Interplay of Content and Context

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

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

Abstract. We examine general and abstract approaches to web engineering and context-awareness and how they interact with each other. This involves considering the appropriateness of approaches to context when used by a complex application such as a content management system, while, at the same time, presenting how a content management system can use context information to enrich its functionality. We show that the integration of such systems is feasible only if, in both fields, we make use of approaches based on strong information models. Last but not least, we show that the relationship between context engines and content management systems is not at all a one-sided client-server scenario, but rather a mutually important symbiosis.

1 Introduction

As the amount of available digital data continues to grow, vast quantities of information surround us in our everyday life. With the abundance of information that is accessible to us, the problem no longer is whether information is within reach, but rather whether the right information can be delivered in the right form to the right person at the right time. Hence, the value of information is no longer solely dependent on the quality of the information itself, but is influenced by other factors as well. 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. Even though content management systems were originally developed for the administration of large websites, they have proven to be so successful that a wide range of other applications use content delivery technologies nowadays.

At the same time, a shift in applications to mobile and ubiquitous information environments is clearly noticeable. Applications within this domain require highly situation-aware computing and it no longer suffices to simply present the information to the user in the right form. Depending on the user's situation, the information must be filtered according to what is important for the user. Information is no longer static, but highly dynamic as the situation of the user is altering continually according to an ever-changing environment. Context and context-aware computing have been recognised as key concepts in reaching this goal and a lot of research has been invested in this topic. Today, application-specific solutions and context frameworks exist, but a generic context engine with a semantic model for context is still lacking.

We have developed such a context engine for the management of context information that can be coupled with existing applications to augment them with the notion of context. We have also developed a content management system that supports the delivery of context-dependent content, but does not specify its own model for context. In this paper, we present the integration and interplay of these two aspects and components and how they complement each other. 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.

Sect. 2 presents an overview of related work that has been done in the field of context-aware computing in general and specifically in the area of content management systems. In Sect. 3 we then describe our context engine. Our content management system that is used to demonstrate the interplay of content and context is outlined at the beginning of Sect. 4 followed 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. 5. Finally, future directions and conclusions are given in Sect. 6.

2 Related Work

From the very beginning of the batch processing era, programs have been designed to produce and present an appropriate output to some given input. Although nowadays the input could be issued interactively and an application may even accept input from arbitrary sources, the principle of the black-box still holds. The output is completely determined by the given input. In contrast, context-aware applications also use implicit information hidden in the application's and user's environment [1], in addition to explicit input, to determine the output.

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 communities including philosophy, linguistics and social sciences. Later people in the field of computer science borrowed the term and applied it in various applications [2,3,4]. With the rise of ubiquitous and pervasive computing, context has strongly been associated with information extracted from the physical environment. Spatial information such as location, orientation and speed or environmental information such as temperature, light and noise level are commonly sensed context properties for this application domain.

Context-aware applications have been designed and implemented such as office and meeting tools [5], tourist guides [6,7] and specific fieldwork enhancements [8]. All those examples propose context models specifically tailored to each application domain and directly implemented in the application logic. Many common concepts and components had to be implemented in isolation for each application. At the same time, some researchers wanted to provide a reusable and helpful infrastructure and proposed general context frameworks [9,10].

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]. Further, [9] separates the actual sensor from the abstract context which allows the decoupling of 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 [11], 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. Nowadays, web-based applications represent a major and common class of applications as web technologies and infrastructure have become defacto standards for many application environments. In the field of web engineering, the first steps towards context-awareness were made by introducing adaptation concepts directly into hypertext models [12,13]. 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 [14], user data, usage data and environment data is distinguished. An extensive discussion on the field of AHS and related work can be found in [15]. We should mention here that user characteristics such as presented in AHS are not always necessarily describing context, but rather the user profile in general. For example, user preferences such as colours, styles and interests might be part of the current context, i.e. they contribute to the dynamic characterisation of an interaction situation, but this does not have to be the case.

AHS employs 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.

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 studied with respect to adaptation [16] and context-awareness [17,18,19]. 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 [16], delivery channel characteristics can be used to influence the hypertext and navigational model of the original WebML model [20]. 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.

In OMSwe [17], the database management system OMS Pro [21] is extended with versioning and basic content and presentation management facilities to empower a web development environment. All relevant information, from the development process to implementation and application data, is managed by the database system. 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. eXtensible Content Management (XCM) [22] takes the approach one step further and defines a full-strength content management system with well defined general information concepts. Based on this approach, we present here the interplay between this system and the general context engine that we developed. Details of XCM are given in Sect. 4.

3 Context Engine

In [11], we consolidated research in the field of context and context-awareness and presented a model for a generic context engine. The basic and abstract concept of context is influenced both by the application and framework requirements, as well as the problems presented by Dourish in [23]. In this section, we use an example to describe how our context engine works and how application-specific models can be specified.

Each entity of an application can potentially have a context. An application schema, for instance, could define the concept of a room. The context of a room could be described by its physical properties such as the temperature and the light intensity. For each point in time, the room's context is composed of the set of values over its context properties: e.g. `temperature=30` and `light=40`.

Obviously, the values 30 and 40 do not provide any information about the scales that are used. Moreover, a context engine supplies context information to many different applications, thus making a quantification even more important. We therefore use a type system to define concepts within the context engine.

The system distinguishes four kind of types. *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 `celsius` as a restriction of `real`. Restrictions of a type are expressed through a constraint that checks the validity of the values acceptable for the defined type.

```
btype celsius is real {
        constr: 'self(S), S > -273.16';
};
```

Our prototype system 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 schema. 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 `celsius` temperature, the value is first retrieved using the predicate `self/1` and then checked to be greater than −273.16.

Apart from the base types, the context model uses references to application-specific types to define and control application entities. Apart from a uniquely identifiable *application type*, the context engine does not deal with, and has no control over, the values composing an application entity. Thus an application reference type is sufficiently defined by the pattern `applicationID:typeID`. The third class of types are the *bulk types* which designate sets of values of a given member type. Finally, the model supports context types which define the composition of the context itself rather than its values. As a common composite type, *context types* are defined over a set of attributes of a given type. In the following example, we define a context type `ctx_temperature` with one attribute `temperature` of type `celsius`.

```
contextType ctx_temperature characterises app:physical_obj {
        temperature: celsius;
};
```

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 `physical_obj` defined in application `app`. Application types could be defined in an is-a hierarchy, designating a compatibility among them (not shown in this example).

After declaring the types, an application can create context elements and bind them to some application entity. In the following example we create the temperature context for `app:roomA32`, which is an instance of `app:room`, subtype of `app:physical_obj`.

```
context c_roomA32_temp: ctx_temperature describes app:roomA32
```

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 thermometer and a software sensor could 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 temperature sensor that is connected to the serial port of a computer and provides temperature context information based on the `ctx_temperature` context type. The last statement initialises the actual thermometer based on the given sensor driver, providing context information to the `c_roomA32_temp` context.

```
sensorDriver tempSensor(serial_port: integer): ctx_temperature;
sensor s_thermo1: tempSensor(1) provides c_roomA32_temp;
```

In addition to the sensor driver definition, an implementation is necessary. For our prototype, it is coded in SICStus Prolog [24] and bindings to Java are used in cases where the actual sensor offers a Java API. Examples of such sensors are software sensors that use information collected from the content management system. More details on such sensors will be given in Sect. 4. For a context-aware news application, we have developed a prototype sensor driver for the recognition of people in a room. It is based on the ARToolKit [25] augmented reality library and recognises people through special tags that are recognised by the ARToolKit in a stream from a small video camera.

Having presented our context engine, we now go on to explain in the next section how it can be used to make an existing application context-aware.

4 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) [22,18]. The current version of XCM is implemented in Java and based on OMS Java [26], an object-oriented database management system based on the Object Model (OM) [27], a model that integrates object-oriented and entity-relationship concepts. The original version of XCM supports only a rather simple implementation for context. We are currently working on the extension of XCM for support of the context engine that we have 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, a news article may have multiple variants that correspond to the different languages in which it is available. Multi-variant objects however 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. 1, a metamodel is displayed that shows how XCM represents multi-variant objects. The notation and semantics is that of the previously mentioned Object Model (OM).

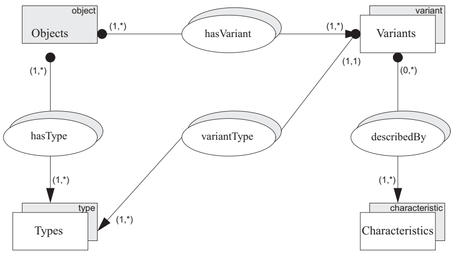

Fig. 1. Metamodel for XCM Multi-Variant Objects

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)`. 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 a news article can have one variant to produce HTML and another variant to display WML. 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 requestor. 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.

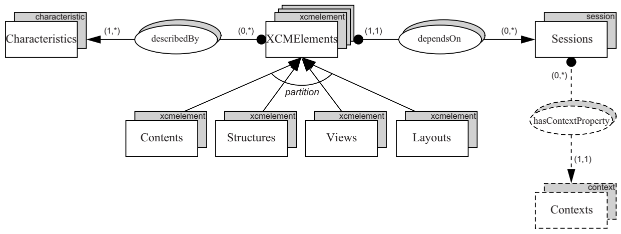

Fig. 2. Conceptual Model for the Context Binding

Figure 2 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 Prolog 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` its 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 ctx_browser {
        name: string;
        version: string;
};

contextType ctx_session characterises xcm:session {
        user: xcm:user;
        lang: language;
        browser: ctx_browser;
        host: ip;
};
```

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 best matching version is selected. 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 [17,28].

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 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.

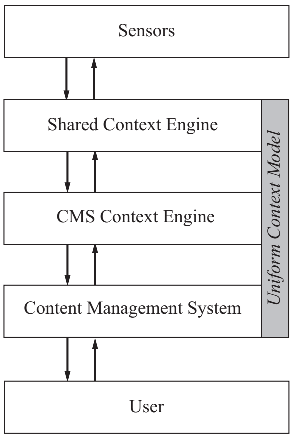

Fig. 3. Overview of the Architecture

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 3 shows a graphical representation of all components. At the bottom, the user is shown that communicates with the content management system through a browser requesting pages. 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.

5 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 describe 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 another context-aware application of an information environment.

In Sect. 3, we introduced the abstract concept of a sensor. A sensor is basically any piece of software or hardware that generates context values for a given context element. Sensors can be classified into two categories, depending on the environment that they target: *Physical sensors* acquire context information from the physical environment, e.g. from a room, the human body or a physical object. *Software sensors*, on the other hand, gather information from the virtual environment formed by the logic and data of an application. The content management system is such an application to which software sensors can be applied. As XCM is implemented in Java, the corresponding software sensors would be implemented in the same language. The binding of the sensors to the context engine is done through the mechanism of a sensor driver as mentioned in Sect. 3.

The concepts of interest are the same for both physical and software sensors. From the main areas described in [29], the *user environment* is of special importance for information environments. Whereas the physical part of the user environment (e.g. location and orientation of a person) is usually relatively easy to acquire with appropriate sensors, the mental state of a user is much harder to obtain. The content management system is able to provide very valuable context information in this area such as what the user is currently working on, which topics are of interest, how busy the user is and whether they are currently working on a precise topic or just browsing an information space.

Context information can be extracted from application data, content data, the local context model of the content management system or the user sessions. Some of this context information, e.g. the level and type of activity, is independent of the actual application data in the content management system and relies only on its metadata. Software sensors can be provided that extract this context information in a generic way. However, for the other 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. Imagine, for instance, a news application where individual news messages are categorised with respect to their topics. In some traditional content management systems that do not support customised data models, this could be modelled with a page for each topic containing a list of news messages in the form of paragraphs, links etc. and a page for each news message with more details. In this setup, it is not possible to extract the topics in which the user is currently interested. As we have described in Sect. 3, 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 news messages that the user is currently browsing and store corresponding entries in the context engine. In contrast to other context frameworks such as [9], as described in Sect. 3, we are also able to use application objects as context values through reference types. 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.

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 information environments and corresponding platforms that integrate multiple context-aware applications for a physical location or virtual community. We have implemented a context-aware news application for such an information environment that uses a content management system for the delivery of information and adapts according to the people in a room. Currently, we are working on the extension of this news application towards the inclusion of virtual rooms and the integration of the context engine described previously.

Another example are community awareness applications that visualise the presence and activities of other users. This is context information of the user. A content management system as part of an information environment of a community could provide some of this context information, for example, for an application such as the context-aware instant messaging and chat application described in [30]. It visualises a list of users that are present along with their current state (present, busy, free-for-chat etc.). Another interesting example is the area of Computer-Supported Cooperative Work (CSCW) or collaborative systems in general. In this context, the content management system could also provide context information for a community of users, such as users working on specific topics, rather than just for individuals. This information could be displayed by other context-aware applications or the context management system itself.

6 Conclusions and Future Work

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 showed 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 showed how this crucial and complete information provides context and how we use XCM as a software sensor to the context engine.

We are currently working on a platform for a context-aware information environment (SOPHIE) that combines features from pervasive computing and information systems. The SOPHIE platform is based on content management technologies and context engines and represents an important step forward in the integration and extension of these technologies to cater for information delivery and interaction in environments that span the physical and digital worlds and involve multiple interrelated users and devices.

References

1. Lieberman, H., Selker, T.: Out of Context: Computer Systems that Adapt to and Learn from Context. IBM Systems Journal **39** (2000)
2. Brown, P.J.: The Stick-e Document: a Framework for Creating Context-aware Applications. In: Proc. EP'96, Palo Alto. (1996)
3. Dey, A.K., Salber, D., Abowd, G.D., Futakawa, M.: The Conference Assistant: Combining Context-Awareness with Wearable Computing. In: ISWC. (1999)
4. Chen, G., Kotz, D.: A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College (2000)
5. Want, R., Hopper, A., Falc o, V., Gibbons, J.: The Active Badge location system. ACM Transactions on Information Systems **10** (1992)
6. Abowd, G., Atkeson, C., Hong, J., Long, S., Kooper, R., Pinkerton, M.: Cyberguide: A mobile context-aware tour guide (1997)
7. Davies, N., Mitchell, K., Cheverst, K., Blair, G.: Developing a Context Sensitive Tourist Guide (1998)
8. Ryan, N.S., Pascoe, J., Morse, D.R.: Enhanced Reality Fieldwork: the Context-aware Archaeological Assistant. In Gaffney, V., van Leusen, M., Exxon, S., eds.: Computer Applications in Archaeology 1997. British Archaeological Reports, Oxford (1998)
9. Salber, D., Dey, A.K., Abowd, G.D.: 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 (1999)
10. Hess, C.K., Ballesteros, F., Campbell, R.H., Mickunas, M.D.: An Adaptive Data Object Service Framework for Pervasive Computing Environments (2001)
11. Belotti, R., Decurtins, C., Grossniklaus, M., Norrie, M.C., Palinginis, A.: Modelling Context for Information Environments. In: Ubiquitous Mobile Information and Collaboration Systems (UMICS), CAiSE Workshop Proceedings. (2004)
12. Brusilovsky, P.: Methods and Techniques of Adaptive Hypermedia. User Modeling and User-Adapted Interaction **6** (1996)
13. Brusilovsky, P.: Adaptive Hypermedia. User Modeling and User-Adapted Interaction **11** (2001)

14. Kobsa, A., Müller, D., Nill, A.: 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)
15. Wu, H.: A Reference Architecture for Adaptive Hypermedia Systems. PhD thesis, Technical University Eindhoven (2002)
16. Ceri, S., Daniel, F., Matera, M.: Extending WebML for Modeling Multi-Channel Context-Aware Web Applications. In: Proc. MMIS'2003, Intl. Workshop on Multichannel and Mobile Information Systems, WISE 2003. (2003)
17. Norrie, M.C., Palinginis, A.: Empowering Databases for Context-Dependent Information Delivery. In: Ubiquitous Mobile Information and Collaboration Systems (UMICS 2003), Klagenfurt/Velden, Austria (2003)
18. Grossniklaus, M., Norrie, M.C., Büchler, P.: Metatemplate Driven Multi-Channel Presentation. In: Proc. MMIS 2003, Intl. Workshop on Multi-Channel and Mobile Information Systems, WISE 2003, Roma, Italy (2003)
19. Perkowitz, M., Etzioni, O.: Towards adaptive Web sites: conceptual framework and case study. Computer Networks (1999)
20. Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): A Modeling Language For Designing Web Sites. Computer Networks (2000)
21. Norrie, M.C., Würgler, A., Palinginis, A., von Gunten, K., Grossniklaus, M.: OMS Pro 2.0 Introductory Tutorial. Inst. for Information Systems, ETH Zürich. (2003)
22. 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 (2002)
23. Dourish, P.: What We Talk About When We Talk About Context. Personal and Ubiquitous Computing **8** (2004)
24. Swedish Institute of Computer Science S-164 28 Kista, Sweden: SICStus Prolog User's Manual. (1995)
25. Kato, D.H., of Washington, H.L.U., of Canterbury, H.L.U.: Artoolkit. http://www.hitl.washington.edu/artoolkit/ (2003)
26. Kobler, A., Norrie, M.C.: OMS Java: Lessons Learned from Building a Multi-Tier Object Management Framework. In: Proc. OOPSLA'99, Workshop on Java and Databases: Persistence Options. (1999)
27. Norrie, M.C.: An Extended Entity-Relationship Approach to Data Management in Object-Oriented Systems. In: Proc. ER'93, 12th Intl. Conf. on the Entity-Relationship Approach. (1993)
28. Norrie, M.C., Palinginis, A.: Versions for Context Dependent Information Services. In: Proc. COOPIS 2003, Conf. on Cooperative Information Systems. (2003)
29. Schilit, B.N.: A System Architecture for Context-Aware Mobile Computing. PhD thesis, Columbia University (1995)
30. Ranganathan, A., Roy H. Campbell, A.R., Mahajan, A.: ConChat: A Context-Aware Chat Program. Pervasive Computing **1** (2002)