanism to support context-awareness. The context model of an application defines a set of characteristics and the context state at any time is a corresponding set of characteristic/value pairs. Different versions of an object can exist corresponding to different characteristic values. For example, if language is one of the context characteristics of an application, then different language versions of an object can exist, each labelled with a specific value for the language characteristic. One or more characteristic/value pairs may be associated with an object version.

Context information is provided in the form of characteristic/value pairs from a context gateway component. This information builds the request state, based on which, the database will retrieve the most appropriate versions of the objects involved in the response. Since all information—application data, web components and presentation templates—are all represented as objects in the system, all may be context-dependent. Thus, through a single mechanism, OMSwe is able to make content, structure and presentation context-aware. These essentially correspond to the content, hypertext and presentation levels that are defined in [26] and used in [24] to classify the scope of the adaption to context.

In a later project, the eXtensible Content Management (XCM) system [27] took the approach one step further and defined a full-strength content management system with well defined general information concepts. Also, based on our experiences with OMSwe and as mentioned above, we developed a more general context metamodel and engine suited not only to general web publishing, but also for applications in the ubiquitous and mobile domains [12].

In the following sections, we present the main features of the context engine and XCM and then go on to discuss in detail the interplay between them. However, before doing so, we provide an overview of the demonstrator application used in all examples to explain the model and operation of the resulting integrated system.

3 Edinburgh Festival Example

Tourism is a domain with considerable potential for the use of mobile technologies and context-awareness. Typically, the information that a tourist wants is related to various contextual factors such as location and time of day as well as preferences and places already visited. A number of projects have therefore developed context-aware digital tourist guides, including Cyberguide [6] and GUIDE [28].

To test the appropriateness of our system as a general platform for mobile information systems, we also chose to look at tourism as an application domain. However, in contrast to other projects, we wanted to investigate the use of technologies for digitally augmented paper [29] in conjunction with voice input/output as the basis for interaction rather than PDAs. In addition, we wanted to provide access to the information services via desktop browsers. The sheer variety of interaction modes, together with the severe limitations of effective bandwidth in dialogue-based interaction, presented serious challenges in terms of the flexibility and generality of the underlying content management system. In addition, instead of opting to implement a general city guide, we developed a demonstrator system for visitors to the Edinburgh Festivals which is much more dynamic in terms of data and context-dependent requirements. In the remainder of this section, we present an overview of the functionality

and architecture of the festival system.

Every year in August, the city of Edinburgh in Scotland is host to the Edinburgh Festivals, a federation of individual festivals including the International Arts Festival, the International Book Festival, the International Jazz and Blues Festival, the International Film Festival, the Fringe Festival and the famous Military Tattoo. The various festivals cover a broad variety of cultural interests such as theatre, art, music, dance, cinematography, literature and much more. Performances take place in various locations ranging from large theatres to pub scenes and open-air stages in the streets of the city centre. Every year, the festivals attract thousands of visitors from around the world as well as local residents. In 2002, close to half-a-million tourists have been estimated to have visited the festivals.

In a situation where a large number of events take place in a decentralised manner throughout the city over a short period of time, effective information delivery to the visitors is crucial to their satisfaction and the overall organisational success of the festival. A major problem for tourists is how to select events to attend and put these together as part of an overall schedule.

Printed information about the events comes in various forms, including festival programmes, reviews and daily event programmes published by newspapers and flyers. This information would easily fill hundreds of pages on its own while offering only limited search facilities. Due to its generic nature, it also does not offer any form of personalisation and hence does not fit the needs of individual tourists. In addition, each festival offers a web site with information about venues, events and on-line ticket ordering. In the case of the Fringe Festival, visitors can also enter their reviews on events. Although these sites do not currently offer personalisation and customisation, this is something that would be useful and could be incorporated into future versions.

Major sources of information such as the various festival brochures and web sites are best suited to pre-visit access where users can take the time to carefully study the programmes, plan their visit and order tickets. However, especially in the case of the Fringe Festival where hundreds of events are on offer each day, tourists often decide on events during the actual visit. In this case, selections are often based on locality, time of performance and reviews which are copied and pasted on noticeboards outside venues.

Apart from selecting events and planning which shows to visit, navigation and orientation present serious challenges, especially for foreigners, as there are hundreds of events each day and many of the festivals take place at multiple venues spread across the city.

The aim of the Edinburgh Festival project (EdFest) was to develop a prototype application based on technologies for ubiquitous and mobile information environments to support visitors of the Edinburgh Festivals. While the main focus of the initial phase of this project was the context-aware delivery of information and services through various channels, we also wanted to support collaboration among tourists through the entry and sharing of personal reviews during visits.

Generally, the tourist experience can be divided into three stages: pre-visit, visit and post-visit, and we wanted to support all three stages based on a single system. Starting with the pre-visit phase, tourists should be able to access information about festivals, select events and book tickets through their normal desktop web browser. Later, during their visit, tourists should in addition be able to access up-to-date or last-minute event information. They also need to be assisted in finding their way around the city and to have easy access to information

about venues and events near to their current location. They may also want to enter reviews shortly after visiting an event and share these with other tourists. Finally, after the visit, tourists may want to generate a personal diary of their visit as well as entering reviews or discussing their experiences with other people.

With the EdFest project, we advocate that all information should be managed by a single web information system that can be accessed through various channels corresponding to the different needs of the user. The web sites of the individual festivals, the mobile clients that users take with them on their visit, as well as the discussion boards they use to share their experiences, are built using one context-aware content management system.

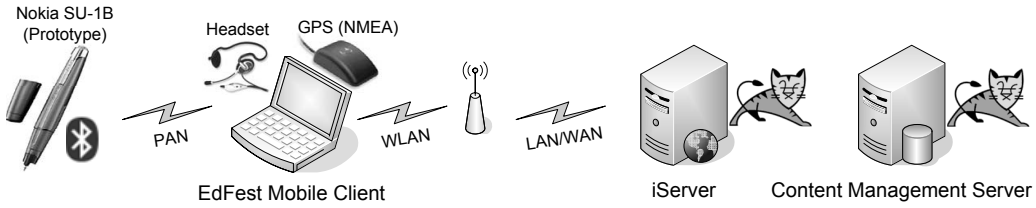

Fig. 1. Architecture of the EdFest Prototype

Figure 1 displays an overview of the architecture of the EdFest prototype application that was developed during a first project phase and tested in Edinburgh during August 2004. During this phase, we decided to investigate alternatives to PDAs and visual displays in a mobile information environment and developed mobile clients based around an interactive paper map and programme, using a system for digitally augmented paper based on Anoto technologies and a specially modified Nokia digital pen. This was coupled with the use of speech dialogue for both input and the delivery of information.

It is beyond the scope of this paper to describe these client technologies in detail and also the cross-media link server (iServer) that we have developed for linking paper and digital media. The interested reader can find further information in [29, 30]. In the context of this paper, what is important is the fact that we had to support a large variety of channels and modes of interaction, including conventional desktop browsers, speech dialogue and also completely new forms of interactive paper. Further, since these are the subject of experimentation in mobile environments, it is important that we can easily adapt interfaces and introduce alternative channels.

The EdFest Mobile Client shown on the left-hand side of the figure consists of a wearable computer, a headset, a GPS receiver and digital pen. As all user interaction with this client is purely based on the digital pen and paper together with audio, the client's principal task is to detect which regions of the printed festival programme and map the visitor is pointing at with the digital pen. From this position data, the cross-media link server (iServer) is able to determine what information has been linked to the respective region on the paper. iServer then constructs a request that is sent to the content management server shown on the right-hand side of the figure. The content management server runs a combination of our Context Engine and the eXtensible Content Management System (XCM).

In the remaining sections, we will focus on the content management server and show how its two components can be combined together to deliver context-dependent information to users. While this approach is not limited to the EdFest project nor in any way was developed

solely for it, we will use this example throughout the following sections to illustrate the interplay of content and context and indicate what is involved in using the system to develop a context-aware application.

4 Context Engine

In this section, we introduce the general context engine that we have developed, using examples based on the EdFest application to show how application-specific models can be specified and explain the operation of the context engine.

The context engine is based on a basic and abstract concept of context that was influenced by the application and framework requirements, as well as the discussion of context presented by Dourish in [31]. While the underlying metamodel is described in full in [12], here we concentrate on explaining how an application-specific context model is described in terms of the metamodel and the associated language.

Every entity of an application can potentially have one or more context elements associated with it. Then we say that the application entity is the subject of the associated context elements. For instance, the EdFest application schema defines the concept of a user, as is typical of many context-aware mobile information systems. The context of a user could be described by physical properties such as the user's position. Then a context element *position* would record the history of user positions through the automatic creation and storage of multiple associated context instances, each of which would hold a particular position value, e.g. `4722.882N,832.7608E,571,3`.

For the above example of a position value, we could define a context element with the raw, unstructured position information as delivered by a GPS receiver. There are standardised formats for GPS data, such as for example the NMEA format. For the example, we use a simplified version of this format for better readability. It consists of latitude, longitude, height and the number of satellites that were used to determine the position.

If the values stored within context instances do not provide any information about the scale and format used, this could cause problems in ensuring that these values are handled correctly and errors detected. Moreover, since the context engine may supply context information to many different applications, it is vital that some form of description is available to ensure that data is processed correctly and possibly transformed into the formats required by a particular application. We therefore introduced a type system for defining concepts within the context engine.

The system distinguishes four kind of types as shown in Fig. 2. The notation used is that of the Object Model (OM) [32], a model that integrates object-oriented and entity-relationship concepts. *Base types* define common values such as `string`, `integer`, `boolean` etc. The model allows base type restrictions as used in XML Schema. Hence, we can create base type restrictions with new names that represent values with specific semantics. In the example below, we define a base type `gps_coordinate` as a restriction of `string`. Restrictions of a type are expressed through a constraint that checks the validity of the values acceptable for the defined type.

```
btype gps_coordinate is string {
      constr:  'self(S), check_gps_format(S)';
};
```

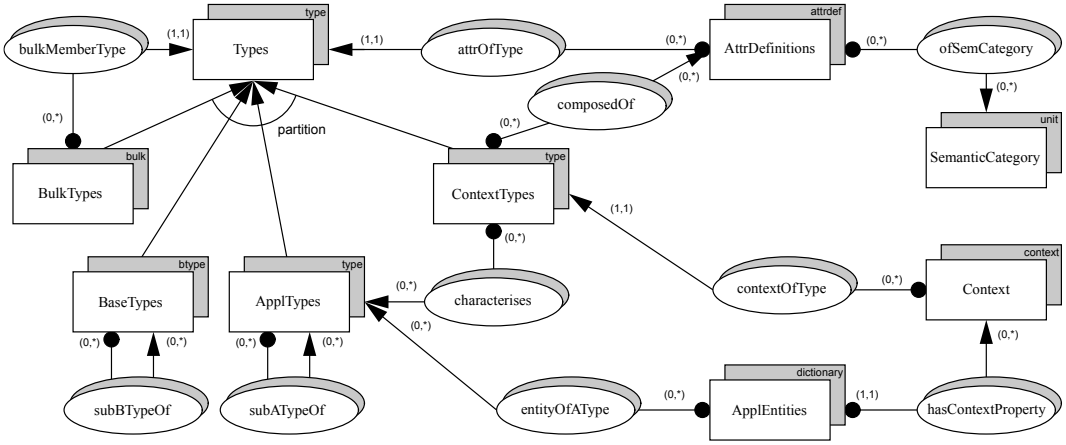

Fig. 2. Context Type Model

The context engine is implemented in Prolog and the Context Definition Language (CDL) used here has been developed to define concepts and their types for an application-specific context model. Operational components of CDL are defined in Prolog. For the above example, the constraint is a predicate that evaluates to `true` if the given value is legal. To check if a value is appropriate as a `gps_coordinate`, the value is first retrieved using the predicate `self/1` and then checked to be in the correct format using the `check_gps_format/1` predicate.

In addition to the base types, the context model uses references to application-specific types to define and control application entities. Apart from a unique reference of the pattern `applicationID:typeID`, the context engine does not deal with any aspects of these *application types*. This allows the context engine to use application objects as context elements without actually having to know about the semantics of these objects. An object of type `edfest:user`, for example, might have a context element `currentLocation`, which is of type `edfest:venue`.

The third class of types are the *bulk types* which designate lists of values of a given member type. Bulk types come in four different flavours *set*, *sequence*, *ranking* and *bag*, depending on the bulk semantics. A set contains no duplicates and is unordered. A bag is also unordered, but elements may have multiple occurrences. A ranking is ordered and elements may occur only once. A sequence is as well ordered and elements may have multiple occurrences. In the EdFest project, we use the languages that a user understands as one context element for providing content to the user. This information is retrieved from the user profile. As most users understand multiple languages, we use a bulk type rather than a base type for this context element.

```
btype language is string {
     constr:  'self(S), check_iso_language(S)';
};

bulkType languageList is ranking of language;
```

The bulk type `languageList` is actually a *ranking*, i.e. the list is ordered but does not contain duplicates, allowing a user's order of language preference to be modelled. In the case of EdFest, if a content object is available in multiple languages, the content management system goes through the `languageList` context element of the user and selects the version that corresponds to the first language in the list supported by the content object. If the content is not available in the preferred language of the user, the system can therefore provide an alternative version that is also useful to the user. If no matches are provided, then it provides it in the default language, which in this case is English. More details of this mechanism are given in Sect. 5.

Finally, the model supports *context types* which define the composition of the context over a set of attributes of a given type. In the following example, we define a context type `ctx_position` with one attribute `coordinate` of type `gps_coordinate`.

```
contextType ctx_gps_position characterises edfest:user {
     coordinate:  gps_coordinate;
};
```

Optionally, the context type designates the type of entity to which it can be bound. In the above example, the defined context poses the constraint to characterise application entities of the type `user` defined in application `edfest`. Application types can be defined in an is-a hierarchy, designating a compatibility among them (not shown in this example).

Note that although values of the context type `ctx_gps_position` have a well-defined semantics, applications which want to use them still have to parse the coordinate string in order to get the specific parts of the value. A more structured representation of GPS coordinates could be implemented, based on btypes `gps_longitude`, `gps_latitude` and `float` for the height above ground. The two representations can coexist and each application can choose the context element that best matches its requirements.

```
contextType ctx_position characterises edfest:user {
     longitude:  gps_longitude;
     latitude:  gps_latitude;
     height:  float;
};
```

After declaring the types, an application can instantiate context elements and bind them to some specific application entity. In the following example, we create the position context for the entity `edfest:fred`, which is an instance of `edfest:user`.

```
context c_fred_position:  ctx_position describes edfest:fred
```

Context elements are either queried directly from client applications or received through an event notification mechanism. Such clients play the role of consumer with respect to the context engine. On the other hand, a sensor abstraction exists that encapsulates the context acquisition mechanism. Sensors are well defined components that are initialised with some parameters and bound to context elements. Sensors could either be hardware or software in nature. A hardware sensor could be a GPS device and a software sensor could, for example,

be a program that extracts the current application status with respect to an interaction.

To allow reusability and separation of concerns, we introduced the concept of a *sensor driver*. It holds the acquisition logic of the sensor. Multiple sensors can be instantiated based on a single sensor driver. The next example defines a GPS sensor that is connected to the serial port of a computer and provides the user's position context information based on the `ctx_position` context type.

```
sensorDriver gpsSensor(serial_port: integer): ctx_position;
```

Then the following statement initialises the actual GPS based on the given sensor driver, providing context information to the `c_fred_position` context.

```
sensor s_gpsFred: gpsSensor(1) provides c_fred_position;
```

Moreover, if you need to change the current source of a certain context element, you can do so simply by changing the sensor providing the context information without the need to redefine anything else. The following code demonstrates how you can simply change the definition of the source of position context information for `fred`.

```
sensor s_position1: newPositionSensor(1) provides c_fred_position;
```

In addition to the sensor driver definition, an implementation is necessary. For our prototype of the context engine, it is coded in SICStus Prolog [33] and bindings to Java are used in cases where the actual sensor offers a Java API. In the EdFest project, we have integrated the context engine with various physical and software sensors. For example, we use GPS receivers to acquire context information about the user's positions in the city and software sensors to get context information from various components of the EdFest application. This will be discussed in more detail in Sect. 6. In the next section, we first want to look at how applications, in particular a content management system, can use context information provided by the context engine.

5 From Context to Content

To demonstrate the interworking of the previously described context engine with an application system, we show in this section how it can be integrated with a content management system. Further, we outline how information flows from the context engine to the content management system and vice versa. As a content management system, we have chosen our own eXtensible Content Management (XCM) [27, 21]. Like the previously described context engine, the current version of XCM is also defined using the Object Model (OM). It is implemented in Java using OMS Java [34], an object-oriented database management system based on the OM model. The original version of XCM supported only a rather simple implementation for context and has now been extended to work together with our context engine described in the previous section.

XCM has been designed as a platform that provides support to applications requiring features typical of content management such as user management, workflows, personalisation, multi-channel delivery and multi-variant objects. At the heart of the system stands the

separation of content, structure, view and layout. The concept of *content* allows data to be accumulated into content objects and stores metadata about this content. As it is often required that the same content object may be delivered in different variations, the system supports what we call "multi-variant objects". For example, an event description may have multiple variants that correspond to the different languages in which it is available.

Multi-variant objects can comprise much more than the simple dimension of language. Other dimensions are often required, such as the target group of users or whether it is free or premium content for which users have to pay. Our system does not predefine the possible dimensions, but rather provides support to handle any annotation of content variants that makes sense in a given application domain. To achieve this, the concept of a content object is separated from its actual representation while still allowing for strict typing. In Fig. 3, a metamodel is displayed that shows how XCM represents multi-variant objects. Again, the notation and semantics are those of the OM model.

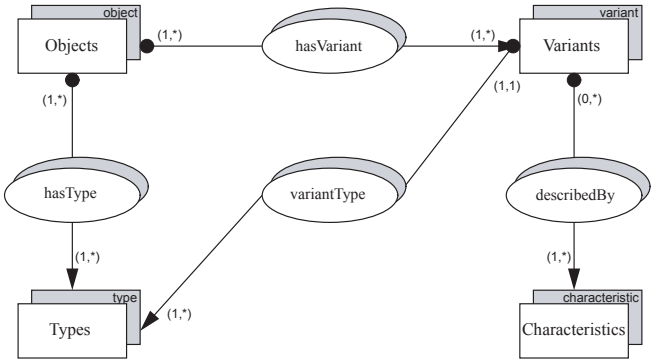

Fig. 3. Metamodel for XCM Multi-Variant Objects

At the top of the figure, the separation of the concept of an object and its actual content is clearly visible in the form of `Objects` linked to a non-empty set of `Variants` by means of the association `hasVariant`. Each variant of a multi-variant content object can be described by `Characteristics` that are linked to the variant over the association `describedBy`. A characteristic is simply represented as a `(name, value)` tuple. For example, to annotate a variant for English, one would simply associate it with the tuple `(language.name, english)`. This mechanism is actually very similar to the one of OMSwe described earlier in Sect. 2. As we will see later, characteristics play an important role in context-aware content management applications.

XCM also manages metadata about the types of content objects and their variants. Thus, in the metamodel, objects as well as variants are associated with `Types` using the `hasType` and `variantType` associations, respectively. This information can then be used by the system to determine whether the type of a variant matches the type of the corresponding object. As the cardinality constraints indicate, an object can have more than one type to support polymorphism and multiple instantiation.

XCM uses the concept of *structure* to build content hierarchies from multi-variant content objects. In a content management system, structures are required to build complex objects

such as pages, collection of pages or "folders". Our system uses the very simple component-container approach to build these structures. As a container can hold a number of components and other containers as well, arbitrary tree-based structures can be represented. Structure objects, i.e. containers, form the inner nodes of the tree, whereas the actual content is located in the leaves. Separating the structure from the content makes possible different access patterns to the same content.

Personalisation in XCM is achieved through the concept of a *view*. A view decides which aspects of a multi-variant content object are presented to the user. Further, it can aggregate other information to the object based on the schema of the content, thereby augmenting it. Hence, our view concept is not unlike that found in relational database systems. Managing different views for different users or situations effectively provides support for personalisation and custom content delivery.

Finally, the concept of *layout* encapsulates the graphical representation of the content. The layout is applied to container and content objects by means of templates that match the type of the target object. As for all four basic concepts of our system, layout objects can have multiple variants for different requirements. For example, a template to render an event from one of the Edinburgh Festivals can have one variant to produce HTML and another variant to produce VoiceXML for rendering by a text-to-speech engine such as the IBM VoiceServer SDK that we have used in the EdFest project. Hence, layout objects are required to support multiple presentation channels.

Based on this overview of the basic elements of XCM, we now go on to describe the integration of the context engine into the system. As mentioned before, our context model allows application entities to be linked to context elements within the context engine. In the example of the content management application, it is intuitive to link the application concept of `Sessions` to various context elements such as the user's identity and language, the version of the browser and other information about the current situation of the user. As sessions are already used in content management systems to store values essential to the current user, it is only natural to use this concept and augment it with context information.

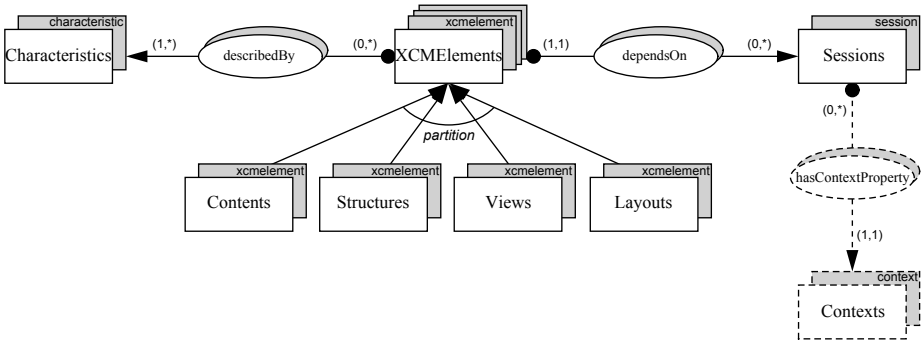

Fig. 4. Conceptual Model for the Context Binding

Figure 4 shows the model of the binding between XCM and the context engine. The parts that physically belong to the context model of our context component are represented with dashed lines. The four basic concepts have been unified with a common super concept

`XCMElements` which depends on the values of a given session. The separation between an object and its variants has been collapsed into a box with multiple shades to indicate that these objects can have multiple variants. From the model, it is apparent that, not only content, but also structure, view and layout can have multiple variants and thus adapt to context. Again, each variant can be described with characteristics as shown on the left-hand side of the figure.

This conceptual link between context elements and the application concept of a session is materialised using the context type definition given in the example below. The displayed CDL code first creates a base type for physical internet addresses by means of a restriction on the type `string`. Then a context type is created to represent the browser that is used in a given context. The actual binding is defined by the third declaration. It defines a type `ctx_session` which is linked to the concept `xcm:session` within XCM. It comprises four attributes that represent the context information available from the sensors. The attribute `user` provides the identity of the user of the current session, `lang` their language and `browser` the name and version of the browser used. Finally, the attribute `host` gives information about the physical address from which the content management system is accessed.

```
btype ip is string {
     constr:  'self(S), split(S,".",L),
                    \+ (member(X,L), \+ conv_integer(X,_))';
};
contextType browser {
     name:  string;
     version:  string;
};
contextType ctx_session characterises xcm:session {
     user:  xcm:user;
     lang:  language;
     browser:  browser;
     host:  ip;
};
```

It is also possible to create context types that are built from existing context types. The following code illustrates how we can define a `ctx_user_session` type that characterises application objects of type `edfest:user` and includes the context information about the aforementioned session context `ctx_session` and the user's current position `ctx_position` that was defined in Sect. 4.

```
contextType ctx_user_session characterises edfest:user {
     position:  ctx_position;
     session:  ctx_session;
};
```

Having explained how the context engine is bound to the content management application, we now go on to discuss how such a system reacts to the information supplied by the context

component. In XCM, context information is used to select the appropriate variant of an object, i.e. the context information is compared to the characteristics of each variant of such an object and the version that matches best is selected automatically. In practice, this can prove to be quite difficult, as the characteristics available to the system need not fully match the incoming context information. It is often the case that the context information comes at a higher level of detail than the metadata stored in the content management system. Of course, the opposite case that the metadata is more precise is also possible. In these cases of "under-" and "over-specified" context information, the matching process uses heuristics to select from the set of possible variants or to select a default variant if no match is found at all. A similar matching process together with heuristics is used in [20, 35].

For the mapping of context information to characteristics to work, the names of the `(name, value)` tuples have to match. This means that some sort of convention, taxonomy or ontology has to be used that assures a uniform naming scheme. As the definition of such conventions is clearly beyond the scope of this work, we adopt the simple approach of what we call "property paths". A property path locates a value inside a context element and can be derived from its context type definition, e.g. the value of the browser version in the current session would be identified by the path `ctx_session.browser.version`.

Further complexity arises from the fact that XCM supports not only simple values for characteristics, but also sets and ranges. This is used to specify, for instance, a set of matching browser versions with the characteristic `(ctx_session.browser.version, [5.0, 5.5, 6.0, 6.1])` or a time period given as an interval when a variant of an object is valid with `(valid, [1/1/2004..31/1/2004])`. XCM also provides the notion of *mandatory* and *"show-stopper"* characteristics. In contrast to *unconstrained* characteristics, these categories of properties are not freely matched to the context information, but instead are treated specially. A mandatory characteristic takes precedence over all other non-mandatory values, whereas a "show-stopper" property would not allow a certain variant to be selected if the context value does not match that of the characteristic.

Having discussed the integration of the context component into our content management system, we can finally give an architectural overview of the whole system. Figure 5 shows a graphical representation of all components. At the bottom, we show the user who communicates with the content management system through a browser requesting pages. In the case of the EdFest system, this may be a usual desktop browser, a voice browser or an "interactive paper browser" realised through a combination of paper augmented with the Anoto pattern, a digital pen and the iServer.

The content management system then interworks with a private context component to manage the context proprietary to the system. Further, the private context component is connected to a shared context component that manages context for several applications and provides a global notion of context. This shared context component is connected to the sensors as shown at the top of the figure. The two context components and the content management application are conceptually connected by the model that is common to all three of them. As the figure also indicates, information can always flow in both directions. We discuss the opposite flow of information in the next section.

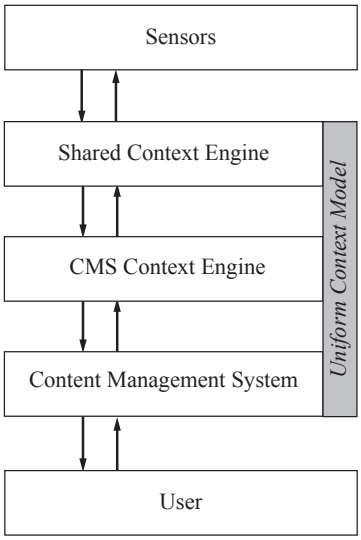

Fig. 5. Overview of the Architecture

6 From Content to Context

In the previous section, we described how the content management system can use the context engine to provide context-aware content for a user. In this section, we focus on the reverse relationship. The content management system can also act as a provider of information for the context engine. This context information can then be used either by the content management system itself, or by other context-aware applications and components of the information environment.

From the different types of context described in [36], the *user context* is of special importance for information environments. The acquisition of context information from the external, physical environment of the user is relatively simple to achieve. Sensors, mounted on the person or embedded in the physical environment, can measure the relevant values of the user's environment. For example, an EdFest user carries a GPS receiver that is used to determine their current position within the city.

Context information from the internal, mental state of the user on the other hand is very difficult to acquire. The mental state includes things like the thoughts of the user, mood, intentions and associations. It makes up a significant and important part of the user's context. However, it cannot be measured or sensed directly. The only way to acquire context information about the inner state of the user is indirectly, through manifestations in the physical and virtual realm. For example, the destination of a user who moves around a city cannot be determined by a sensor directly. However, the user's last positions, along with a profile of the user's destinations, the current weekday and time etc. might give information about probable destinations [37]. Most of the information in this example is sensed from the physical environment. Information from the virtual environment, for instance the applications that a person uses, is also very valuable for the indirect derivation of context. In the previous example, the fact that the user just looked up the location of a venue on a geographical

information system or has scheduled an appointment with a friend at a particular location in five minutes, is a very useful piece of contextual information for determining or predicting the next destination of the user.

The content management system is an important and central component of the application and often also acts as gateway to other applications and services of the information environment. Therefore, it is highly relevant for the acquisition of context information. Services and other applications can often define more detailed context information because of their more restrictive functionality. However, the content management system can provide the necessary semantics for the context information to actually be useful. It has additional metadata that is used to turn information into content and can also use this metadata to provide or enrich context information. Imagine, for example, a service that is able to look up the location of a particular venue for the EdFest project. On its own, the information from a request to this service cannot really be used as contextual information since we do not know for instance which user requested the information. In this example, the content management system uses its own context, i.e. which user triggered a request, to provide additional context information. Another example would be the lookup of an event in the EdFest database. It makes a big difference whether the user explicitly and consciously requested information about this event or whether it was just displayed as a recommendation from the system. The EdFest database cannot detect this difference as it only receives a request for an object. The information, however, is available in the application metadata of the content management system. This metadata describes how the response is composed, what parameters are determined by explicit input from the user and what parameters are constructed from context information or other components.

Context information can be extracted from application data, content data, the local context model of the content management system or the user sessions. For example, session information can be used to provide contextual information about the level and type of activity of a user such as whether the user is very busy with the system, just clicking around or not using the system at all. An example of context from the local context model of the content management system would be the current language. The user might have changed this language explicitly in the process of an interaction. The kind of context information mentioned in these two examples is quite simple and could also be provided by other components. It becomes much more interesting, if we take the content metadata and application data into account. With the content metadata, for example, the content management system might be able to tell whether the user is currently browsing an overview page or drilling down to a very deep level of detail. This could be estimated in a generic way by the number of content objects that are used to compose a page. The fewer content objects used, the larger the probability is that the page is a detail page. The advantage of using the content metadata only is that it is independent of the actual application that is running on top of the content management system. Software sensors can be provided that extract this context information in a generic way. However, if we also take the application model into account, the context information can be even more specific. For example, the category of the event that the user is currently browsing can be used as context information in the EdFest system.

Note that, for most of the examples mentioned above, the actual data model of the application is very important. The fact that we have an expressive semantic data model in our

content management system greatly simplifies the task. Actually, the extraction of semantic context information would be very difficult, if not impossible, without a proper application model in the content management system. In some traditional content management systems that do not support customised data models, the EdFest example with the categorised events could be modelled with a page for each category containing a list of events in the form of paragraphs, links etc. with a page for each event giving more details. In this setup, it is not possible to extract the category in which the user is currently interested.

As described in Sect. 5, in our content management system, the application data is not stored in the form of paragraphs, links and pages, but rather according to a semantic application model. A customised software sensor could thus easily retrieve the categories of the events that the user is currently browsing and store corresponding entries in the context engine. In contrast to other context frameworks such as [9], we are also able to use application objects as context values through reference types as outlined in Sect. 4. This means that the content management system can store the actual object that represents the category in the context engine, rather than just the name of the category.

In the following example, we provide the definition of a context `ctx_category` that defines the category browsed by a user of type `edfest:user`. This type actually belongs to the application domain and is only referenced by the context engine.

```
contextType ctx_category characterises edfest:user {
      category:  edfest:category;
};
```

We instantiate a new context `c_fred_category` of this type for the user `fred`. This entity is also represented only by a reference to the application data.

```
context c_fred_category:  ctx_category describes edfest:fred;
```

We define a software sensor `categorySensor`, which is responsible for gathering the relevant category information from the content management system and associate it to the current user. `edfest:category` is also a concept from the EdFest application model. Finally, we instantiate the software sensor for the subject `fred`.

```
sensorDriver categorySensor():  ctx_category;
sensor s_category_fred:  categorySensor() provides c_fred_category;
```

Note that the names of the types and variables in this example are for better readability only. In the actual system, every concept is represented by an object rather than a name. Other instances of the software sensor `categorySensor` can be used to provide context information for other users of the application. But as the sensor works on the application data model, it cannot be used for other applications of the content management system. On the other hand, sensors that only work on the content metamodel of the content management system can be reused for all of the applications that are based on the metamodel.

It is important not to confuse the extraction of context information with general data mining, e.g. for user profiling. Similar technologies and algorithms can be used for both of

them, but whereas data mining focusses on general facts and information, for example about users, *context mining* is only concerned with the current state of the user. This might include historical information, but, usually, context information is of interest for a limited time only.

The extracted context information can be used for a variety of applications. Of special interest to us are ubiquitous and mobile information environments and corresponding platforms that integrate multiple context-aware applications for a physical location or virtual community. The EdFest project is an example of such an context-aware information environment and it integrates both a context engine and a content management system, as well as multiple application components. The core application of the EdFest prototype focusses on information browsing and presentation. It is based on the content management system and delivers content in HTML, VoiceXML and PDF (for the paper interfaces). It makes use of context information from the context engine, such as the location of the users. This information is provided by GPS sensors that the users carry around and is abstracted through the context engine. The content management system also uses the context engine for its own local context information, such as the current language of the user or information about the client device that is used to browse the system.

Another component of the application framework is the *Friend-Nearby* service. When two registered friends using the EdFest prototype come close together within the city, the service sends a notification to these users, informing them about each other. The component works on information provided by the context engine. This includes the position of the users, as well as a callback URL for notifications to the client devices. This URL is provided to the context engine by the client devices and can be used for any component of the platform to send events to the client devices. In the case of the Friend-Nearby component, the notification event triggers a request from the client device to the content management system that finally provides the event information to the user. The Friend-Nearby component uses the content management system for publishing and pushing information to the user. The actual events are only triggered based on context information from physical sensors and the client devices.

A similar notification service could also be implemented for friends who are interested in the same events or when one person browses an event that a friend of theirs has already seen. In contrast to the Friend-Nearby component, this service can only be provided based on context information that is held by the content management system. It could provide a `current_events_of_interest` context element, which is the list of events that the user has recently shown interest in. This information can be determined based on the pages that the user has recently requested. Another component could then trigger events based on this context information, perhaps combined with application data such as what events a user has visited. Actually, as the context engine already supports the notion of *context events* as described in Sect. 4 or more detailed in [12], this could also be implemented in the context engine itself without an additional component.

Another application for the use of context information provided by content management systems are community awareness applications. These applications focus on visualising contextual information about users to increase the level of awareness between the users. This information could include the presence and activities of a user, such as in the context-aware instant messaging and chat application described in [38]. It visualises a list of users that are present, along with their current state (present, busy, free-for-chat etc.). The current state

of the user is determined by context information. We plan to integrate a similar mechanism into the EdFest project whereby the system visualises contextual information related to users. This could include rather concrete things like the current location of a user or his activity (walking, browsing information or attending an event), providing users with awareness of the whereabouts of their friends and what they are currently doing. This is information which is already available in the context engine of the current implementation of the EdFest system or could be added quite easily by additional software sensors. Another more abstract and less direct example would be the visualisation of the trails of users not only through the physical space of the city as used in [39], but also the virtual world of the concepts of the information space. However, one issue with such applications is clearly that of privacy invasion.

The possibilities for the use of context information, be it for visualisation to increase awareness amongst a group of people or for providing context-aware information and services in general, are endless. The context engine with its metamodel of context information is the core component for providing these features. However, in the end, the quality of the context information is the linchpin of context-awareness. Content management systems with rich semantic constructs and well-defined context-aware publication processes are able to provide high-quality context information in a generic way and ease the implementation of application-specific software sensors for context acquisition.

7 Conclusions

We have discussed the interplay between context and content in terms of the relationship between the basic concepts of context engines and content management systems, respectively. We used the example of our own content management system to demonstrate how our generic context engine enabled us to seamlessly adapt all dimensions of content delivery—content, view, structure and presentation—through a process of matching context to multi-variant objects for these dimensions.

Moreover, we have shown that a content management system can be, not only a client to the context engine, but also a potential provider of context information. Our system XCM exhibits two features that make it an ideal component for determining the computation context: It consolidates well organised information from arbitrary sources and acts as a bridge between the user and the organisation through multiple communication channels. We have shown how this crucial and complete information provides context and how we use XCM as a software sensor to the context engine.

Our experiences developing a mobile information system for visitors to the Edinburgh Festivals have shown that the coupling of a general context engine and a content management can yield a powerful system for the development of context-aware applications. In particular, using database-driven approach leads to an ideal experimental platform for mobile and ubiquitous information systems in which all aspects of the systems can be updated dynamically.

Further demonstrator applications are under development, including one which deals with promoting community awareness in both physical and virtual environments. In this case, the context engine and content management system work together to control ambient information in a research laboratory environment. As an example, a public news service integrates both global and local news items and adapts according to the set of users present in the environment.

References

1. Peter J. Brown. The Stick-e Document: A Framework for Creating Context-aware Applications. In *Proc. EP'96, Palo Alto*, January 1996.
2. Anind K. Dey, Daniel Salber, Gregory D. Abowd, and Masayasu Futakawa. The Conference Assistant: Combining Context-Awareness with Wearable Computing. In *ISWC*, 1999.
3. Guanling Chen and David Kotz. A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, November 2000.
4. Henry Lieberman and Ted Selker. Out of Context: Computer Systems that Adapt to and Learn from Context. *IBM Systems Journal*, 39(3), 2000.
5. Roy Want, Andy Hopper, Veronica Falco, and Jonathan Gibbons. The Active Badge location system. *ACM Transactions on Information Systems*, 10(1), January 1992.
6. Gregory D. Abowd, Christopher G. Atkeson, Jason Hong, Sue Long, Rob Kooper, and Mike Pinkerton. Cyberguide: A mobile context-aware tour guide, 1997.
7. Nigel Davies, Keith Mitchell, Keith Cheverst, and Gordon Blair. Developing a Context Sensitive Tourist Guide, 1998.
8. Nick S. Ryan, Jason Pascoe, and David R. Morse. Enhanced Reality Fieldwork: the Context-aware Archaeological Assistant. In V. Gaffney, M. van Leusen, and S. Exxon, editors, *Computer Applications in Archaeology 1997*, British Archaeological Reports, Oxford, October 1998.
9. Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In *Proceedings of the 1999 Conference on Human Factors in Computing Systems (CHI '99)*, Pittsburgh, PA, May 1999.
10. Christopher K. Hess, Francisco Ballesteros, Roy H. Campbell, and M. Dennis Mickunas. An Adaptive Data Object Service Framework for Pervasive Computing Environments. In *Proc. 6th USENIX Conference on Object-Oriented Technologies and Systems*, San Antonio, Texas, USA, 2001.
11. Christopher K. Hess and Roy H. Campbell. A Context-Aware Data Management System for Ubiquitous Computing Applications. In *Proc. Intl. Conf. of Distributed Computing Systems (ICDCS 2003)*, Providence, Rhode Island, May 2003.
12. Rudi Belotti, Corsin Decurtins, Michael Grossniklaus, Moira C. Norrie, and Alexios Palinginis. Modelling Context for Information Environments. In *Ubiquitous Mobile Information and Collaboration Systems (UMICS), CAiSE Workshop Proceedings*, June 2004.
13. Peter Brusilovsky. Methods and Techniques of Adaptive Hypermedia. *User Modeling and User-Adapted Interaction*, 6(2-3), 1996.
14. Peter Brusilovsky. Adaptive Hypermedia. *User Modeling and User-Adapted Interaction*, 11(1-2), 2001.
15. Alfred Kobsa, Dietmar Müller, and Andreas Nill. KN-AHS: An Adaptive Hypertext Client of the User Modeling System BGP-MS. In *Proc. of the Fourth Intl. Conf. on User Modeling*, Hyannis, MA, 1994.
16. Hongjing Wu. *A Reference Architecture for Adaptive Hypermedia Systems.* PhD thesis, Technical University Eindhoven, 2002.
17. Paul De Bra, Geert-Jan Houben, and Hongjing Wu. AHAM: A Dexter-based Reference Model for Adaptive Hypermedia. In *ACM Hypertext '99, 10th ACM Conference on Hypertext and Hypermedia*, pages 147–156, Darmstadt, Germany, February 1999.
18. Paul De Bra and Licia Calvi. AHA! An Open Adaptive Hypermedia Architecture. *The New Review of Hypermedia and Multimedia*, 4:115–139, 1998.
19. Stefano Ceri, Florian Daniel, and Maristella Matera. Extending WebML for Modeling Multi-Channel Context-Aware Web Applications. In *Proc. MMIS'2003, Intl. Workshop on Multichannel and Mobile Information Systems, WISE 2003*, December 2003.
20. Moira C. Norrie and Alexios Palinginis. Empowering Databases for Context-Dependent Information Delivery. In *Ubiquitous Mobile Information and Collaboration Systems (UMICS 2003)*, Klagenfurt/Velden, Austria, June 2003.
21. Michael Grossniklaus, Moira C. Norrie, and Patrick Büchler. Metatemplate Driven Multi-Channel

Presentation. In *Proc. MMIS 2003, Intl. Workshop on Multi-Channel and Mobile Information Systems, WISE 2003*, Rome, Italy, December 2003.
22. Mike Perkowitz and Oren Etzioni. Towards Adaptive Web Sites: Conceptual Framework and Case Study. *Computer Networks*, 31(11–16), 1999.
23. Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web Modeling Language (WebML): A Modeling Language For Designing Web Sites. *Computer Networks*, 2000.
24. Gerti Kappel, Werner Retschitzegger, and Wieland Schwinger. Modeling Customizable Web Applications - A Requirement's Perspective. In *Kyoto International Conference on Digital Libraries*, pages 168–179, Kyoto, Japan, November 2000.
25. Moira C. Norrie, Alain Würgler, Alexios Palinginis, Kaspar von Gunten, and Michael Grossniklaus. *OMS Pro 2.0 Introductory Tutorial*. Institute for Information Systems, ETH Zürich, March 2003.
26. Daniela Florescu, Alon Y. Levy, and Alberto O. Mendelzon. Database Techniques for the World-Wide Web: A Survey. *SIGMOD Record*, 27(3):59–74, 1998.
27. Michael Grossniklaus and Moira C. Norrie. Information Concepts for Content Management. In *Proc. DASWIS 2002, Intl. Workshop on Data Semantics in Web Information Systems, WISE 2002*, Singapore, Republic of Singapore, December 2002.
28. K. Cheverst, N. Davies, K. Mitchell, A. Friday, and C. Efstratiou. Developing a Context-Aware Electronic Tourist Guide: Some Issues and Experiences. In *Proc. CHI 2000*, The Hague, September 2000.
29. Moira C. Norrie and Beat Signer. Switching over to Paper: A New Web Channel. In *WISE 2003, 4th International Conference on Web Information Systems Engineering*, Rome, Italy, December 2003.
30. Moira C. Norrie and Beat Signer. Information Server for Highly-Connected Cross-Media Publishing. *Information Systems Journal, Special Issue: The 15th International Conference on Advanced Information Systems Engineering (CAiSE'03)*, 2005.
31. Paul Dourish. What We Talk About When We Talk About Context. *Personal and Ubiquitous Computing*, 8(1), 2004.
32. Moira C. Norrie. An Extended Entity-Relationship Approach to Data Management in Object-Oriented Systems. In *Proc. ER'93, 12th Intl. Conf. on the Entity-Relationship Approach*, December 1993.
33. Swedish Institute of Computer Science, S-164 28 Kista, Sweden. *SICStus Prolog User's Manual*, 1995.
34. Adrian Kobler and Moira C. Norrie. OMS Java: A Persistent Object Management Framework. *L'Object*, 6(3/2000), November 2000.
35. Moira C. Norrie and Alexios Palinginis. Versions for Context Dependent Information Services. In *Proc. COOPIS 2003, Conf. on Cooperative Information Systems*, November 2003.
36. Bill N. Schilit. *A System Architecture for Context-Aware Mobile Computing*. PhD thesis, Columbia University, 1995.
37. Donald J. Patterson, Lin Liao, Krzysztof Gajos, Michael Collier, Nik Livic, Katherine Olson, Shiaokai Wang, Dieter Fox, and Henry Kautz. Opportunity Knocks: A System to Provide Cognitive Assistance with Transportation Services. In *Proceedings of UBICOMP 2004: The Sixth International Conference on Ubiquitous Computing*, Nottingham, UK, September 2004.
38. Anand Ranganathan, Roy H. Campbell, Arathi Ravi, and Anupama Mahajan. ConChat: A Context-Aware Chat Program. *Pervasive Computing*, 1(3), July-September 2002.
39. Amsterdam Realtime Project. http://www.waag.org/realtime/.