Table of Contents

4 Context-Aware Data Management

The necessity for context-aware computing is well documented by the existence of numerous applications that exhibit adaptive behaviour and by framework systems that attempt to provide comprehensive support for the challenges encountered in the realisation of such applications. Some examples of applications and systems in the domains of mobile, ubiquitous and pervasive computing as well as web engineering have been analysed in Chapter 2, in terms of their notion of context and their adaptation capabilities. As demonstrated by these approaches, context-awareness has been tackled on a large scale and in very different ways. Most solutions, however, focus on context representation or on enabling adaptation at the level of the application logic. The impact context-awareness has on information systems in terms of data management and querying is often neglected.

Many frameworks for context-awareness introduce the concepts of variants for aspects of a system that need to show adaptive behaviour under different context states. In contrast to context-aware data management in general, the management of alternative versions of data has been addressed by many solutions in the past. As presented in the previous chapter, these approaches were originally designed to cope with temporal and engineering data or with software configurations. Apart from alternative versions, most of these systems also provide revisional versions to manage the evolution of applications. The same requirement is also present in the domain of web engineering where keeping track of the development process of web-based systems has repeatedly been recognised as a key challenge. For example, in their paper introducing OOHDM, Schwabe et al. [251] state the following.

> "[. . .] *design decisions should be recorded and traced backward and forward in the development process.*"

Instead of proposing revisional versions, however, OOHDM addresses this requirement at a much coarser level by structuring the design process based on a sequence of models that build on each other and thus enable developers to move back and forth in the process. Nevertheless, documenting development and evolution is important in any complex system, regardless of the proposed solution.

Both alternative and revisional versions have been introduced to address specific requirements. Therefore, they have quite different properties in terms of how they are used and perceived. To illustrate this, assume that two kinds of actors interface with the data management system. Users with the developer role design the application and provide the content. In contrast, users with the client role query the content managed by the system. While we do not claim that this simple role model is of any relevance to real application scenarios, it is sufficient to exemplify the different nature of revisional and alternative versions. As revisional versions are intended to support the implementation of an application by keeping track of its evolution, they are usually visible to the developer only. Clients of an application are not aware of their existence as they will simply see the most recent data. Alternative versions cater for the management of variants of the same object. Therefore the definition of such default behaviour in analogy to revisions is often not straightforward and the system has to rely on the client to specify which variant they require. As a consequence, variants are not transparent to the users, as potentially they can witness how such objects change according to the specification they provide along with the query. Further, when users having the client role are given permission to create and update data in the system, the different perception of revisions and variants becomes even more obvious. Whereas revisions can be created automatically and behind-the-scenes, the creation of variants requires the client to specify a description that will later be used for retrieval. Another way of looking at the different characteristics of revisions and variants is the nature of queries that involve each kind of version. In contrast to revisions that are most likely to be used off-line to analyse the development of an application, variants are used in an on-line fashion as an integral and active part of the production system.

Many existing systems have managed to successfully integrate revisional and alternative versions and offer complete and comprehensive support for the domain for which they were developed. However, none of these systems have been developed to support context-aware data management based on alternative versions. As context-aware computing differs from existing domains in terms of the requirements in the area of data representation and query processing, the application of these systems in this field is not straightforward. In this chapter we present our approach to context-aware data management that builds on the existing version models and extends them to cope with these new requirements. As motivated earlier, object-oriented systems are better suited for the implementation of version models and our version model has therefore been defined as a refinement of an object-oriented data model. To provide an understanding of this object-oriented data model and its functionality, it will be introduced based on a metamodel detailing how the objects defined by it are represented. As a part of the work presented in this thesis, this metamodel has been extended to allow revisional and alternative versions of objects to be managed. In combining revisional and alternative versions into one model, our approach reflects the convictions that we have set forth in Section 2.5. Even so, it has to be accepted that revisional versioning is an already well researched area and little is to be gained from a repeated discussion of this subject. Rather than expanding these concepts that have been put forward in existing systems, our version model uses and integrates them. We have therefore decided to place the emphasis of the following presentation and examples on the concept of context-dependent variants and how they have been combined with revisions. As a consequence, it is in this part of our version model for context-aware data management where we see the original contribution of our work.

In this chapter, we also introduce the notion and representation of context that is assumed in the scope of this thesis. Based on this representation of context, we show how data objects are annotated to characterise the situation in which they are appropriate. These characterisations are then used by the query processor when evaluating queries in a given context. For each object, the version graph is analysed and the variant that best matches the current context is used for subsequent query processing. The properties and functioning of this matching algorithm are also be discussed in detail, as it is a major element of context-aware data management. Finally, the implementation within the framework of an object-oriented database management system is presented. The chapter concludes with a discussion of our version model relating it to the requirements of context-awareness and comparing it to previously defined models.

4.1 Context, Context Space and Context State

In recent years, a multitude of models has been proposed that address the management and representation of context. Typically, these systems suffer from limitations that are rooted in either of two design decisions in the development of the context model. On the one hand, many models have evolved from specific application domains and therefore assume a notion of context that is relevant to that field. Targeting a context model to a specific application domain often leads to a reduced generality, as such models tend to restrict themselves to a fixed set of context dimensions. For example, a common characteristic of the models discussed in the preceding chapter is that most of them classify context into user, device and environmental factors and predefine the dimensions that are required for each of those classes. On the other hand, some models dictate how context is represented and stored, as a consequence of the technologies that have been used to implement them. Hence, representation of context is often tightly coupled to the platform that is used to store and process contextual information. Solutions range from collections of simple attribute values to complex approaches that represent context data in terms of database objects, or are based on class hierarchies in an object-oriented programming language. The integration of a context model into a general-purpose database management system can only be justified if the model is free from these two limitations. Therefore, it is our goal to establish a set of universal primitives that will serve the purposes of several different usage scenarios.

The first challenge that has to be addressed when defining such a general context representation is taking care not to impose a specific notion of context onto client applications. There have been many approaches that have proposed models that are general in the sense that they can be configured by the client application to use its notion of context, i.e. these models expect the application to define what context is. While these models can be configured by the client, most of them still have some built-in assumption about the meaning of certain context dimensions in terms of context classifications or ontologies. Often these factors lead to situations where different context dimensions are implicitly treated quite differently. Clearly, this is not desirable for a general context representation which needs to refrain from any predefined assumption about the notion of context used. Therefore, it is even necessary to go one step beyond the current approaches and completely abandon any presumptions about the meaning of context.

As a consequence, context in our approach is defined in terms of its effect rather than its meaning, to guarantee that the system built on top of it is free from any hidden or implicit assumptions. Applied to the situation of a data management system, this means that the effect of context on query evaluation needs to be specified. Thus, in our system context is seen as a set of additional parameters that in combination with the set of parameters specified by the query determines the result computed by the query processor. However, the roles of these two sets of parameters are quite different. Query parameters are explicit parameters in the sense that they were defined by the client of the data management system. Therefore, these parameters constitute a specification that the query processor has to follow exactly. In contrast, context parameters are implicit parameters as they are not specified by the client as part of the query but rather defined by the situation in which the query is sent to that data management system. As an implication of this, the role of context is not specification but rather a factor that is used to refine the results of query evaluation. Due to this notion of context, implicit parameters have to be considered as optional since they refine the functioning of the query processor rather than providing a specification for it to follow. Even in the absence of context information, query evaluation will return a fully defined result that is computed using a previously specified default representation of each object involved. An overview of this notion of context and its impact on the database system is presented in Figure 4.1.

As shown, a query can be abstracted in terms of an n-ary function q that is given as input to the query processor of the data management system. In a traditional database system, the result r of the query evaluation only depends on the set of parameters $\{x_1, x_2, \ldots, x_n\}$ that are specified in the query. As an extension, a context-aware query processor additionally considers a context $c(c_1, c_2, \ldots, c_m)$ during the evaluation of the query. Later in this section, we will come back to how this notion of context is represented exactly.

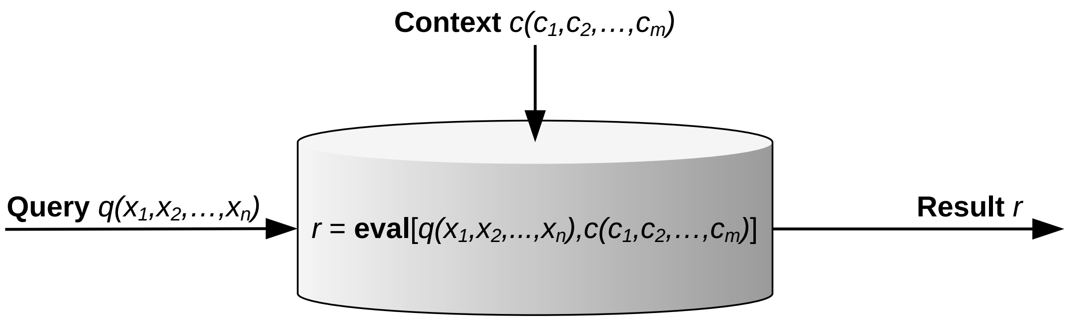
Context $c(c_1,c_2,\ldots,c_m)$
Query $q(x_1,x_2,\ldots,x_n)$
r = **eval**$[q(x_1,x_2,\ldots,x_n),c(c_1,c_2,\ldots,c_m)]$
Result r

Figure 4.1: The role of context

The open notion of context introduced by our approach successfully avoids the first of the two pitfalls discussed at the beginning of this section. As it defines context in terms of its effect rather than is meaning, it is free from any assumptions that are specific to a given application domain. Nevertheless, it is important to establish a common understanding between the client application and the data management systems which context dimensions have to be handled in a given application domain. Context data may come from a variety of sources such as direct sensor access as well as context reasoning, inference or augmentation. As presented in Chapter 2, numerous powerful solutions for context processing exist already and the data management system presented in this thesis does therefore not offer any functionality to address these concerns. Instead, it is assumed that context processing is done outside the system by a context component specific to the application in question. In the following, we assume that a given set NAMES exists that contains the names of all valid context dimensions. Based on this given set, the number of possible dimensions for context values is limited by the concept of a *context space* as specified in Definition 1.

Definition 1 (Context Space)**.** A *context space* represented by S denotes which context dimensions are relevant to an application of the version model for context-aware data management. It is defined as

$$S \;=\; \{name_1, name_2, \ldots, name_n\}$$

such that $\forall\, i : 1 \leq i \leq n \Rightarrow name_i \in \mathsf{NAMES}$ and therefore $S \subseteq \mathsf{NAMES}$.

Note that a context space does not define a domain for the data value of the corresponding context dimension. As we see later, the decision to have a loosely typed context representation is motivated by the fact that it permits more flexibility in query evaluation.

Although a context space expresses a shared understanding about the context dimensions that exist in an application domain, it is of equal importance to also share a common representation of context. There are two possible solutions to address the problem of an application-specific representation of context. The first solution would be to define a comprehensive representation of context that embraces all possible notions of context that any current or future applications require. The other solution would be to build such a context representation based on an elementary approach that only provides the basic building blocks necessary to ensure that all existing solutions can be mapped to it. For several reasons, we have decided to follow the latter approach and rely on an elementary context representation. Our survey of existing context-aware applications has shown that most of these systems do not employ sophisticated context models but rather rely on collections of attribute values to capture context. Further, we believe that a basic yet flexible context representation is a better candidate for integration into a database management system. While a comprehensive representation would be constructed as a unification of existing approaches, an elementary representation is defined based on a metamodel of the concepts that are common to all known solutions. Hence, the comprehensive representation cannot cope with requirements that have not yet surfaced, whereas the basic representation is better equipped to do so.

In our version model, we have decided to use a context representation that is based on $\langle name, value\rangle$ pairs. Even though such tuple values constitute a very simple technique to capture context, most of the context models presented in Chapters 2 and 3 use this representation. This simple solution both fulfils the requirement of a basic representation and, at the same time, is already widely used. Again, we assume the existence of a given set VALUES that contains all possible values for the context dimensions defined by the NAMES given set. Based on these given sets, *context values*, as specified by Definition 2, form the basic building blocks.

Definition 2 (Context Value)**.** A *context value* denoted by c is defined as a tuple $c = \langle name, value\rangle$ where $name \in \mathsf{NAMES}$ and $value \in \mathsf{VALUES}$.

The equality of two context values c_i and c_j is defined as $c_i = c_j \Leftrightarrow name_i = name_j \wedge value_i = value_j$. Each context value captures one context dimension by associating it with a data point. A *context* as specified in Definition 3 aggregates individual context values into a collection of context values. Within a context, every dimension of the context space can be associated with at most one value, i.e. a context that contains multiple context values for the same dimension is not valid.

Definition 3 (Context)**.** $C(S)$ denotes a *context* for a context space S and is represented as a set of context values

$$\begin{aligned} C(S) &= \{\langle name_1, value_1\rangle, \langle name_2, value_2\rangle, \ldots, \langle name_m, value_m\rangle\} \\ &= \{c_1, c_2, \ldots, c_m\} \end{aligned}$$

such that $\forall\, i : 1 \leq i \leq m \Rightarrow name_i \in S$ and $\forall\, c_i, c_j \in C : c_i \neq c_j \Rightarrow name_i \neq name_j$.

Based on this definition of context, it is now possible to revisit the role of context illustrated in Figure 4.1 and specify how context is handled and represented in our approach. In contrast to traditional data management systems that do not feature a system state to influence the operation of the system, a context-aware data management system is not stateless. As mentioned above, in such systems, context information is considered during the evaluation of context-aware queries. In order to fully support context-aware data management, it is therefore necessary to extend the information system with the notion of a context state. From a general point of view, this integration of context into the information system itself represents a move away from the paradigm of a stateless towards a stateful data management system. Definition 4 explains the understanding of the term as used in our approach.

Definition 4 (Context State)**.** A *context state* denoted by $C_\star(S)$ is a special context, where $\forall\, name \in S : \exists\, \langle name, value\rangle \in C_\star(S)$.

In contrast to the concept of a context, a context state $C_\star(S)$ requires the presence of a context value for each dimension of S. However, if the value of a context dimension is unknown this can be expressed by a context value containing $\perp$ as value. Later in this section, we discuss how the context state can be influenced to reflect the current context of the client application. However, before going into these details of dynamic query processing, we continue with presenting the static organisation of our version model first.

4.2 An Object-Oriented Version Model

The version model for context-aware data management has been developed based on *OM* [200, 201], an extended E/R model for object-oriented data management. Even though the concepts introduced by our version model are universal and could also be applied to other data models such as the relational model, we have chosen to base it on one concrete model. The choice of the object-oriented OM model is motivated by the following reasons. OM has been designed to be semantically expressive, making it a powerful tool for database development. In order to do so, it features concepts that are absent in other models. Therefore, OM can encompass less expressive models. Such models can either be represented in OM based on their metamodels or by limiting OM to a subset of its concepts to obtain a model with equivalent semantics. Finally, OM itself has been specified completely in terms of metamodels expressed in OM. These metamodels constitute a firm basis for specification of the extension of OM for context-aware data management. Further, since most data management systems that provide support for OM are implemented based on these metamodels, they also serve as the starting point of the integration of the version model into one of these systems. It is therefore useful to give a brief overview of the main distinguishing features of OM before presenting how it was extended to support both revisional and alternative versions.

4.2.1 The OM Data Model

As the OM data model is an integration of object-oriented concepts into the well-known E/R model, it also builds on the notions of application entities and the relationships that exist between them. However, in contrast to the E/R model where there is some dispute whether entities represent entity types or entity sets, OM introduces a clear separation between the typing and the classification of entities. This distinction is achieved using a two-layered model. On the lower level, types describe the representation of entities, whereas the upper level captures the semantics of entities, based on the concept of collections. As OM is an object-oriented data model, all data and metadata is represented in terms of objects. Each object is defined by at least one object type that specifies the attributes and methods that its instances will have. Object types can form type hierarchies that are built using inheritance between supertypes and subtypes. In contrast to most existing object-oriented systems, OM supports the concept of multiple inheritance, i.e. a subtype may be defined as a specialisation of two or more direct supertypes. Apart from multiple inheritance that is defined statically, OM also features multiple instantiation enabling objects to dynamically gain or lose types that have either been defined along parallel inheritance paths in a type hierarchy or are not related through inheritance at all.

Based on the type model, a classification model uses collections to define flexible semantic groupings. Each collection has a member type that governs which objects can be contained in the corresponding collection. As it is not required to define a collection for each type, and it is possible to have multiple collections with the same member type, the typing layer and the classification layer are almost independent of each other. Similar to types, collections can build collection hierarchies based on the notion of supercollections and subcollections, to represent specialisations and generalisations of classification concepts. Again, a subcollection can have multiple supercollections and, vice versa, a supercollection can have multiple subcollections. Collections can either be sets, bags, rankings or sequences, depending on whether they can contain duplicates and if they are ordered or unordered. For example, a set is an unordered collection with no duplicates in the algebraic sense, whereas a sequence is an ordered collection that can contain duplicates. In contrast to a ranking that has an order but cannot contain duplicates, a bag is defined to be unordered with duplicates. Similar to multiple instantiation on the typing level, multiple classification permits an object to be a member of any collection that matches one of its types or a supertype of its types.

So far we have only described the concepts provided by the OM model to represent and classify entities, so-called objects. However, OM also features a relationship concept that is more powerful than the one known from the classical E/R model. Relationships in OM are represented by bi-directional associations defined in terms of a source and a target collection. Associations are a first-order concept and are defined as n-ary collections with $n > 1$. Much of the expressiveness of OM stems from the set of constraints it provides to control several aspects of a data model. Some constraints such as the relationships between subtypes and supertypes as well as subcollections and supercollections have already been discussed. In addition, the OM model also features cardinality constraints that govern the participation of objects as the source or target of an association. As in other extended E/R models, these cardinalities are specified in terms of a minimum and maximum value that expresses the number of objects to which an object can be linked. These so-called integrity constraints are complemented by classification constraints that control the membership of objects in collections. For example, the subcollection relationship between two collections can be declared to be either equal, strict or total. An equal constraint ensures that both collections contain the same elements in the same number and order. To assure that a sequence collection is a subsequence in terms of a segment of its supercollection it can be declared as a strict subcollection. Finally, when a bag must contain all occurrences of elements contained in its supercollection, it is defined to be a total subcollection. Additional classification constraints deal with collection families, i.e. a collection with either multiple supercollections or subcollections. For example two or more subcollections of a collection can be defined to be disjoint or to form a cover or a partition of their supercollection. Vice versa, if a collection has two or more supercollections, it can be defined to be the intersection of those collections. Finally, the last class of constraints, evolution constraints, monitor how a database grows and changes over time. It is, for example, possible to specify which types an object can gain and lose in the future, to limit the scope of object evolution.

Data models are specified in OM either by using its graphical notation or the textual data definition language. This data definition language is a subset of the *Object Model Language (OML)* [182] which also contains a data manipulation and a query language. The query language is based on a collection algebra that defines a set of operators that manipulate and process collections as well as associations. Apart from being used for data definition, manipulation and querying, the union of the three languages contained in OML also serves as a declarative object-oriented implementation language for object methods, triggers and database macros. While many conceptual models have no direct platform support, a family of data management systems [168, 296, 167] has been built that allow OM data models to be implemented directly. These platforms cover a wide variety of requirements, such as rapid prototyping, as well as productive operation both as light-weight in-memory and server databases. During its existence, the OM model has been extended a number of times to address the requirements of special application domains. These projects have led to a temporal data management system [270], a next-generation file system [228], and an extension of the model that is capable of role-based modelling of interactions in databases [219] to capture access control, context-aware and proactive operations.

As mentioned before, one reason for choosing the OM data model is the fact that it is capable of capturing its complete specification in terms of metamodels. This level of expressiveness has been the reason why most of the systems providing support for OM have been built based on this metamodel. As a consequence, they also represent and manage the entire metadata of a database in terms of objects, collections and associations. One advantage of this approach is the fact that the same language can be used to define, manipulate and query both data and metadata. Another benefit is that the implementation of extensions to these systems is relatively straightforward, as new system objects can be introduced at any time. An example of such a metamodel is given in Figure 4.2 using the graphical notation of the OM data model.

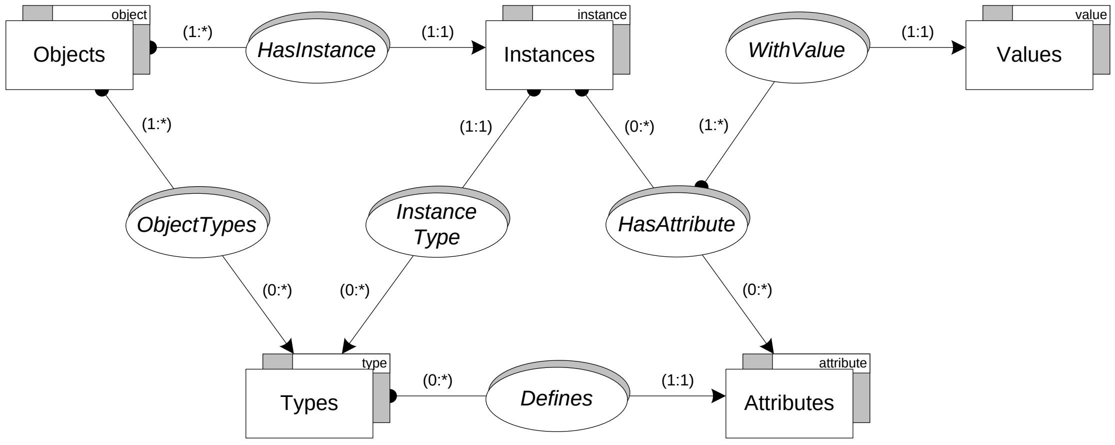

Figure 4.2: Metamodel of an object in OM

The model shows the definition of an object in OM. In the upper left-hand corner the collection Objects with member type object is depicted. It contains all objects that exist in a database at any point in time. The fact that OM supports multiple instantiation is captured by the fact that each object is linked to one or more instances through the *HasInstance* association. Both objects and instances are related to the Types collection through associations *ObjectTypes* and *InstanceType*, respectively. Whereas an object can be linked to one or more types as expressed by the cardinality (1:*) as a consequence of multiple instantiation, an instance is bound to exactly one type as indicated by the (1:1) constraint. However, not all constraints that are required to ensure the validity of an object can be expressed in the graphical notation. For instance, all types referenced by the instances of an object have to be referenced by the object itself. Also, an object has to be linked to an instance of each type with which it is associated. Hence, the number of associated types and instances has to be the same. These additional constraints are captured by the following condition given in OM collection algebra.

$$\mathit{ObjectTypes} = \mathit{HasInstance} \circ \mathit{InstanceType}$$

The left-hand side of the condition represents all links between objects and their types as a set of pairs. Analogously, the expression given on the right-hand side is also a set of pairs consisting of an object and a type. However, it relates objects to all types of the instances of these objects. As bi-directional associations are relations in the mathematical sense, this is achieved by using the composition (∘) of the *HasInstance* and *InstanceType* association.

As can be seen in the figure, each type contained in collection Types defines a set of zero or more attributes. Type attribute specifies how attributes are represented in the system. For each attribute defined by a type, an instance of that type has to specify a value. This relationship is captured by the *HasAttribute* association between collections Instances and Attributes. While it is not possible in OM to attach attributes to relationships, it is possible to define complex n-ary relationships by combining multiple binary associations. In contrast to other proposals for relationships of a higher degree, this approach has the advantage that it defines an order among the associations. In the given model, *HasAttribute* is the primary relationship, while the *WithValue* association, that links a value to a pair of an instance and an attribute, depends on it. As with objects, instances and types, the constraints between instances, types and attributes cannot be completely specified in the graphical model. To express that an instance has to specify a value for each attribute defined by its type, the condition given below is necessary.

$$\mathit{HasAttribute} = \mathit{InstanceType} \circ \mathit{Defines}$$

Again, the condition is expressed using a composition of two associations.

4.2.2 Integrating Revisions and Variants

To support revisional and context-aware data, the metamodel presented in Figure 4.2 has been extended with concepts that allow objects to be versioned. Inspired by the version models discussed in the previous chapter, the notion of a version has been refined to be either a revision or a variant of an object. Each of the new concepts represents one dimension of our version model. Revisions capture the evolution of the database over time. While they document how an object has been updated and revised, they are not intended to represent temporal data. From the three notions of time—transaction, valid and user-defined time—presented earlier, only transaction time is supported. Each time a revision of an object is created, it is marked with a timestamp ts representing the current logical or physical time. Therefore, revisions capture the time when a change has been recorded in the database. As the sequence of timestamps $T_{obj} = \{ts_1, ts_2, \ldots, ts_n\}$ assigned to the revisions of an object grows monotonically, it imposes a total order on the set of revisions. Revisional versions of an object o can therefore be identified based on a tuple $\langle oid, ts \rangle$ consisting of an object identifier and a timestamp that indicates when the corresponding version was created. The revision selected by the given tuple is a revision of the object with identifier $oid_o = oid$ that has the timestamp satisfying $max(\{ts_o | ts_o \in T_{obj}$ and $ts_o \leq ts\})$. As most queries to a database management system will request current data, a special revision is the one with the timestamp $ts = max(T_{obj})$. We will refer to this revision as the latest revision of an object. Note that the status of latest version will be assigned to different revisions over time as the database is updated. The concept of latest revision is therefore only meaningful at a fixed point in time.

The second dimension of our version model provides support for context-dependent data. Similar to the engineering databases and software configuration systems presented in the previous chapter, our model relies on variants to represent context-dependent alternatives of the same data. Different variants of an object are distinguished using the concept of a variant context as given in Definition 5 that describes in which situation the corresponding variant is an appropriate representation of the object.

Definition 5 (Variant Context)**.** A *variant context* for the context space S is a special context denoted by $C_v(S)$.

In the following, the context values contained in $C_v(S)$ are also called either the properties or the characteristics of a variant. Whenever a variant of an object is created, a set of context values characterising the new variant has to be given explicitly by the creator. The context defined by these properties has to be different from all variant contexts of existing variants of the same object.

In contrast to revisions, addressing a specific variant of an object is not straightforward. Not unlike revisions, variants are also identified by a tuple $\langle oid, C(S) \rangle$, but in this case, it consists of an object identifier and a context. While the mathematical properties of the timestamp sequence used to describe revisions have rendered the selection of the desired revision trivial, there is no order relation between the properties of two variants. To select the desired variant, the system has to match the given context $C(S)$ to the variant context $C_v(S)$ specified by each variant of the object. As this process is rather complex, we defer the detailed presentation of this matching algorithm to Section 4.3 that discusses query processing. A concept similar to the one of latest revision for revisional data is the notion of the main derivation as presented in the previous chapter. As shown in existing systems, it is useful to designate one variant of an object as its default variant to cope with the situation where the matching of a variant specification to the variant descriptions leads to ambiguous results or no context information is available. In some systems, the status of default variant can be reassigned to different variants of an object during system evolution. In our model, we have however decided not to offer this functionality, as we believe it should be the developer's choice at design-time to specify the default representation of the created data.

If both revisional and alternative versions are to be supported by the version model, both dimensions characterised above have to be combined. It is important to note that the two dimensions are independent of each other as it would be possible to construct systems that only support one of them. This orthogonality is further supported by the fact that both dimensions evolve along different axes. While revisions progress along the time axis, alternatives grow along the variation axis. Taken alone, both axes are one-dimensional, and, consequently, a two-dimensional version space will result when they are combined. Identical to most existing approaches, our model uses a version graph to organise this two-dimensional version space. Although, in principle, there are several different possibilities as to how this version graph can be defined, it is the requirements of an application domain that finally shape the concrete layout of the graph. For the following reasons, we believe that existing solutions cannot be applied unaltered to the domain of context-aware data management. First, the internal organisation of the version graph depends directly on the nature of the expected queries, as certain graph structures will favour some types of queries over others. Further, some conditions presented above, such as the uniqueness of properties among the variants of an object, are closely related to the intended use of the model. Ensuring such constraints can be simple in one graph structure while cumbersome and difficult in another. Finally, there are some questions arising from the combination of revisions and variants that need to be addressed based on the requirements of the application domain. For instance, the scope of revisions and variants has to be clearly defined. While in the one-dimensional case the scope of a version was the entire object, in the combined case there is a choice. A revision could either be a version of the entire object or of a variant of the object. Vice versa, alternatives could be defined in the scope of a single revision or of the entire object.

In the following, we will discuss the advantages and disadvantages of defining the scope of revisions and variants one way or the other. At the same time, this discussion of possible graph structures will also serve as a motivation for the approach that we have chosen for our model. As shown in Figure 4.3, three candidate solutions to organise the version graph can be obtained by simply enumerating all possible definitions of the scope of revisions and variants. The version graph resulting if both revisions and variants are defined to have the entire object as their scope is depicted in Figure 4.3(a). As can be seen in the figure, the object is linked to a set of round nodes representing the revisions and a set of triangular nodes representing the variants. The different versions of the object are in turn associated with its instances shown as small circles at the bottom of the figure. Instead of linking each version with each instance, we have chosen to introduce the artificial concept of an instance group, represented as a dashed oval, to render the figure more legible. The concept of the latest revision and the default variant are illustrated in the figure using bold arrows linking the object to revision ts_3 and to variant P_2. The labels of revisions and variants are meant to suggest that they are distinguished based on timestamps and properties, respectively. This organisation of the version graph provides direct access to both revisions and variants. However, there are also several limitations implied by structuring the graph in this way. Consider, for example, a query requesting the latest revision of variant P_3. To answer this query, both instance groups linked to P_3 first have to be retrieved. Then, starting from ts_3, the sequence of revisions has to be accessed backwards until one of the two previously retrieved instance groups is encountered along this path. In other words, the query is evaluated by computing the intersection of the revision and the variant dimension. Another difficulty arises from queries that need to find out what variants of an object exist for a given revision. Such queries occur when an application wants the variant of an object that matches context $C(S)$ at time $t < ts_2$. To find the set of variants that are eligible at time t, the sequence of revisions has to be traversed in forward order until $t \geq ts_2$, to collect all instance groups that have been created up to that point in time. Then, as before, these instance groups have to be cross-referenced with the variant dimension to find the variants P_1 and P_2.

The problem of having to search all revisions or variants to find the one that matches a version group retrieved through the other access

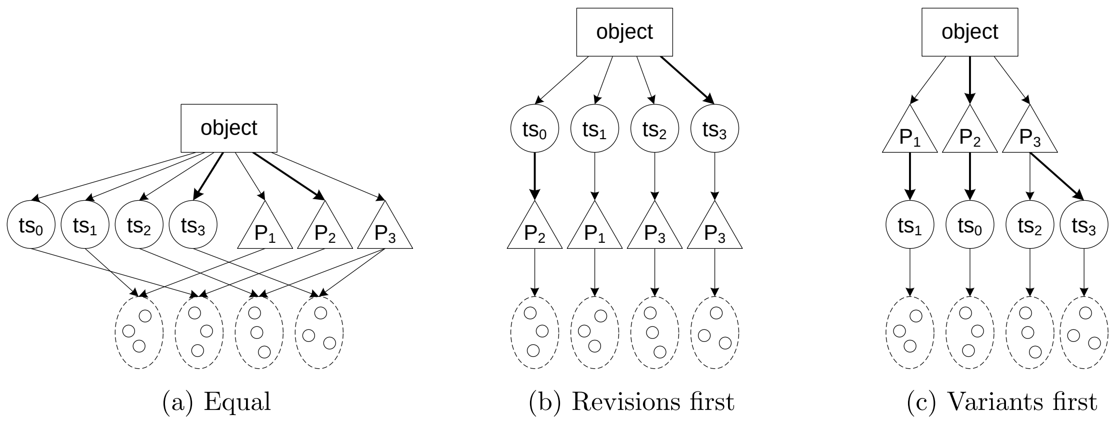

Figure 4.3: Different organisations of the version graph

path can be remedied by setting the equal status of both concepts aside. This implies, however, that one of the two concepts is dependent on the other and thus such a version graph organisation will only favour certain queries. Figure 4.3(b) depicts the same object as Figure 4.3(a) structured in a graph that gives priority to versions over variants. As can be seen from the figure, the processing of the query for the variants of the object at time $t < ts_2$ is now straightforward as the variants are directly linked to the revisions. However, computing which variant of the object matches a specified context $C(S)$ is now far more complex, as the set of variants is no longer accessible directly and has to be constructed first. Another issue that arises from this organisation of the version graph is the fact that a single variant can now be represented by more than one node, as has happened with variant P_3 in the figure. As the properties describing the variant would typically be stored in this node, the question whether these characteristics of a variant can also change over time has to be addressed. We believe that such functionality is not required, as properties are essentially metadata that is used exclusively by the system. However, analogous to temporal databases, where minor errors can be corrected without changing the history of an object, our model should provide the possibility for system administrators to edit property sets, if required. In such a scenario, the properties of the individual variants would have to be kept consistent at all times either through managing their redundancy or through a more complex graph structure.

A further question posed by both the graph structure presented in Figures 4.3(a) and (b) is when an object can gain variants. One possible solution is to define all variants when the object is created, while another would be to have the possibility that an object can gain variants over time. Additionally, if the second approach is chosen, it has to be decided whether capturing the times when such an operation happens is important. Defining the evolution of an object to include the creation of new variants is useful to cope with the advent of new requirements at a later stage of the lifetime of an application. However, similar to the latest revision that provides access to the most up-to-date version of an object, the current set of variants should be favoured over the management of historical states of an object. Figure 4.3(c) therefore shows a version graph where variants are given priority over revisions. Due to the on-line nature of variants, we believe that it is important to optimise the version graph towards queries involving mainly alternative versions. As can be seen from the figure, the currently existing variants are linked to the object and can thus be accessed directly. Each variant is linked to its revisions that have been made over time. As indicated by the bold arrow, again the latest revision is marked specially to enable faster look-ups. Although off-line queries discussed above are supported by this version graph, their evaluation is burdened with a performance penalty resulting from the fact that the history of objects has to be computed. Nevertheless, we have chosen to base our version model on this third approach of organising the version graph. In Figure 4.4, we show how the metamodel of an OM object has been extended to accommodate revisions and variants in the discussed manner.

In contrast to the metamodel of the traditional OM object presented in Figure 4.2, the metamodel of the versioned object does not link objects to their instances directly. Instead, the model has been extended to include two additional collections, Revisions and Variants. True to the intention of favouring variants over revisions motivated above, the path connecting the Objects collection with the Instances collection goes to variants first before following revisions. Hence, an object is linked to one or more variants through the *HasVariants* association. As indicated by the dashed arrow, association *DefaultVariant* is a subcollection of *HasVariants*, capturing the fact that every object has exactly one default variant. The same design has been used to connect a variant to a non-empty set of revisions. Again the association *LatestRevision* is modelled as a subcollection of the |*HasRevisions*| association, denoting one revision as the most recent version of the object. To establish an order among the revisions of a variant, association |*HasRevisions*| has been declared to be a ranking represented graphically by vertical bars surrounding the name of the association. While this ranking orders the revisions within the scope of a variant, it does not guarantee a complete order over all revisions of an

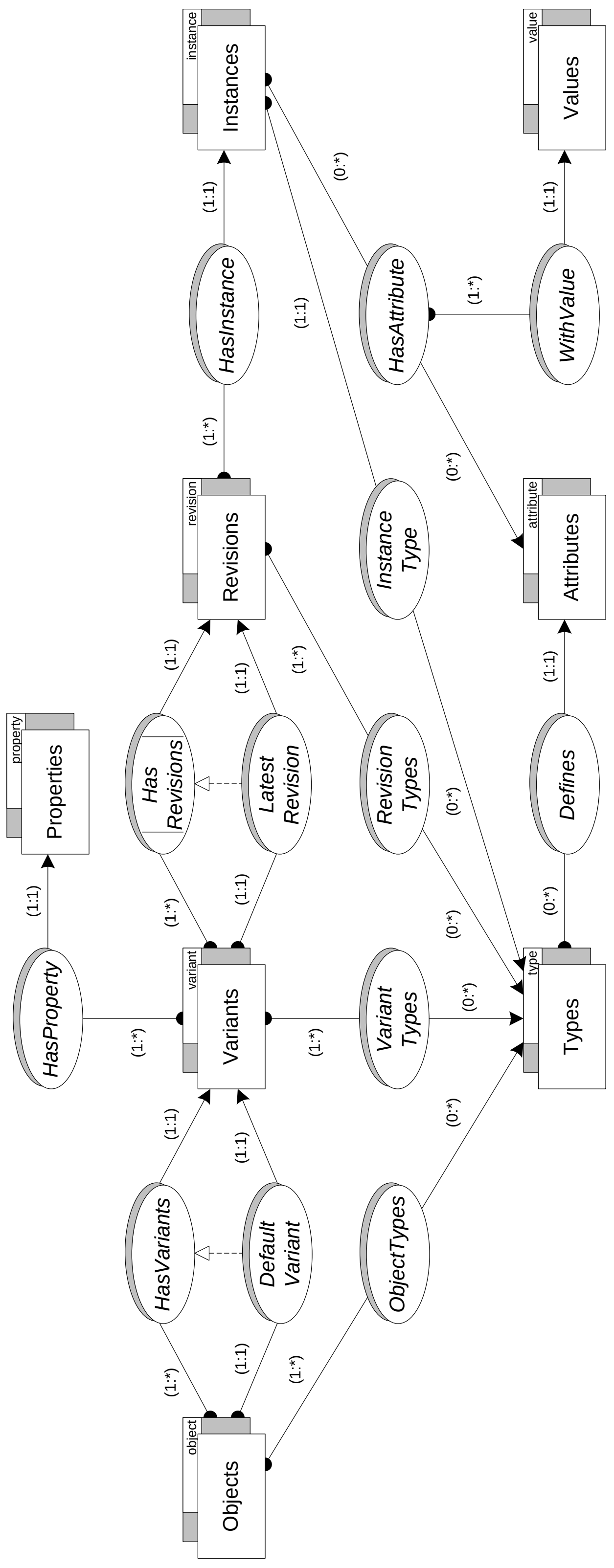

Figure 4.4: Metamodel of an extended object in OM

object. Therefore, type revision still defines a timestamp attribute that is used to capture the sequence of revisional versions of the entire object.

Each variant contained in collection Variants is associated with one or more properties through the *HasProperty* association. The condition that variant properties do not partake in the revisional evolution of an object is ensured by linking them directly to the variant concept itself. As the revisions of an object are referenced through one of its variants, this model ensures that they all share the same properties. The additional condition that it is illegal for an object to have two or more variants with the same set of properties cannot be expressed using the graphical notation. We use the algebraic expression below to specify this characteristic of our version model.

$$\begin{aligned}&\forall\, o \in \text{Objects},\ \forall\, v_1, v_2 \in rng(\textit{HasVariants}\ dr(\{o\})) :\\ &\quad v_1 \neq v_2 \Rightarrow rng(\textit{HasProperty}\ dr(\{v_1\})) \neq rng(\textit{HasProperty}\ dr(\{v_2\}))\end{aligned}$$

The expression defines v_1 and v_2 to be variants of an object o by first restricting the domain of the *HasVariants* association to only occurrences of o. The domain restriction operation dr constructs an intermediate association that only includes links that have one of the objects contained in the given set as their source. Then, the range operation rng traverses the links expressed by an association and returns the target objects only. In the above condition, the result of this algebraic expression will be the set of all variants of the object o. The same construct is used in the implication stating that if two variants of the same object are different then the set of properties defined by them is also different. Here *HasProperty* is restricted to v_1 and v_2, respectively. The set of property values is then obtained by applying the rng operation to the resulting intermediate association.

The *ObjectTypes* and *InstanceType* associations that capture which types are defined for an object and its instances have been taken on from the original metamodel. However, in the metamodel describing the versioned object, they are complemented with two additional associations, *VariantTypes* and *RevisionTypes*. Although an object can have multiple instances in OM, versions of an object are always defined for the entire object including all its instances. As a consequence, this means that our version model will not be able to track if an object evolves in terms of gaining or losing instances. Although this is a serious disadvantage, if the model were to be used to support the versioning of the database schema, we have decided not to offer this functionality for the time being. Therefore, the new associations that capture which types are defined for a revision and for a variant have to specify the same set of types as the object through association *ObjectTypes*. Again, this condition cannot be expressed graphically and is therefore given in algebraic form below.

$$\begin{aligned}&\forall\, o \in \text{Objects},\, \forall\, v \in rng(\textit{HasVariants}\ dr(\{o\})) :\\ &\quad rng(\textit{ObjectTypes}\ dr(\{o\})) = rng(\textit{VariantTypes}\ dr(\{v\}))\end{aligned}$$

$$\begin{aligned}&\forall\, o \in \text{Objects},\, \forall\, r \in rng(\textit{HasVariants}\ dr(\{o\}) \circ \textit{HasRevisions}) :\\ &\qquad rng(\textit{ObjectTypes}\ dr(\{o\})) = rng(\textit{RevisionTypes}\ dr(\{r\}))\\ &\quad \wedge\ rng(\textit{ObjectTypes}\ dr(\{o\})) = rng(\textit{HasInstance}\ dr(\{r\}) \circ \textit{InstanceType})\end{aligned}$$

The first expression ensures that all variants of an object o define the same set of types as the object itself. Similar to the expression discussed before, this is achieved by restricting the domain of association *VariantTypes* once to each variant v of o and then comparing the range of the intermediate association to the set of types associated to o through *ObjectTypes*. The second expression contains two conditions. On the one hand, it uses the same approach as the first expression to check that all revisions of an object specify the same set of types as the object o itself. On the other hand, the second clause of the conjunction additionally controls that all instances associated with a revision r of object o also define this set of types. This part of the expression replaces the condition presented in the traditional model, ensuring that the types of the instances of an object match the types defined for the object.

Compared to the original metamodel of an object in OM, collections Revisions and Variants and a few associations linking them to other collections are the only additional concepts introduced in the extended metamodel. The rest of the collections and associations remains unchanged, which means that just the path connecting an object to its instances is now more complex than before. Both members of collections Types and Instances continue to be associated with attribute metadata represented by objects belonging to collection Attributes. Also, data is still linked to an attribute of an instance using the concept of the ternary association *WithValue* that has the *HasAttribute* as its source.

4.2.3 Identifying and Referencing Objects

Having discussed the internal structure of a single object in terms of its version graph, it is important to specify how the different parts of an object can be identified and referenced from the outside. As with any other object-oriented systems, platforms built to support the OM data model use the notion of object identifiers to reference a database object. An object defined by the original model presented in the previous section is completely identified using a simple identifier of the form `o927`. However, as an object can have multiple instances of types that have been defined along different paths of an inheritance hierarchy, access to data can still be ambiguous. For instance, if two parallel types define an attribute with the same name, there is no way of knowing which instance of the referenced object should be accessed. Therefore, to unambiguously access the data stored in an object, both an object identifier and a so-called browsing type that defines in which role the object is accessed must be given.

Although a goal of our version model is to retain this interpretation of object identifier to ensure its compatibility with the OM data model as originally specified, an extension of the format of object identifiers is inevitable. In the extended metamodel of an OM object, instances are associated with revisions and it is therefore necessary that each revision can be addressed directly. Assuming that at every point in time t at most one revision of an object is created in the system, it suffices to extend the object identifier with the corresponding timestamp. However, to keep track of variants and to enforce the underlying orthogonality of the two concepts, it is also worthwhile to include an indication in the object identifier to which variant a revision belongs. In Definition 6 we specify the syntax and the semantics of the extended object identifier used to reference objects in our system.

Definition 6 (Extended Object Identifier)**.** The format of the extended object identifier that is used to reference versioned objects as a whole or in parts is given in Backus-Naur Form (BNF) below.

$$\texttt{"o"}\ \mathit{oid}\ [\ \texttt{"@"}\ \mathit{rev}\ ]\ [\ \texttt{"["}\ \mathit{var}\ \texttt{"]"}\ ]$$

The object itself is referenced by the *oid* part of the extended identifier, whereas its revisions and variants are identified by the two specifiers *rev* and *var*, respectively. An example of an extended object identifier would be `o927@9[3]` which references variant 3 in revision 9 of object 927.

As can be seen from the definition of the extended object identifier, the specification of both a revision and variant specifier are optional. This flexibility has been built into the object identifier to support both specific and generic references, as introduced in Chapter 3. In our model, a specific reference to an object is made using the full format of the object identifier, allowing a specific version of the object to be pinpointed. Generic references, however, are supported through partially specified identifiers that reference all versions, or a subset of the versions, of an object. At run-time, such partially specified identifiers are completed by the system to form a fully specified reference. To illustrate how generic references work in our model, it is necessary to first understand how identifiers are assigned to the versions of an object. An example of a versioned object is given in Figure 4.5.

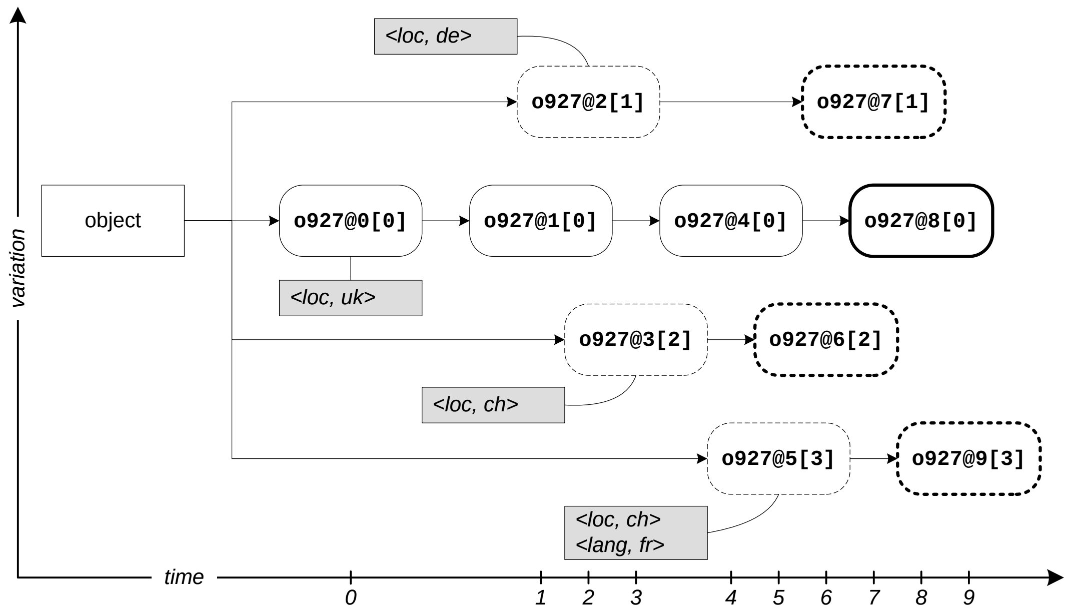

Figure 4.5: Example of a versioned object

The figure shows the evolution of object 927 along the time and variation axes. The object itself is symbolised by the rectangle shown at the left hand side of the figure. Its evolution in terms of revision and variants is represented by the tree rooted at the object. Each version of the object is shown as a rounded rectangle containing the fully specified object identifier of the corresponding revision or variant. At time $ts = 0$, the first version of the object is created. It is the default variant of the object and, as indicated by the context value $\langle loc, uk \rangle$, its content has been designed for the United Kingdom. Note that, in our version model, the first variant that is created for an object and all its successors are assumed to be the default variants. This is indicated in the figure by representing these versions with a rounded rectangle delineated by a solid line. The same fact, however, is also represented in the object identifier as all default variants have $var = 0$ as their variant specifier.

At time $ts = 1$, the default variant is updated and a new revision created. Note that the revision specifier of the new version is incremented to $rev = 1$, while the variant specifier is left unchanged. Also, the properties associated with the original revision of a variant are not duplicated to its successor as our model does not allow properties to be changed over time. The creation of a variant of the object is shown in the figure to occur at time $ts = 2$, when a variant for Germany is added to the object. As mentioned before, to create a variant, a set of context values has to be provided that is different from all existing variant properties. As the set consisting of the context value $\langle loc, de \rangle$ is different from the set specified by the default variant, the new variant is valid and can be created. The variant specifier in the object identifier of the new version is set to $var = 1$ to reflect the fact that this version is an alternative to the existing versions of the object. This situation is captured graphically in the figure by representing variants other than the default variant with a rounded rectangle delineated by a dashed line.

Later, two additional variants are created at times $ts = 3$ and $ts = 5$ to reflect the requirements of Switzerland and the French speaking part of Switzerland. Again, the rule that a new variant has to define properties that do not match any of the previously defined sets has to be observed. Note that the set $\{\langle loc, ch \rangle\}$ of variant 2 and the set $\{\langle loc, ch \rangle, \langle lang, fr \rangle\}$ of variant 3 are not in conflict as they are not considered to be the same, due to the additional value in the properties of variant 3. The most recent revisions of each variant are those furthest to the right and are depicted as rounded rectangles delimited by a bold line. In contrast to default variants, where it is possible to encode their special status in the object identifier using variant specifier $var = 0$, distinguishing the latest revision in such a way is not possible, as versions gain and lose that status as the object evolves. Hence, this metadata about the object is managed by a look-up table stored in the object itself that maps each variant specifier to the revision specifier of the latest version.

In order to access the data represented by an object, it is necessary to reference the required version of the object using a fully specified object identifier. As previously stated, generic references are supported based on partially specified identifiers and hence there has to be a way to complete such partial identifiers. Having defined the notions of latest revisions and default variants, our model can rely on these concepts to determine which version is referenced by an incomplete object identifier. If the object identifier does not contain a revision specifier, the corresponding specifier of latest revision is inserted. Otherwise, our model uses the variant specifier of the default variant if the object identifier omits this information. Finally, if an object identifier consists of the object specifier only, both the latest revision and the default variant are accessed. In this way, our version model is also capable of transparently interfacing with client applications that do not require the management of revisional or alternative data. Table 4.1 shows three examples of such completions of generic references.

Partial Identifier	**Completed Identifier**
`o927[3]`	`o927@9[3]`
`o927@3`	`o927@1[0]`
`o927`	`o927@8[0]`

Table 4.1: Completing generic references

The first line shows how an object identifier that omits the revision specifier is completed. The given references specifies that variant 3 of object 927 should be accessed. Coming back to the example given in Figure 4.5, this partial object identifier can be completed to include the revision specifier $rev = 9$, as this is the latest revision of the specified variant. The case presented in the second line of the table is more complex. As the object identifier does not reference a specific variant, the default variant has to be accessed. However, in order to do so, it is not sufficient to simply insert the variant specifier $var = 0$ pointing to the default variant. As can be seen from the figure, there is no version of the object that is identified by the resulting completed object identifier. Hence, it is necessary to also adapt the revision specifier to obtain a valid object identifier. The revision specifier in the scope of the default variant that corresponds to $ts = 3$ in the scope of the whole object can be found by traversing the revision chain of the default variant to find $max(T_{var=0}) \leq 3$. Doing so, the correct revision specifier $rev = 1$ is found, and the complete object identifier as shown in the table can be constructed. Finally, if both the revision and the variant specifier are omitted, the latest revision of the default variant are accessed. This completion process is again very straightforward, as the revision specifier $rev = 8$ and the variant specifier $var = 0$ can simply be inserted into the object identifier.

An important aspect of object-oriented systems is the fact that objects do not exist in isolation but are interconnected with other objects to form complex graph structures. To form such structures, the OM data model provides two concepts that allow an object to be associated with other objects. Simple references, as known from most object-oriented programming languages, can be expressed by defining an attribute that points to another object. Such simple references are managed within an object. They are uni-directional and do not provide referential integrity. When the referenced object is deleted, the problem of a dangling reference can arise if the value of the corresponding attribute of the referencing object is not updated accordingly. To address these issues, the OM data model provides the previously mentioned concept of associations that expresses relationships between objects both at the conceptual and the implementation level. Analogous to collections that contain object identifiers, associations are also represented by objects, but contain tuples of object identifiers. These object identifier tuples capture which object is connected to which other object. Hence, associations are managed outside the objects as first-order concepts. They can be navigated in both ways as they are completely symmetrical. Due to the fact that they express references among objects explicitly, associations can be directly addressed in the query language, based on their name. Finally, associations provide a natural point in the system to manage and enforce cardinality constraints as well as to prevent referential inconsistencies.

The question arises whether relationships between objects should be versioned together with the objects. We believe, that the information content of an object-oriented database consists of both the objects and the relationships. Therefore, the capability to version relationships is desirable and should be supported by our version model for context-aware data management. In the case of simple references, versioning is supported by the fact that these relationships are expressed as attribute values of an object. Using specific references, different versions of the referencing object can point to different versions of the referenced object. As a consequence of the versioning of the object containing the reference, the relationship itself appears to be versioned. Clearly, this solution suffers from the deficiency that the relationship cannot be versioned independently from the object. Again, associations provide an elegant solution to this shortcoming. As associations are themselves objects, they can have revisions and variants as with any other object in the system. Therefore associations can be used to capture any state of the relationships in an object-oriented system that is specified based on the OM data model. While the fact that both objects and relationships can be versioned using one set of concepts validates our approach, it also demonstrates the power of the OM data model at the same time.

4.3 Context-Aware Query Processing

Based on the presented version model that defines how the revisions and variants of an object are organised, a query processing mechanism has been defined. In contrast to traditional database management systems where the result of a query is entirely defined by the query itself, in our system the outcome of evaluating a query depends on additional factors. While the revision of an object that will be accessed during query processing depends on the current time, the retrieved variant can change according to the context in which the query is evaluated. However, similar to the definition of the extended object identifiers, care has to be taken to ensure that the augmented query processor is compatible with systems that do not require additional functionality. Thus, the query processor has to be able to also produce meaningful results in the absence of time and context specifications. Apart from query evaluation, compatibility with traditional applications also has an impact on the query language used by the system. While some changes have to be made, it is not desirable to define a whole new query language to profit from the additional features of the model. Rather, all modifications should be made in the form of optional extensions to the language that can be omitted if not required. In the remainder of this section, our approach to define such a query processor will be presented. While queries along both the time and the variation axis are possible, the discussion will focus on the context-aware aspects of the system in terms of selecting appropriate variants. The processing of queries involving time has been well researched in the field of temporal database systems and is thus not presented here.

4.3.1 Influencing the Context State

As mentioned before, all context factors that can affect the evaluation of a query are captured by the context values within the context state $C_\star(S)$. Influences other than these values will have no effect on context-aware query processing as proposed by our approach. Therefore it is important that client applications can configure the context state to reflect all relevant context information. To provide support for influencing the context state, two major issues have to be addressed. The first issue is to design interfaces that enable applications to communicate their present state to the system. The second issue is to determine the sphere of influence of these communicated context values on a conceptual level. While the range of interfaces supported by our system will be presented later in

Level	**Description**
global	The global level is the most general level and allows context values to be specified that apply to all queries evaluated by the system.
session	Context values specified at the session level influence all queries that are evaluated in the corresponding client session.
command	The finest level of granularity is the command level that allows context values to be specified that influence a certain set of commands only.

Table 4.2: Granularity levels of $C_\star(S)$

Section 4.4 describing the implementation of the version model, at the moment, we focus on the scope of the context state of an application. Depending on the application requirements, it makes sense to consider influencing $C_\star(S)$ at different levels of granularity. For example, certain context values, such as the preferred language or the location of a user, are valid within a single client session only, while other factors may be shared among several sessions or even the entire system. Finally, it is also imaginable that certain applications require the context to change within a single query, to perform inter-context computations combining data valid in separate contexts. Based on the application requirements presented in Chapter 2, we have decided to offer three levels at which $C_\star(S)$ can be influenced. An overview of these granularity levels is given in Table 4.2.

Introducing a hierarchy of granularity levels raises the question of precedence among the different levels. As it is possible to define context values with the same name and different values at several levels, the value that will be used by the system has to be defined. The most natural solution is to consider the values defined at the lowest level first. Therefore, a value defined at the command level takes precedence over values defined at the session level, while session level values are used before global values. This simple approach using inheritance and overriding is, however, not sufficient to cover all application requirements, and further refinements are necessary. When a client application influences a subset of the context state at the session level, context values defined at the global level only could interfere with the requirements of the application. Assume for example, that the global context state contains the value

Mode	Description
inherit	The inherit mode is the default mode and uses values from a higher level if they are not specified by the level that is currently used to define the context.
replace	Context values that are not specified by the defining level are assumed to be undefined, as the replace mode blocks inheritance.
combine	In the combine mode, values from the level used to define the context and all the above levels are integrated.

Table 4.3: Precedence modes

$\langle lang, fr \rangle$ and the application sets the value $\langle loc, ch \rangle$ at the session level. As the language context is not set by the application, it will be inherited from the global level and thus all subsequent queries are evaluated in the context of the French speaking part of Switzerland. Depending on the application scenario, this result could be unintended and hence it should be possible to block the inheritance of context values in some cases. In addition to the three granularity levels, a context-aware query processor therefore also needs to give control over the precedence among those levels to the application. Table 4.3 details three modes that can be used in conjunction with the granularity levels to obtain better configurability. As an example of how the combined mode is used, assume that the session level defines the context value $\langle lang, de \rangle$ while the context value $\langle lang, en \rangle$ is defined at the global level. The resulting context state $C_\star(S)$ will then reflect that both English and German language variants are valid.

4.3.2 Context Matching

Context matching determines which variant of an object is accessed during query evaluation. It uses a scoring function f_s that compares the context state $C_\star(S)$ to the variant context $C_v(S)$ of each variant v of an object o. In this comparison, f_s assigns a score value to every object variant, which is then used to select the highest scoring variant. For a variant to be selected, however, its score has to be higher than a certain threshold that can be configured in the system. If no variant scores higher than this threshold or if two or more variants are assigned the highest score, the default variant is used instead. The notion of thresholds controlling variant selection has been introduced to prevent unintended results in situations where, due to unexpected context values, all variants are assigned a low score or multiple variants obtain the highest score. The solution of returning the default variant in these cases has been chosen to ensure deterministic behaviour under any circumstances that would not be guaranteed if a variant were to be selected at random.

As f_s ranks the different variants in terms of their suitability in the current context, context matching uses a best match rather than an exact match algorithm. This approach is motivated by the special characteristics of context-aware query processing. True to the nature of context-aware computing as described in Chapter 2, the context state of the system does not represent an integral part of the query, but rather optional information that is used by the system to augment or enhance the query evaluation process. Apart from the fact that the system also has to produce meaningful results when no context information is available, another consequence of this understanding of the notion of context is that it should not be regarded as a specification that has to be followed exactly. However, in order to support this notion of context, it has to be possible for the query processor to use the default variant of an object instead of another variant at any time during query evaluation. This assumption that the default variant is always a valid choice to represent an object is a strong premise as it will have a noticeable impact on how variants of an application object have to be designed in order to obtain the desired context-aware behaviour of the whole application.

Based on these considerations, the matching algorithm that is expressed in Figure 4.6 in terms of the operations of the algebraic query language has been defined. Given the object o that needs to be accessed by the query processor and the context state $C_\star(S)$, the algorithm returns the best matching variant. If a best matching variant according to the conditions described earlier is not found, the default variant is returned. As specified by the algebraic expression on line 1, the algorithm begins by retrieving the set of all variants V_0 of the given object o. In the next step on line 2, the algorithm uses the map operation ($\propto$) to generate a set V_1 that contains tuples relating each variant to its set of property values. The map operation takes as arguments a set and a function. It then returns the set that results from applying the given function to each member element in the given set. Here, the function $x \rightarrow (x \times rng(\textit{HasProperty}\ dr(\{x\})))$ is mapped to the set V_0, causing every element $x \in V_0$ to be represented by a corresponding tuple in

V_1 specified by the function using the tuple constructor ($\times$). To illustrate the functioning of the map operation, assume a possible member of V_1 could be the tuple $\langle v_1, \{\langle lang, en\rangle, \langle loc, uk\rangle\}\rangle$. The map operation is used once again in the following processing step, when the function $x \longrightarrow (dom(x) \times f_s(rng(x), C_\star(S)))$ is mapped to the set V_1 on line 3. In the resulting set V_2, each element of V_1 is represented by a tuple relating the variant to its score value computed by f_s according to the context state $C_\star(S)$. Without making any assumption about the definition of f_s, the member of V_2 corresponding to the example tuple given above might take the form $\langle v_1, 0.75\rangle$. Note the use of the *dom* and *rng* operation to access the left-hand or right-hand side of a tuple, respectively.

```
MATCH($o, C_\star(S)$)
1  $V_0 \leftarrow rng(\textit{HasVariants}\ dr(\{o\}))$
2  $V_1 \leftarrow V_0 \propto (x \longrightarrow (x \times rng(\textit{HasProperty}\ dr(\{x\}))))$
3  $V_2 \leftarrow V_1 \propto (x \longrightarrow (dom(x) \times f_s(C_\star(S), rng(x))))$
4  $s_{max} \leftarrow max(rng(V_2))$
5  $V_3 \leftarrow V_2 \,\%\, (x \longrightarrow rng(x) = s_{max})$
6  if $|V_3| = 1 \wedge s_{max} \geq s_{min}$
7     then $v \leftarrow V_3\ nth\ 1$
8     else $v \leftarrow rng(\textit{DefaultVariant}\ dr(\{o\}))\ nth\ 1$
9  return $v$
```

Figure 4.6: Matching algorithm

Based on V_2, the maximum score value s_{max} over all variants of the object o is computed on line 4. On line 5, this maximum score is then used to select the variants that reach this value. In order to do so, the matching algorithm uses the selection operation (%) that applies a Boolean function to each element of a set. If the Boolean function returns *true*, the corresponding element will be included in the result set and omitted otherwise. To only include variants with the maximum score in V_3, the algorithm uses the predicate function $x \longrightarrow rng(x) = s_{max}$. Finally, lines 6–8 check if only a single variant has obtained the maximum score and whether this score is higher than the system threshold s_{min}. If both of these conditions are fulfilled, the algoritm uses the *nth* operation to extract the first element of V_3 and assigns it to v. Otherwise, the default variant is retrieved and assigned to v. In the last step, the algorithm returns v which now contains the best possible representation of o under context state $C_\star(S)$.

The functioning of the matching algorithm itself is generic, as it only defines the process of how a variant of an object is selected. The decision as to which variant will be selected is encapsulated in the scoring function f_s. It is therefore the scoring function rather than the algorithm that influences the result of the matching process. This separation of concerns between the algorithm and the scoring function is the basis of matching behaviour that is specific to a particular application scenario. Although the selection process is always the same and built into the system, the matching can be configured by using different scoring functions for different requirements. The major advantage of this approach is that it supports a certain degree of flexibility without abandoning control over context-aware behaviour completely. This trade-off accommodates both the needs of applications and the requirement of data management systems having to operate within well-defined boundaries. While there can be no universal scoring function that will produce satisfactory results for all applications, we have developed a scoring function general enough to cope with most analysed scenarios. We now present how this function is defined and illustrate its operation.

At the core of every scoring function f_s lies the method which is used to match the context values in $C_\star(S)$ to the properties of a variant context $C_v(S)$. The definition of any such function has to specify the conditions under which a context value matches a property value. Although we have presented an equality for two context values, using this equality as the basis for a matching condition would lead to a rather limited system. Assume for example that an object variant represents content for both Switzerland and Liechtenstein. To characterise this variant, we associate the tuples $\langle loc, ch\rangle$ and $\langle loc, li\rangle$ with the variant. While this is a possible solution to solve this particular problem, it has some disadvantages. One problem is that this solution does not scale when more than just a few context values are represented by the same variant. Imagine, for instance, that a variant is valid during a certain time period only. Clearly, it would not be feasible to link a property value capturing each day of this validity period to the variant. Further, this solution is in direct violation of the definition of a context as Definition 3 states that no context can contain more than one value with the same *name* field. Of course, the definition could be changed accordingly to solve this particular problem. However, we believe that this solution would lead to a cumbersome representation of context that is hard to process by the data management system. As we will see in the following, there are also additional challenges that cannot be addressed by loosening the definition of a context.

For the reasons given above, we have decided to introduce a less restrictive condition to determine if two context values match. The first step towards such a matching condition is the definition of additional given sets that partition the VALUES given set as shown in Table 4.4. Based on these new given sets, the *value* field of a context value can be used to specify more than one value. The original behaviour resulting from using identity to define equality can still be obtained by using atomic values from given set ATOM. In addition to that, collections of atomic values can be expressed using a value from the SET given set. For example, the situation discussed above can now be resolved elegantly by assigning the property value $\langle loc, ch{:}li\rangle$ to the variant in question. To address problems such as validity periods, values defined by given set RANGE allow intervals to be defined based on a lower and upper bound. Finally, if a variant is an appropriate representation for all possible values of a context dimension, the wildcard value from given set STAR can be used. Providing a wildcard in this setting might seem paradoxical at first, as one might argue that the corresponding context value could simply be omitted. Its right to exist will be motivated later, when we present how the scoring function f_s computes the score value, based on value matching.

Set	**Syntax**	**Description**	**Examples**
ATOM	x	Atomic value	`en`, `27`
SET	$x_1\{: x_i\}$	Set of atomic values $S = \{x_1, \ldots, x_n\}$	`at:ch:de`, `red:blue`
RANGE	$x_{min}..x_{max}$	Range of atomic values $I = [x_{min}, x_{max}]$	`5.5..7.0`, `a..f`
STAR	$*$	Wildcard	`*`

Table 4.4: Extended value syntax

Defining in which cases two context values specified using the extended value syntax match is the next step in establishing a less restrictive matching condition. As context values describe both the context space of the system and the variants of an object, it has to be possible to compare any combination of these four given sets. For each of the resulting sixteen cases, Table 4.5 defines the matching of extended values represented as $\cong$ by listing the conditions that govern when two values match.

x	y	**Matching Condition**
ATOM	ATOM	$x = y$
ATOM	SET	$x \in y$
ATOM	RANGE	$y_{min} \leq x \leq y_{max}$
ATOM	STAR	$\top$
SET	ATOM	$y \in x$
SET	SET	$x \cap y \neq \emptyset$
SET	RANGE	$\exists\, k \in x : y_{min} \leq k \leq y_{max}$
SET	STAR	$\top$
RANGE	ATOM	$x_{min} \leq y \leq x_{max}$
RANGE	SET	$\exists\, k \in y : x_{min} \leq k \leq x_{max}$
RANGE	RANGE	$max(x_{min}, y_{min}) < min(x_{max}, y_{max})$
RANGE	STAR	$\top$
STAR	ATOM	$\top$
STAR	SET	$\top$
STAR	RANGE	$\top$
STAR	STAR	$\top$

Table 4.5: Matching of extended values

The condition specified on the first line represents equality based on identity, as given in Definition 2. In contrast, the subsequent lines provide support for a more flexible description of the matching. The first group of equality conditions, for example, specifies when a context value containing an ATOM value matches one with a value represented by given set SET, RANGE or STAR. The definitions are straightforward as an atom matches a set or a range if it is contained within it. The symbol $\top$ is used in those lines of the table where at least one of the two context values contains a wildcard from STAR to indicate that these values always match any other value. Context values containing a SET value are compared to those of the four other types in the second group of conditions. While the first line simply reverses the condition given above, the second line states that two values contained in SET match if their intersection is not the empty set $\emptyset$. In order for a value of the SET given set to match a value of given set RANGE, there has to be at least one element that is within the interval boundaries. The third group of conditions relates context values containing a RANGE value to all other kinds of values. Again, the first two conditions are defined based on a previously discussed condition, with the roles of x and y interchanged. Two RANGE values match if the corresponding intervals overlap.

Finally, based on the extended value syntax and the extended definition of equality, a scoring function f_s can be specified. As can be seen from the matching algorithm presented in Figure 4.6, the scoring function takes as arguments two contexts and returns a real number indicating the score of the match. Even though this is not strictly required by the matching algorithm to function correctly, we assume that the score values returned by f_s are normalised and that $0 \leq f_s \leq 1.0$ holds. Generally, the value returned by a scoring function should quantify the similarity of the given contexts, where 0 indicates no correspondence and 1 represents a complete match. The scoring function given in Definition 7 exhibits these characteristics, as the returned score increases with the number of context dimensions that match.

Definition 7 (Simple Scoring Function $f_{s'}$)**.** The *simple scoring function* $f_{s'}$ takes two contexts C_1 and C_2 as arguments and returns a scoring value representing the number of matching context dimensions of the two contexts normalised by $|N_1|$. It is defined as

$$f_{s'}(C_1, C_2) = \frac{1}{|N_1|} \sum_{n \in N_1} f_i(n, C_1, C_2)$$

where N_1 denotes the set of all names of context values specified by C_1 and the *indicator function* f_i is given by

$$f_i(n, C_1, C_2) = \begin{cases} 1 & \exists\, c_1 \in C_1, c_2 \in C_2 : \\ & \quad name_1 = name_2 = n \wedge value_1 \cong value_2 \\ 0 & \text{otherwise.} \end{cases}$$

Figure 4.7 shows a versioned object with three alternative representations for different contexts. Starting from this example, we will illustrate the functioning of a matching algorithm that uses the simple scoring function introduced above. Assume the context state of the system is defined as $C_\star(S) = \{\langle \textit{format}, \textit{html} \rangle, \langle \textit{lang}, \textit{en} \rangle\}$. Based on $C_\star(S)$, the scoring function $f_{s'}$ will assign a score of 1.0, 0.5 and 0.0 to the variants `o927@0[0]`, `o927@1[1]` and `o927@2[2]`, respectively. The first variant defines a variant context $C_v(S)$ that matches all context dimensions contained in context state $C_\star(S)$ and thus receives the maximum score. As the variant context of the second variant contains one matching value only, it gets a lesser score. Finally the score of 0.0 of the last variant is explained by the fact that none of the context values in its variant context match a value in $C_\star(S)$. Therefore, the matching algorithm would return the first variant which, incidentally, is also the default variant of the object.

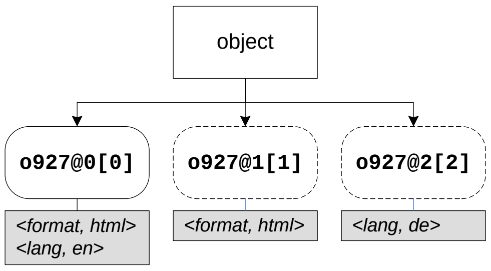

Figure 4.7: Example object

Unfortunately, this simple matching function does not always produce satisfactory results when used in conjunction with the presented matching algorithm. To improve the practical value of the proposed query processing mechanism, there are two major problems that still need to be solved. For example, ambiguous situations where multiple variants of an object obtain the highest score have to be avoided whenever possible. As the algorithm simply returns the default variant in this case, the potential of our approach in general is limited needlessly. And while it is neither possible nor reasonable to give complete control over the matching process to the application developer, the developer has to be in as much control as required to prevent undesired or unintended query outcomes.

Figure 4.8(a) shows another example of a versioned object that is very similar to the one presented in Figure 4.7. The only difference between the two objects is the fact that, in contrast to the previous object, variant `o927@1[1]` is additionally described by the context value $\langle loc, uk \rangle$. Assume that the object is involved in a query processed under the context state $C_\star(S) = \{\langle format, html \rangle, \langle lang, en \rangle, \langle loc, uk \rangle\}$. In that case, both variants `o927@0[0]` and `o927@1[1]` are assigned a score of $0.\overline{6}$ which is also the highest score, as the remaining variant `o927@2[2]` does not match at all and is consequently assigned score 0.0. In this case, the default variant, indicated by a solid line in the figure, is returned. Although the default variant is, by definition, always a valid representation of the object, the fact that this choice actually favours context dimension *lang* over dimension *loc* is somewhat unsatisfactory. Instead of weighting context dimensions implicitly, based on which variant they describe, the application developer should be given the means to specify these relationships among context dimensions explicitly.

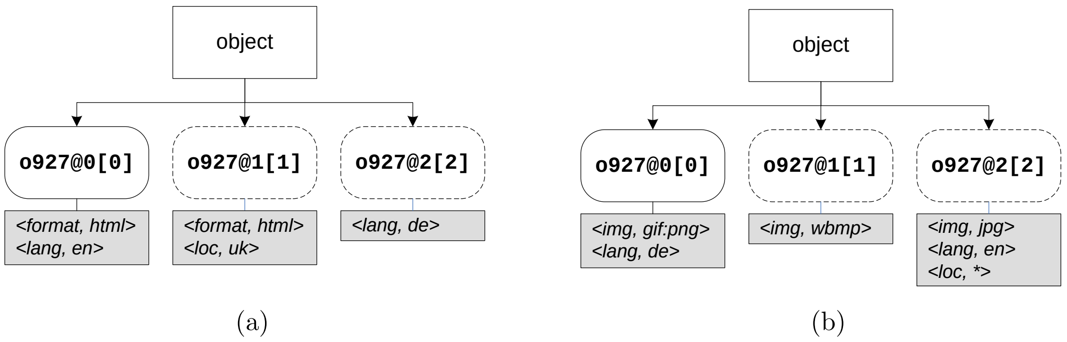

Figure 4.8: Example objects

An example of a situation where the matching algorithm returns an inadequate result is illustrated using the versioned object shown in Figure 4.8(b). The depicted object constitutes an image that has three alternative representations in terms of the image format. While the variant on the left-hand side is appropriate if the context state requests an image in the GIF or PNG format, the one on the right-hand side stores the image data in the JPEG format. Finally, the variant in the middle contains data in WBMP, a format required by older generations of mobile phones. Assume such a client querying for the object has set the context state to $C_\star(S) = \{\langle img, wbmp\rangle, \langle lang, en\rangle, \langle loc, uk\rangle\}$. By applying the scoring function to all three variants, the matching algorithm will obtain the values 0.0, $0.\overline{3}$ and $0.\overline{6}$ for the three variants from left to right, respectively. According to these scores, variant `o927@2[2]` will be used to represent the object in the context state $C_\star(S)$. This, however, will lead to problems, as the client that has set the context state will not be able to process the image data in this format. As motivated before, we see context information as an additional factor that is used to augment the functionality of our system. While we intend to stay true to this notion of context, we feel that it is necessary to give more control over the matching process to both the database developer and the client application. Therefore, in order to avoid situations such as the one just presented, we will describe how to raise the expressiveness of the syntax that is used to describe both the context state of the system and the variants of an object.

As mentioned before, ambiguous situations where more than one variant obtains the maximum score value can be reduced by explicitly weighting the context dimensions in relation to each other. Assume, for example, that in a given application scenario a match on the context dimension *format* should be considered more important than a match on the *lang* dimension. This assumption is valid in many systems, as retrieving content in the wrong language will at worst prevent the user from understanding it, whereas content in the wrong format will not be displayed at all. In order for our matching framework to support a mechanism that allows the query evaluation to be influenced in this way, we define a function that assigns a weight to every name of a context dimension relevant to an application. The specification of this context dimension weight function is presented in Definition 8.

Definition 8 (Context Dimension Weight Function)**.** The *context dimension weight function* denoted by w returns a weight $w(n) \in \mathbb{R}^+$ for every name $n \in N$, where N represents the set of the names of all context dimensions.

The weight function $w(n)$ is inserted into the simple scoring function as specified in Definition 7 as a factor which is multiplied with the indicator function f_i. It is therefore reasonable to assume that $w(n)$ returns a value of 1.0 as the default weight. If a context dimension is considered to be either half or twice as important as one with the default weight, the weights 0.5 and 2.0 are used. Although weights are useful to prevent cases where the system needs to fall back to the default variant, by reducing the probability of having multiple variants with the highest score, they have to be specified globally and apply to all objects defined in the system. This global nature renders the use of weights difficult in situations where undesired query outcomes need to be prevented. Tuning the weights to get the intended results in one case, may easily lead to new situations where the system performs differently from the designer's expectations. Depending on the complexity of the application, it can be very hard to predict the impact of changing a weight. Since testing if the system handles all context constellations correctly is not feasible in an efficient manner, it is difficult to track down problematic situations.

To address the problem of unpredictable matching results at a local level, we need to extend the syntax used for context values once again. The goal of this additional extension is to empower the creator of a variant to express required and illegal values of context dimensions. For instance, in the example situation illustrated by means of Figure 4.8(b), the client specifying the context state knows that it requires the image in the WBMP format but has no possibility to include this knowledge in the context state $C_\star(S)$. Therefore, we have introduced two prefixes $+$ and $-$ that can be used to indicate whether the prefixed context value is required or illegal respectively. Thus, in our example, the client could instead specify the context value $\langle$*img*, *+wbmp*$\rangle$ which would cause the scoring function to assign a value of 0.0 to all variants of an object that do not feature the value $\langle$*img*, *wbmp*$\rangle$ as part of their variant context $C_v(S)$. Consequently, the first two variants of the object now get a lower score which leads to the selection of the third variant, as intended by the client. In Table 4.6, we give the conditions under which two values x and y that are possibly prefixed are considered to match. Based on this table, we introduce the notation $x \cong_\pm y$ to specify that two values match according to their respective prefixes.

x	y	**Match Condition**
x	y	$x \cong y$
x	$+y$	$x \cong y$
x	$-y$	$\neg(x \cong y)$
$+x$	y	$x \cong y$
$-x$	y	$\neg(x \cong y)$
$+x$	$+y$	$x \cong y$
$+x$	$-y$	$\perp$
$-x$	$+y$	$\perp$
$-x$	$-y$	$x \cong y$

Table 4.6: Matching of prefixed values

Using the presented enhancements, it is now possible to define the general scoring function used as a default in our version model to support context-aware query processing. Definition 9 specifies this general scoring function f_s that incorporates the previously defined weighting function w as well as the notion of required and illegal context values. As mentioned previously, the weighting function is introduced as a factor multiplied with the indicator function f_s. Prefixed values are handled by a special function based on the matching condition $\cong_\pm$. Computing the product of all these comparisons and multiplying it with the score of the current variant will reset the value to 0.0 if a matching condition is violated and leave it unchanged otherwise.

Definition 9 (General Scoring Function f_s)**.** The *general scoring function f_s* takes two contexts C_1 and C_2 as arguments and returns a scoring value representing the number of matching context dimensions of the two contexts normalised by $|N|$. It is defined as

$$f_s(C_1, C_2) = \frac{1}{|N|} \sum_{n \in N} (w(n) \times f_i(n, C_1, C_2)) \times \prod_{n \in N} f_{\pm}(n, C_1, C_2)$$

where N denotes the union $N_1 \cup N_2$ of the sets of all names of context values specified by C_1 or C_2. The indicator function f_i is used as given in Definition 7 and the weight function w as introduced in Definition 8. Finally, the matching function for prefixed values $f_{\pm}$ is defined as

$$f_{\pm}(n, C_1, C_2) = \begin{cases} 1 & \exists\, c_1 \in C_1, c_2 \in C_2 : \\ & \quad \mathit{name}_1 = \mathit{name}_2 = n \wedge \mathit{value}_1 \cong_{\pm} \mathit{value}_2 \\ 0 & \text{otherwise.} \end{cases}$$

Note that we use the union $N = N_1 \cup N_2$ of the sets of all names of context values specified either by C_1 or C_2 to normalise the scoring value returned by f_s. This change caters for the fact that either C_1 could specify more values than C_2 or vice versa. In this situation, taking the notion of these so-called under-specified and over-specified variants into consideration can be of additional help in preventing ambiguity. Normalising over all names of context values is one possible way of achieving this in a neutral manner. Depending on application requirements, however, it can be necessary to favour either under-specified or over-specified variants. To realise this behaviour, we would need to integrate an additional summand into the matching function that tips the scales into the desired direction.

4.4 Implementation

As a proof-of-concept and basis for experimentation, the version model for context-aware data management has been implemented in an existing database management system. The choice of database that was extended with the presented functionality is a consequence of the selection of the OM data model as the framework in which the version model has been specified. Therefore, the natural foundation for an implementation of context-aware data management has been a platform that already provides native support for the basic concepts of the OM data model, such as multiple instantiation, classification and inheritance, as well as bi-directional associations. In the past, several such implementations of the OM data model have been realised, leading to a family of Object Model Systems (OMS) that comprises platforms implemented in Prolog, Python and Java. Each of these platforms addresses specific requirements of the development process of a database application. Implemented in Prolog, *OMS Pro* [296] has focussed on rapid prototyping by providing tools that both facilitate the design of a database model and help with testing the model with sample data. A subset of the concepts defined by the version model for context-aware data management were implemented in an extended OMS Pro platform called *OMSwe* [202, 203, 204] as a first effort to support context-aware web engineering. The *eOMS* [182] platform was implemented in Python as an extension of the relational database management system PostgreSQL and addresses multi-user access to data and transactional query processing. Applications communicate with the eOMS server using OML in requests and get back XML as a response. Supporting the needs of object-oriented application programmers are *OMS Java* [167] and, more recently, *OMS Avon* that were implemented in Java, based on different solutions for object persistence. Both approaches feature a comprehensive application programming interface that supports data access and modification based on the concepts of an object-oriented language.

However, a seamless development process can only be achieved if it is possible to move freely from one system to another. Therefore, all implementations of the OM model share a common language for data definition, manipulation and querying. While the OM data model standardises the way data is represented, OML standardises how this data is processed. What is missing is an application programming interface shared by all platforms that defines how programs can work with an OMS database. To close this gap, a uniform Java interface providing access to heterogeneous OMS platforms was developed. This interface, called OMS$^{\mathrm{jp}}$, offers a database concept to represent all aspects of an OMS implementation. Through this concept, the underlying platform can be configured and databases can be managed. OMS$^{\mathrm{jp}}$ also provides functionality for querying the database as well as accessing data and metadata objects. As everything in OMS—even metadata and system objects—is modelled as objects, it makes sense to offer certain of these concepts in the interface. OMS$^{\mathrm{jp}}$ therefore comprises objects that represent collections, associations, methods and types. The most important of these concepts is the representation of an object within the OMS database. As the object model of OMS is very different from the object model that is used in Java, it is, unfortunately, not possible to establish a direct mapping between Java objects and OMS objects. Hence, the interface simply offers one class that is used to represent all user defined object types. It is through this class that data can be read from and written to objects. Types are represented by special classes in the Java interface, as they are used frequently and have special properties. As OMS differentiates between different kinds of types there are also classes for these types in the interface. Also available as special classes are the fundamental concepts of object collections and associations. These objects offer various operations of the collection algebra that is defined by the OM data model. Hence, they can be used to implement complex queries on the model, instead of sending a textual OML statement to the database.

In OMS, triggers can be defined that are fired when certain events occur. As these events can be important to the client application, they are also propagated to the programming environment. This allows an application programmer to register code, by means of an event listener, that will be executed when events occur. Finally, for reasons of performance, the application programming interface also incorporates a cache. The cache is used to store query results and object data that have been fetched previously. Retrieving data from OMS can be very expensive and, since metadata in particular is very static, caching of results can lead to significant improvements in system performance. The cache is kept consistent using the previously discussed event listener mechanism. Whenever the interface receives a notification that a certain object has been changed or deleted, the corresponding cache entries are also invalidated. The cache component has been designed with extensibility in mind. A client can provide its own cache according to its needs and memory limitations. By default, the system uses an unlimited cache that grows corresponding to the number of objects in the underlying database, and has no replacement strategy.

The architecture and properties of OMS$^{\text{jp}}$ as discussed above make it an ideal framework for experimentation with our version model for context-aware data management. Therefore, in this section, we will start by presenting the aspects of the architecture of OMS$^{\text{jp}}$ that have formed the basis for its extension. We then go on to discuss those extensions and show how context-aware data management has been integrated into OMS$^{\text{jp}}$. A high-level overview of the architecture of OMS$^{\text{jp}}$ is shown in Fig. 4.9. As shown in the figure, the framework consists of three parts. Close to the *Database Application*, the *OMS^{jp} Interface* that defines the way in which an application interacts with the framework is shown. The interface is then implemented by a platform-dependent *Driver* which translates the functionality offered by the interface into the concepts offered by the database server. As with Java Database Connectivity (JDBC), the driver is completely decoupled from the interface and thus the back-end database server can be exchanged at any point in time. Whereas this decoupling of interface and driver is the main advantage of OMS^{jp}, a disadvantage inherent in this approach is that the interface has to be defined as the intersection of the functionalities offered by each platform. Therefore, the potential strength of certain database servers cannot be leveraged when they are accessed through OMS^{jp}.

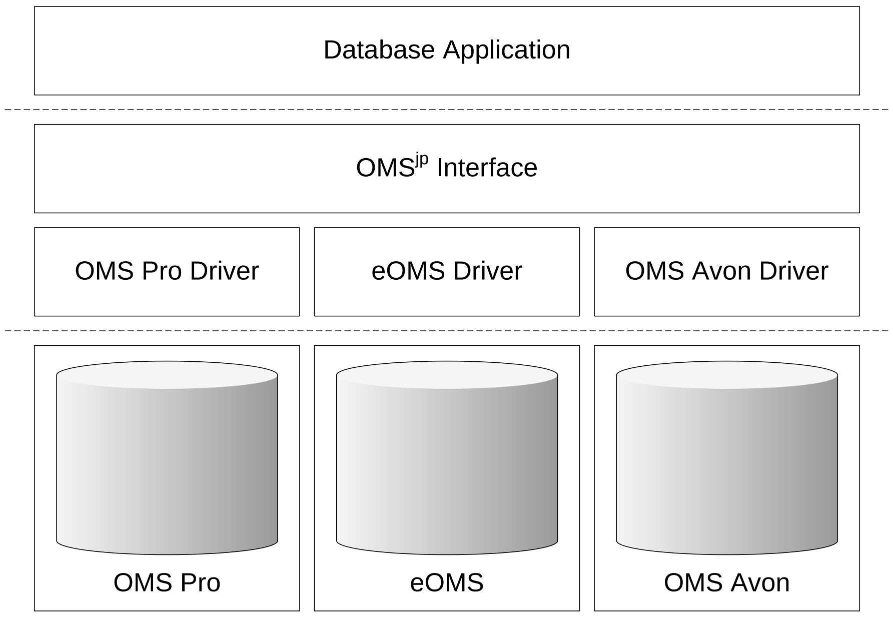

Figure 4.9: Overview

The functionality defined by the OMS^{jp} interface can be subdivided into three main areas. First, it provides support to manage the drivers required to communicate with the underlying OMS platform. When a client application initiates a connection to a database server through OMS^{jp}, it requests a suitable driver by specifying a URL that describes which platform, server and database are to be used. Based on a driver registry maintained by the interface, the corresponding driver is loaded, initialised and returned to the application. From this driver, the application can gain access to the specified database, which is the second area of functionality offered by the OMS^{jp} interface. As mentioned before, the database abstraction gives access to all functionality as well as to both data and metadata of the OMS system. Finally, the last part, the so-called value interface, defines how objects and values retrieved from the underlying OMS database are represented in Java and how Java values are converted to native values of the OMS platform.

While the first two parts of the OMS^{jp} interface are not relevant to the discussion of the implementation of our version model, the value interface is where the extension is most visible. An overview of the value interface as a UML diagram is given in Figure 4.10. At the top of the figure, interface OMSValue describes an interface that contains the methods common to all values that exist in the underlying OMS platform. At this level, the only method is toNative() that converts a Java value back to its native representation as defined by the database back-end. As can be seen from the specialisations of OMSValue shown in the figure, the interface defines several classes to represent different kinds of values. Interfaces OMSObject and OMSInstance probably define the most important kinds of values provided by the framework. They give access to the database objects and their instances. As mentioned above, it is not possible to express the concept of an object as defined by OM using a single Java class. Therefore, one OM object is represented as an instance of OMSObject and a set of instances of OMSInstance, each representing one of the object's types. The relationship between an object and its instances will be discussed in more detail later, as it is the focal point of the extension allowing objects to have versions.

Special predefined extensions of the OMSInstance interface represent concepts unique to the OM data model or system metadata. For example, interfaces OMSCollection and OMSBinaryCollection represent the concpets of unary and binary collections, as defined in the model, together with the corresponding algebraic operations. Associations between collections are captured by interface OMSAssociation and type metadata is accessible through OMSType and its extensions. Most OMS platforms offer three different kinds of value types, represented by interfaces OMSObjectType, OMSStructuredType and OMSBaseType. While instances of OMSObjectType describe what attributes and methods an object has, instances of OMSBaseType provide metadata about the types of base or atomic values. Finally, some OMS platforms support the notion of structured values that enable the database designer to define data records without having to create an object type. The most common use of these structured values that are described by instances of OMSStructuredType is to represent members of n-ary collections, with $n > 1$.

On the right-hand side of Figure 4.10, three additional types of database values are shown. First, values described by interface OMSBaseType

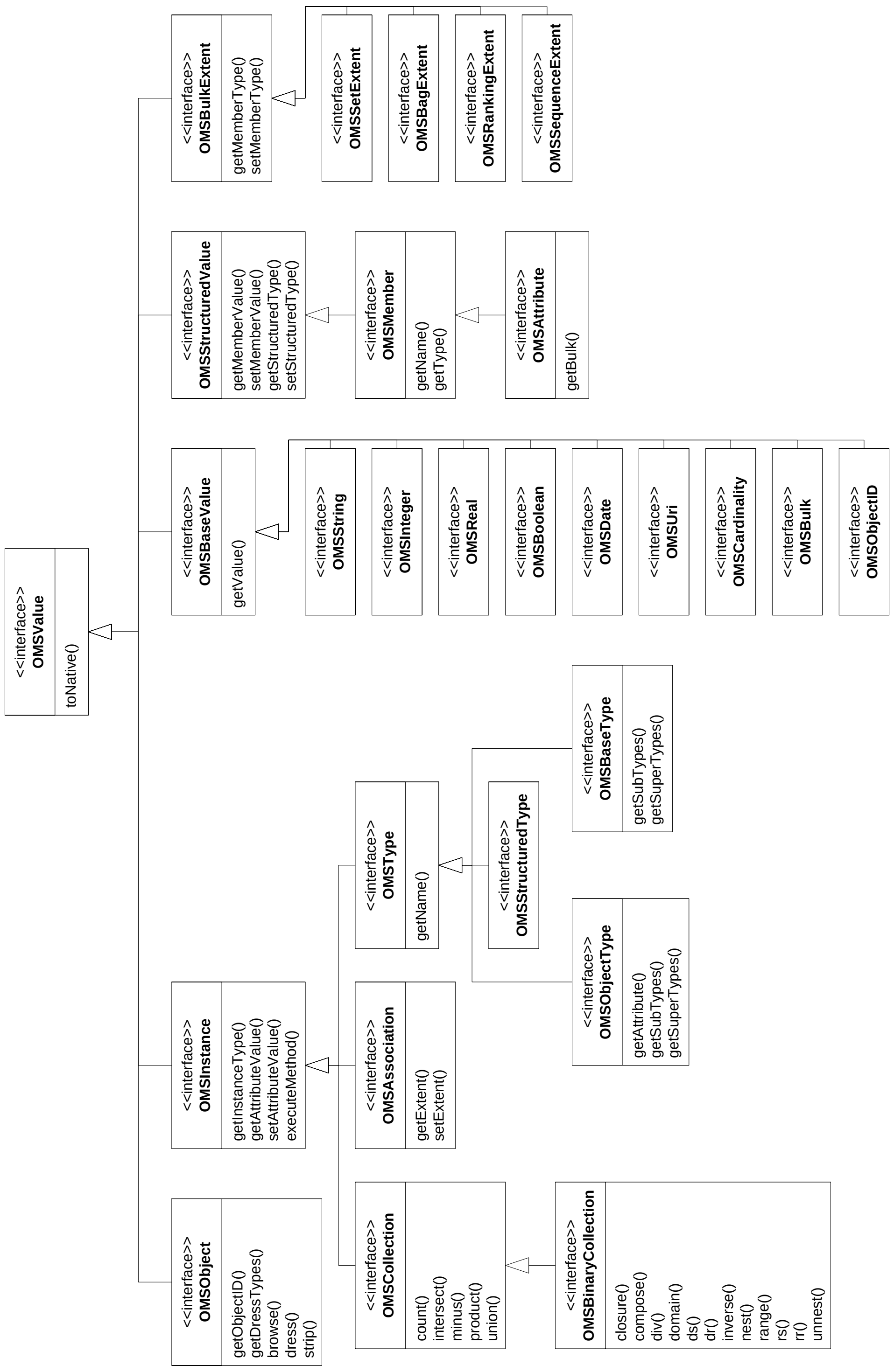

Figure 4.10: OMS$^{\mathrm{jp}}$ value interface

are represented in OMS^{jp} through interface OMSBaseValue and its specialisations. These values correspond to the basic data types defined in all OMS platforms, such as string, integer, real or date. Interface OMSStructuredValue represents database values that have been defined as records by an instance of OMSStructuredType. Two specialisations of this class are featured in the interface to facilitate access to metadata. Interfaces OMSMember and OMSAttribute provide access to the members and the attributes of a structured and an object type, respectively. The purpose of the third kind of values is to represent the collection extents. As introduced in the description of the OM data model, an extent value is either a set, a bag, a ranking or a sequence, depending on the type of collection. To cater for the different semantics and the different operations associated with these types of extent values, the OMS^{jp} interface defines four specialisations, OMSSetExtent, OMSBagExtent, OMSRankingExtent and OMSSequenceExtent, of interface OMSBulkExtent.

The OMS^{jp} interface has been implemented for several OMS platforms by providing a driver that maps the presented concepts to a database back-end. However, a comprehensive discussion of the platform-dependent details of such a driver implementation is out of the scope of this thesis. Nevertheless, it is important to examine the relationship between an object and its instances further by looking at the classes found in the abstract implementation which is the common basis of all platform drivers. It is this relationship that has to be modified in order to incorporate versioning into the implementation of OMS^{jp}. Figure 4.11 contains an excerpt of the original implementation of OMS^{jp} that shows the three most important classes involved in the relationship between an object and its instances. Represented in UML, these three classes are OMSObjectImpl, OMSIntanceImpl and OMSObjectTypeImpl, which implement the previously introduced interfaces OMSObject, OMSIntance and OMSObjectType, respectively.

As shown, class OMSObjectImpl stores all properties that are inherent to the concept of an object in the OM data model. The field oid stores a reference to the object's identifier represented as an instance of class OMSObjectID which is not shown in the figure. The object identifier can be obtained using method getObjectID() but it cannot be set. Object identifiers are assigned to an object when it is created by the underlying platform and are therefore read-only. As an OM object can have multiple types, class OMSObjectImpl manages a collection types with references to all types of the object represented by instances of class OMSObjectTypeImpl. This relationship between an object and its types is also represented

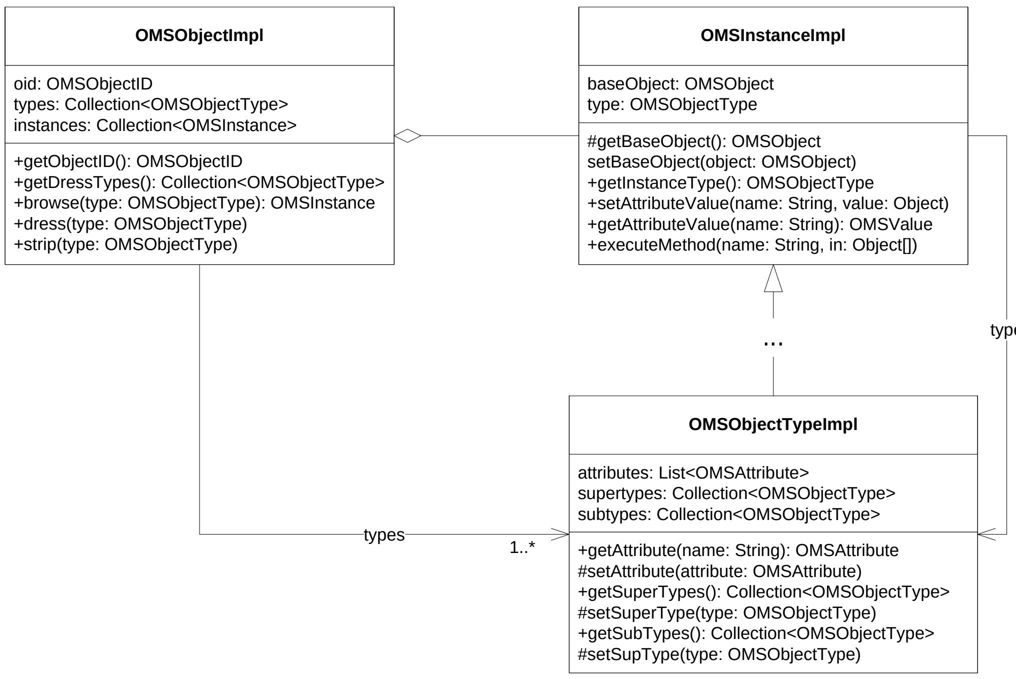
OMSObjectImpl
oid: OMSObjectID
types: Collection<OMSObjectType>
instances: Collection<OMSInstance>
+getObjectID(): OMSObjectID
+getDressTypes(): Collection<OMSObjectType>
+browse(type: OMSObjectType): OMSInstance
+dress(type: OMSObjectType)
+strip(type: OMSObjectType)
OMSInstanceImpl
baseObject: OMSObject
type: OMSObjectType
#getBaseObject(): OMSObject
setBaseObject(object: OMSObject)
+getInstanceType(): OMSObjectType
+setAttributeValue(name: String, value: Object)
+getAttributeValue(name: String): OMSValue
+executeMethod(name: String, in: Object[])
...
type
OMSObjectTypeImpl
attributes: List<OMSAttribute>
supertypes: Collection<OMSObjectType>
subtypes: Collection<OMSObjectType>
+getAttribute(name: String): OMSAttribute
#setAttribute(attribute: OMSAttribute)
+getSuperTypes(): Collection<OMSObjectType>
#setSuperType(type: OMSObjectType)
+getSubTypes(): Collection<OMSObjectType>
#setSupType(type: OMSObjectType)
types
1..*

Figure 4.11: Excerpt of the OMS^{jp} implementation

in the figure by the aggregation between the corresponding UML classes. The types of an object can be retrieved using the getDressTypes() method that returns the set of types the object is currently dressed with. To modify this collection of types, methods dress() and strip() can be used to add or remove object types. While dressing an object with a given object type leads to the creation of a new instance of the object, stripping an object of a given type deletes the corresponding instance. The current set of instances of an object is maintained by the field instances that contains references to instances of class OMSInstanceImpl. This relationship between an object and its instances is again represented using an aggregation between the corresponding UML classes. Note, that members of the instances collection can not be added or removed directly, but rather through the use of methods dress() and strip() only.

A striking property of the OMSObject interface implemented by class OMSObjectImpl is the fact that it lacks any methods that would allow access to the object's data. In contrast to object-oriented systems such as Java that feature single instantiation, OM allows its objects to have instances from parallel or even unrelated inheritance hierarchies. As a consequence, the type of an object cannot be determined without additional information about the role in which the object is currently browsed. Including methods providing access to an object's data in the OMSObject interface would therefore potentially lead to ambiguous situations, as it would allow the object to be accessed without the required role information. As a solution to this situation, OMS$^{\text{jp}}$ offers the browse() method in the OMSObject interface together with the interface OMSInstance that provides the required methods to retrieve and update data on the object as well as executing its methods. As browse() expects an argument of type OMSObjectType, it sets the role in which the object is browsed and eliminates any ambiguities by returning the corresponding instance of OMSInstance. Class OMSInstanceImpl implements the OMSInstance interface and thus provides methods getAttributeValue() and setAttributeValue() to read and write the values of an object's attributes. To invoke a method, its name and the corresponding input parameters have to be given to the executeMethod() method which evaluates the method and returns any result values computed by the method. Also visible in the figure is the fact that class OMSInstanceImpl maintains a reference to its base object and to the type of which it is an instance. This information can be accessed using the getBaseObject() and setBaseObject() methods as well as method getInstanceType(). In contrast to the methods discussed so far, the method providing read-access to the base object is protected as this functionality should only be open to the subclasses of OMSInstanceImpl. Write-access to the base object of an instance is even more restricted, as the corresponding method has default access and thus is only open to other classes in the same package.

Class OMSObjectTypeImpl is a specialisation of class OMSInstanceImpl that represents an object instance that describes the attributes and methods of other objects. To do so, class OMSObjectTypeImpl maintains a list of OMSAttribute instances that describe which attributes an instance of this type will have. As the order of these attributes is important, they are managed as a list. The attributes defined by a type can be retrieved by name using the getAttribute() method. A new attribute can be appended to the type using setAttribute(). In addition to the metadata about a type's attributes, class OMSObjectTypeImpl also maintains information about the type's supertypes and subtypes. As the OM data model allows types to inherit from multiple supertypes, the references to both supertypes and subtypes are stored in a collection. As these collections cannot be accessed directly, the interface of class OMSObjectTypeImpl provides methods to retrieve and insert both supertypes and subtypes. Similar to class OMSInstanceImpl, methods providing read-only access are public and can be invoked by any other class. Methods that modify the metadata, however, have protected access to ensure that they are not called from the outside. To modify the database schema in OMS$^{\text{jp}}$, the previously discussed database concept offered by the interface has to be used. In contrast to the representations of metadata objects, the database concept also takes care of the required data evolution operations that have to be executed to ensure database consistency.

To accommodate versioned objects, as defined at the beginning of this chapter, the OMS^{jp} interface and its implementation has been extended with the necessary concepts. As point of extension we have chosen the driver that manages the database back-end of an OMS^{jp} interface implementation. In our extended implementation, the driver maps the versioned objects visible to the application programmer to the metamodel, shown in Figure 4.4, which is used to manage versioned objects at the storage level. Since the driver now takes on the management of revisions and variants as well as version-aware query processing, in addition to the mapping between OM concepts and Java concepts, it effectively becomes a part of the data management system itself. Apart from the back-end implementation, the OMS^{jp} interface itself has to be adapted as well. At the heart of this augmented interface is the extended model of an OM object shown in Figure 4.12 as a UML class diagram. For reasons of conciseness, we have decided not to discuss the interface and implementation of the extended object representation of OMS^{jp} in two separate figures. Although the concepts shown in the figure are labelled with the names of the interfaces introduced previously, they nevertheless represent classes materialising these interfaces where all methods that have public access are implementations of interface methods. Thus, the interface corresponding to any of the classes shown can be derived from the name of the class and the set of its public methods. Methods and fields that retain their meaning and functionality from the discussed basic OMS^{jp} object model are shown in grey, while extensions to the model are set in black.

Before presenting each of the adapted classes in some more detail, it is worthwhile discussing the general structure of the new object model first. The most visible change is the introduction of a new class OMS-ObjectVersion that has been included as an additional step of indirection between the object and its instances. The changes that have been made to the classes OMSObject and OMSInstance are mostly direct results of the new structure of the object model. In contrast to the storage layer that uses two distinct concepts for revisions and variants, the interface layer only uses the concept of a single version representation that represents both revisions and variants. The motivation for this deviating representation of a versioned object is rooted in the different requirements that

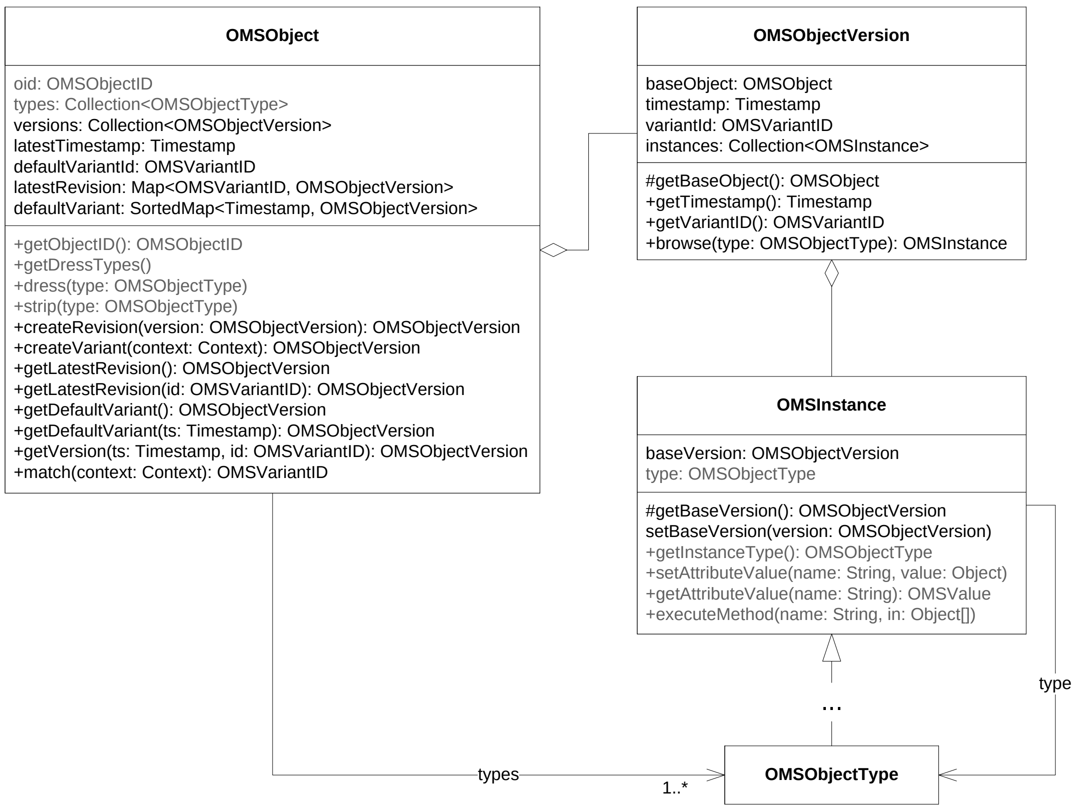
OMSObject
oid: OMSObjectID
types: Collection<OMSObjectType>
versions: Collection<OMSObjectVersion>
latestTimestamp: Timestamp
defaultVariantId: OMSVariantID
latestRevision: Map<OMSVariantID, OMSObjectVersion>
defaultVariant: SortedMap<Timestamp, OMSObjectVersion>
+getObjectID(): OMSObjectID
+getDressTypes()
+dress(type: OMSObjectType)
+strip(type: OMSObjectType)
+createRevision(version: OMSObjectVersion): OMSObjectVersion
+createVariant(context: Context): OMSObjectVersion
+getLatestRevision(): OMSObjectVersion
+getLatestRevision(id: OMSVariantID): OMSObjectVersion
+getDefaultVariant(): OMSObjectVersion
+getDefaultVariant(ts: Timestamp): OMSObjectVersion
+getVersion(ts: Timestamp, id: OMSVariantID): OMSObjectVersion
+match(context: Context): OMSVariantID
OMSObjectVersion
baseObject: OMSObject
timestamp: Timestamp
variantId: OMSVariantID
instances: Collection<OMSInstance>
#getBaseObject(): OMSObject
+getTimestamp(): Timestamp
+getVariantID(): OMSVariantID
+browse(type: OMSObjectType): OMSInstance
OMSInstance
baseVersion: OMSObjectVersion
type: OMSObjectType
#getBaseVersion(): OMSObjectVersion
setBaseVersion(version: OMSObjectVersion)
+getInstanceType(): OMSObjectType
+setAttributeValue(name: String, value: Object)
+getAttributeValue(name: String): OMSValue
+executeMethod(name: String, in: Object[])
...
type
types
1..*
OMSObjectType

Figure 4.12: Extended object model in OMS^{jp}

need to be addressed at different levels of the implementation. At the storage level, the representation of a versioned object has been designed to favour the evaluation of certain kinds of queries. The needs of database application developers are the primary influence that has shaped the structure of the extended OMS^{jp} interface. At this level it is important the keep the number of concepts small in order to provide an elegant and minimal interface. Therefore, we have decided that the differentiation between revisions and variants should not be visible in terms of interface classes but rather in the functionality offered by these classes.

As shown in the figure, class OMSObject continues to be the principal Java representation of an object as defined by the OM data model. Apart from the instances field and the browse() method that are no longer a part of this class, all other methods and fields continue to have the same purpose as discussed above. In particular, all methods that affect the object's set of types remain at the level of the object itself to ensure that all versions of the object have the same set of types at any point in time as required by our definition of the version model. In addition to these class members, the extended representation of an OM object defines a set of fields and methods that manage revisions and variants.

For example, the versions field maintains a collection of references to all versions of the object represented as instances of class OMSObjectVersion. The relationship between an object and its versions is also explicitly shown in the figure as an aggregation of two UML classes. The class interface does not provide direct access to the collection of versions but allows new revisions and variants to be created using the methods createRevision() and createVariant(). As an object can have multiple variants, the application developer needs to specify the version of the object for which they would like to create a new revision. However, the creation of a new revision is only successful if the given version is one of the latest revisions of the object. Otherwise, the call to createRevision() fails and an exception is raised. Our definition of the version model requires all variants of an object to specify a set of context values that is different from the properties of all other variants. Therefore, whenever a variant is created, the application developer needs to state the context in which the variant will be suitable. Method createVariant() then checks if the given set of context values fulfils this constraint prior to creating the variant in the storage layer and raises an exception if a variant for the same context already exists.

To provide efficient access to both the latest revision and the default variant, class OMSObject further maintains the two look-up tables latestRevision and defaultVariant. The interface method getLatestRevision() accesses the first of these look-up tables to retrieve the latest revision of the object. While the latest revision of the default variant can be obtained by using getLatestRevision() without parameters, the latest revision of a specific variant is retrieved by the implementation that takes the corresponding argument. If no variant identifier is specified, the latest revision is computed based on the value stored in field defaultVariantId. The default variant is accessed analogously by using the method getDefaultVariant() of the interface of OMSObject. If no parameter is given, the latest revision of the default variant is returned, based on the value captured by the field latestTimestamp. However, if the application designer specifies a certain point in time as an argument to the method, the latest version smaller or equal to the given value is retrieved. To facilitate this computation, defaultVariant is implemented as a sorted map that maintains an ordering over the sequence of recorded timestamps and therefore supports range queries. To access an arbitrary version of the object, the interface of class OMSObject provides the getVersion() method that allows both the specification of a time-stamp and a variant identifier. The matching algorithm presented in the previous section is invoked by calling method match() and passing the current context as an argument. The method matches the given set of context values to the properties of the available variants and returns the identifier of the best matching alternative. This identifier can then be used to access the variant version itself, by using the methods we outlined earlier. Although the interface of the object class provides methods to create new revisions and variants, there are no counterpart methods that would allow these versions to be removed. The simple reason for this omission is the fact that the semantics of such an operation are not defined by our version model.

As mentioned before, the inclusion of class OMSObjectVersion presents the only structural change distinguishing the extended from the original value interface. An instance of this class represents a version of the base object it is associated with. It therefore maintains a reference to the base object in the baseObject field. The base object is accessible to descendants of the class and other classes in the same package through method getBaseObject() that has protected access. A version of an object is identified by the values stored in fields timestamp and variantId. Similar to the object identifier in class OMSObject, both fields can be accessed by methods getTimestamp() and getVariantID(). Also, as the identifier of the object, these values cannot be updated as they have to remain constant throughout a version's existence. The browse() method that was part of the original interface of OMSObject, as shown in Figure 4.10, has been moved to the class representing a version of an object. The reason for this refactoring is that the object in the extended model is no longer directly connected to its instances. Therefore, this functionality is now offered as part of the interface of class OMSObjectVersion that also maintains a set of references to the object's instances in field instances. Finally, the only changes made to class OMSInstance reflect the fact that it is associated with an instance of class OMSObjectVersion in the extended model, instead of an instance of class OMSObject as in the original object model. Consequently, field baseVersion replaces the previously defined field baseObject and the two access methods to this field have been renamed accordingly.

4.5 Discussion

As a response to the requirements of context-aware applications, we have presented a version model that provides support for the management of alternative versions of data objects while at the same time supporting the system development process with the provision of revisional versions. The model has been defined formally within the framework of the OM data model, an extended E/R model for object-oriented data management. Based on a comparison of the original metamodel of an object as defined by OM to the extended model that introduces a set of new concepts and constraints, we have shown how our model accommodates revisional and alternative versions. An important characteristic of this extended model is how objects are identified and referenced. To allow both generic and specific references, the concepts of a latest revision and a default variant have been presented. Based on these types of references, we have demonstrated how relationships between objects can be versioned using the same concepts as for the objects themselves.

At the heart of our context-aware data management stands an algorithm that matches the current context of an application to the variants of the objects involved in a query. For each variant, the matching algorithm computes a score and then selects the one with the highest score as the representation of the object according to the context. In our version model, context is seen as information that can be used to augment the result of a query rather than a specification that has to be followed strictly. Hence, instead of computing an exact match between the context and the set of object variants, the algorithm uses a best match approach, with the default variant as fall-back option, if no variant scores above a predefined threshold. As system developers have to be given as much control as possible, our matching algorithm further supports the notion of required and illegal values for context dimensions.

Many of the ideas and concepts presented in this chapter have their origin in solutions proposed for temporal and engineering databases as well as software configuration systems. Although all of these systems are capable of either supporting revisional, alternative or even both kinds of versions, none of them has been developed for the management of context-aware data. We argue that this difference in the area of application of a version model gives rise to new requirements that need to be addressed by adapting and extending concepts introduced earlier. As mentioned before, the main difference between existing solutions and our approach is the interpretation of information that exists in addition to the query. As this information is seen as a part of the query in temporal and engineering databases as well as in software configuration systems, these systems are not at liberty to deliver a result representing a best effort. If an object has no version that matches this additional specification exactly, the query processing needs to be aborted and an error reported. For example, if a software configuration system is used to build a system for a certain platform and one module is not available for this platform, it generally cannot be substituted by a version intended for a different platform.

As our model assumes that this additional context information is orthogonal to the query, there are several important implications that distinguish our system from existing solutions. Although most existing version models feature the concept of a latest version and a default variant, their purpose is completely different. In those systems, these two concepts exist only to cope with queries that do not use or are not aware of versions. While this purpose persists in our model, the default variant is also used as a fall-back representation of the object which avoids exceptions during query processing. Such a use of this concept would never be possible in the example of a software configuration system, as it would inevitably lead to faulty results, and therefore exceptions are vital information to the users of such systems. In contrast, having a fall-back variant that is always possible is the enabling factor for a best match algorithm instead of an exact match algorithm. The importance of this approach becomes clear when we go back once again and compare the nature of the additional information in both existing solutions and our version model. Apart from its interpretation, another key difference is where this information originates. In the case of an engineering database or a software configuration system, for example, both the type and the values of this information are defined by the same developers that define the alternative versions. In a context-aware application, however, the developer of the database and the client specifying the context are two different actors. Therefore, our model for context-aware data management has to cope with context configurations that have not been envisioned at design-time. In this setting, using an exact match approach is not possible for two reasons. On the one hand, even small changes in the context values could lead to situations where our system would not be able to deliver satisfactory or even any results. On the other hand, using an exact match would require a variant to be defined for each context configuration. Clearly, such an approach is not feasible, due to reasons of scalability.

The different application of the version models is also manifested in the operational model defined for each solution. In temporal databases, versions are normally created automatically when data is modified based on the current time-stamp, even though systems exist that also provide explicit operations for this purpose. Engineering databases and software configuration systems do not create revisions and variants by default but define a set of commands that allow the version graph of an object to be manipulated. In this respect, our version model is more like these latter systems, as we have not defined a mechanism for automatic version generation, such as a high-level operational model or policies that control when versions are created by the system. In contrast to many of those systems that use the library model of operations where objects are checked out, modified, and later checked in, our system uses an operational model based on the interface presented in the previous section. Both engineering databases and software configuration systems support a merge operation that is used to combine two alternative branches of revisions. As, in practice, this operation will almost always involve some kind of user interaction to resolve conflicts between the variants involved, it is virtually impossible to offer an automatic merge operation. Also, in the setting of context-aware data management, the semantics of merging two variants are not defined and therefore we believe that it is not very likely that this operation will be required. Hence, we have abstained from including a default implementation of this operation in our version model and leave its implementation, if required, to the application developer.

Query processing, or building a configuration, as it is called in software configuration systems, is another area where our system distinguishes itself from existing solutions. If information that is provided in addition to the query is considered to be a form of specification, the query processor has to ensure that the returned result is consistent with the specification in its entirety. Software configuration systems, for example, have proposed a technique where the specification is becoming more and more complete with every object that is included. To do so, the values of parameters that have not given to the query are inserted into the specification, based on unused values obtained from the descriptions of the selected variants. This approach results in a stable and consistent configuration but its outcome highly depends on the order in which objects are accessed. Other solutions have proposed building the entire query tree first and then computing a matching over all objects. While this algorithm leads to a deterministic and consistent result, it is not usable in on-line systems such as our own, for reasons of performance. Our notion of context allows the matching of context to be effected at the time each object is accessed by the query processor. The task of augmenting an existing query evaluator with support for our notion of context-awareness is therefore not a complex one.

Much of the power of the OM data model lies in the fact that the model is completely specified by a metamodel that is also expressed in

OM. As a consequence, most of the OMS platforms that provide native support for databases designed with OM use objects to store both data and metadata. In other words, concepts such as collections, associations, methods or types are all represented and stored as OM objects. Therefore, traditional OMS platforms provide an elegant way of managing, accessing and querying data and metadata in a uniform way. With our extended object model, however, the question has to be raised whether to apply its additional functionality to metadata objects as well. The two extreme solutions—disallowing or allowing all metadata to have versions—do not promise to be very useful. Whereas the first approach imposes an unnecessary restriction that prevents a concept from being used in places where it could be wise to do so, the second solution will undoubtedly lead to a system that would be hard to understand, maintain and control. Hence, we believe, that a selected number of metadata concepts should be allowed to have versions while others continue to exist without them. Ideal candidates among the different types of metadata objects are concepts that are already hard to classify as either data or metadata such as, for example, collections and associations. As previously discussed, versioned associations in particular constitute a powerful concept to track the evolution of entire object systems, as they capture the relationships between objects. Another metadata concept that benefits from the additional functionality of our version model are method objects. Whereas revisional versions are an ideal solution to keep track of the development of a method, alternative versions of a method object can be used to provide context-aware behaviour that goes beyond the presented query processing algorithm.

We firmly believe that the presented version model for context-aware data management has the potential to address the requirements of context-aware applications and to facilitate their development. Nevertheless, in its current form, the version model also has a few limitations that will have to be addressed in the future. In the following, we discuss a few of these shortcomings and provide some thoughts as to how they could be overcome. A drawback of our model is the assumption that an object is either entirely context-dependent or it is not. However, in reality, this is seldom the case as often only parts of an object change depending on the current context. Imagine, for example, an object used to represent a person. Some attributes, such as text containing the biography or a short curriculum vitae, can vary according to context dimensions such as the language in which the object is accessed. Then again, other attributes, such as the name of the person or their phone number, remain the same in all contexts. In a way, this issue relates back to the question that arose in temporal databases, whether the entire tuple or individual attributes should be versioned. Our decision to apply versioning at the level of objects is comparable to the notion of tuple-versioning introduced in these systems. Therefore our model is not able to distinguish between context-dependent and context-independent information within an object, which could lead to data being replicated in all variants. Clearly, storing redundant information requires great care when updating these values, and can lead to consistency anomalies. Fortunately, there are solutions to this shortcoming in our system. Context-independent values could be represented by a special null value in all variants except the default variant. This special value would cause the system to access the default variant whenever this attribute is read in a version that has no data defined for it. If this additional level of indirection implicates a performance penalty that outweighs the benefits of having consistent data, the metamodel of a versioned object would need to be revised and manage context-dependent and context-independent data separately. This second solution relates back to version models that provide the concept of a generic object that represents the properties that are common to all object versions.

While we have presented several techniques to improve the quality of results computed by the matching algorithm, there are still many cases where the outcome is ambiguous. The fact that the matching algorithm needs to resort to the default version in this case is highly unsatisfactory as it prevents our system from tapping its full potential. Therefore, we believe that further refinements of the matching function are necessary. One possible starting point is the different classes of matches that were introduced in Table 4.4. As of now, matches from all classes are considered to be equally good and thus weighted the same. However, it is reasonable to assume that, for example, a variant that is matched with a wildcard is of lower importance than one that specifies the exact value. As a further improvement of the matching algorithm, the presented weighting function could be extended to assign different weights to matches, depending on the class that has been used to specify the context value. Related to this issue is the drawback that our value syntax does not support the definition of priority lists or rankings. Instead of being limited to simply specifying a set of acceptable values for a context dimension, a context should be able to express preferences for these values. For example, if a system considers language as a contextual factor, then it should be possible to declare that English takes precedence over

German. As a consequence, a variant designed for the German context would be assigned a lower matching value than the one for the English context. This behaviour could easily be implemented by extending the scoring function to deal with an additional given set that represents such rankings. However, the integration of a given set RANKING would render the whole system more complex. The need for such a class of values has, therefore, to be underpinned by additional application scenarios.

Finally, another deficiency of our approach pertains to the implementation of versions on the storage level. As presented in Section 4.4, our version model has been realised as part of a database library providing access to different OMS platforms. As our implementation is not integrated into the database system itself, it has to work with the concepts offered by these platforms, instead of being able to access their internal structures. As an example, it would be desirable to use deltas to minimise the space required for storing versions. Instead, every version has to be represented as an object in the underlying database. Apart from space consumption, this approach suffers from additional undesirable characteristics. As our layered approach operates at a relatively high level, it has only limited control over how data is managed. Hence, it is difficult to provide index structures that would allow objects and their versions to be accessed efficiently. As a consequence, the performance of query evaluation in our system does not compare well to a traditional database system. While additional versioning functionality will always be coupled to a performance penalty, we believe that this shortcoming is best addressed by truly integrating the concepts presented in this chapter into the core of a database management system.

Bibliography

[1] G. D. Abowd. Classroom 2000: An Experiment with the Instrumentation of a Living Educational Environment. *IBM Systems Journal*, 38(4):508–530, 1999.

[2] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton. Cyberguide: A Mobile Context-Aware Tour Guide. *Wireless Networks*, 3(5):421–433, 1997.

[3] G. D. Abowd, C. G. Atkeson, J. Brotherton, T. Enqvist, P. Gulley, and J. LeMon. Investigating the Capture, Integration and Access Problem of Ubiquitous Computing in an Educational Setting. In *Proceedings of ACM SIGCHI Conference on Human Factors in Computing Systems, April 18-23, 1998, Los Angeles, CA, USA*, pages 440–447, 1998.

[4] R. Ahmed and S. B. Navathe. Version Management of Composite Objects in CAD Databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data, May 29-31, 1991, Denver, CO, USA*, pages 218–227, 1991.

[5] I. Ahn. Towards an Implementation of Database Management Systems with Temporal Support. In *Proceedings of International Conference on Data Engineering, February 5-7, 1986, Los Angeles, CA, USA*, pages 374–381, 1986.

[6] I. Ahn and R. T. Snodgrass. Partitioned Storage for Temporal Databases. *Information Systems*, 13(4):369–391, 1988.

[7] G. Ariav. A Temporally Oriented Data Model. *ACM Transactions on Database Systems*, 11(4):499–527, 1986.

[8] R. Audi, editor. *The Cambridge Dictionary of Philosophy.* Cambridge University Press, 1999.

[9] C. W. Bachman. Data Structure Diagrams. *ACM SIGMIS Database*, 1(2):4–10, 1969.

[10] M. Baldauf, S. Dustdar, and F. Rosenberg. A Survey on Context-Aware Systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007.

[11] F. Bancilhon, P. Kanellakis, and C. Delobel, editors. *Building an Object-Oriented Database System: The Story of O_2*. Morgan Kaufmann Publishers, 1992.

[12] Y. Bar-Hillel. Indexical Expressions. *Mind*, 63(251):359–379, 1954.

[13] P. Barna, G.-J. Houben, and F. Frăsincar. Specification of Adaptive Behavior Using a General-Purpose Design Methodology for Dynamic Web Applications. In *Proceedings of Adaptive Hypermedia and Adaptive Web-Based Systems, August 24-26, 2004, Eindhoven, The Netherlands*, pages 283–286, 2004.

[14] D. S. Batory and W. Kim. Modeling Concepts for VLSI CAD Objects. *ACM Transactions on Database Systems*, 10(3):322–346, 1985.

[15] H. Baumeister, N. Koch, and L. Mandel. Towards a UML Extension for Hypermedia Design. In *Proceedings of International Conference on the Unified Modeling Language, October 28-30, 1999, Fort Collins, CO, USA*, pages 614–629, 1999.

[16] H. Baumeister, A. Knapp, N. Koch, and G. Zhang. Modelling Adaptivity with Aspects. In *Proceedings of International Conference on Web Engineering, July 27-29, 2005, Sydney, Australia*, pages 406–416, 2005.

[17] J. Baus, A. Krüger, and W. Wahlster. A Resource-Adaptive Mobile Navigation System. In *Proceedings of International Conference on Intelligent User Interfaces, January 13-16, 2002, San Francisco, CA, USA*, pages 15–22, 2002.

[18] D. Beckett, editor. RDF/XML Syntax Specification (Revised). `http://www.w3.org/TR/rdf-syntax-grammar/`, February 2004.

[19] D. Beech and B. Mahbod. Generalized Version Control in an Object-Oriented Database. In *Proceedings of International Conference on Data Engineering, February 1-5, 1988, Los Angeles, CA, USA*, pages 14–22, 1988.

[20] N. Belkatir and J. Estublier. Experience with a Data Base of Programs. In *Proceedings of ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, December 9-11, 1986, Palo Alto, CA, USA*, pages 84–91, 1987.

[21] R. Belotti. SOPHIE – Context Modelling and Control. Diploma thesis, Swiss Federal Institute of Technology Zurich (ETH Zurich), Zurich, Switzerland, 2004.

[22] R. Belotti, C. Decurtins, M. Grossniklaus, M. C. Norrie, and A. Palinginis. Modelling Context for Information Environments. In *Proceedings of International Workshop on Ubiquitous Mobile Information and Collaboration Systems, June 7-8, 2004, Riga, Latvia*, pages 43–56, 2004.

[23] R. Belotti, C. Decurtins, M. Grossniklaus, M. C. Norrie, and A. Palinginis. Interplay of Content and Context. In *Proceedings of International Conference on Web Engineering, July 28-30, 2004, Munich, Germany*, pages 187–200, 2004.

[24] R. Belotti, C. Decurtins, M. Grossniklaus, and M. C. Norrie. An Infrastructure for Reactive Information Environments. In *Proceedings of International Conference on Web Information Systems Engineering, November 20-22, 2005, New York, NY, USA*, pages 347–360, 2005.

[25] R. Belotti, C. Decurtins, M. Grossniklaus, M. C. Norrie, and A. Palinginis. Interplay of Content and Context. *Journal of Web Engineering*, 4(1):57–78, 2005.

[26] G. Benelli, A. Bianchi, P. Marti, D. Sennati, and E. Not. HIPS: Hyper-Interaction within Physical Space. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems, June 7-11, 1999, Florence, Italy*, pages 1075–1078, 1999.

[27] M. Benerecetti, P. Bouquet, and C. Ghidini. On the Dimensions of Context Dependence: Partiality, Approximation, and Perspective.

In *Proceedings of International and Interdisciplinary Conference on Modeling and Using Context, July 27-30, 2001, Dundee, Scotland, UK*, pages 59–72, 2001.

[28] Y. Bernard, M. Lacroix, P. Lavency, and M. Vanhoedenaghe. Configuration Management in an Open Environment. In *Proceedings of European Software Engineering Conference, September 9-11, 1987, Strasbourg, France*, pages 35–43, 1987.

[29] T. Berners-Lee and M. Fischetti. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor.* Harper San Francisco, 1999.

[30] G. Biegel and V. Cahill. A Framework for Developing Mobile, Context-Aware Applications. In *Proceedings of IEEE International Conference on Pervasive Computing and Communications, March 19-23, 2004, Orlando, FL, USA*, pages 361–365, 2004.

[31] A. Binemann-Zdanowicz, R. Kaschek, K.-D. Schewe, and B. Thalheim. Context-aware Web Information Systems. In *Proceedings of Asian-Pacific Conference on Conceptual Modelling, January 18-22, 2004, Dunedin, New Zealand*, pages 37–48, 2004.

[32] A. F. Bobick, S. S. Intille, J. W. Davis, F. Baird, C. S. Pinhanez, L. W. Campbell, Y. A. Ivanov, A. Schütte, and A. Wilson. The KidsRoom: A Perceptually-Based Interactive and Immersive Story Environment. *Presence: Teleoperators and Virtual Environments*, 8(4):369–393, 1999.

[33] A. Bongio, S. Ceri, P. Fraternali, and A. Maurino. Modeling Data Entry and Operations in WebML. In *Selected Papers from the Third International Workshop WebDB 2000 on the World Wide Web and Databases*, pages 201–214, 2001.

[34] C. Boyle and A. O. Encarnacion. MetaDoc: An Adaptive Hypertext Reading System. *User Modeling and User-Adapted Interaction*, 4(1):1–19, 1994.

[35] D. Brickley and R. V. Guha, editors. RDF Vocabulary Description Language 1.0: RDF Schema. `http://www.w3.org/TR/rdf-schema/`, February 2004.

[36] J. A. Brotherton and G. D. Abowd. Lessons Learned from eClass: Assessing Automated Capture and Access in the Classroom. *ACM Transactions on Computer-Human Interaction*, 11(2): 121–155, 2004.

[37] B. Brown and E. Laurier. Designing Electronic Maps: An Ethnographic Approach. In L. Meng, A. Zipf, and T. Reichenbacher, editors, *Map Design for Mobile Applications*, pages 247–262. Springer Verlag, 2004.

[38] P. J. Brown. The Stick-e Document: A Framework for Creating Context-Aware Applications. *Electronic Publishing*, 8(2-3):259–272, 1995.

[39] P. J. Brown. Triggering Information by Context. *Personal and Ubiquitous Computing*, 2(1):18–27, 1998.

[40] P. J. Brown, J. D. Bovey, and X. Chen. Context-Aware Applications: From the Laboratory to the Marketplace. *IEEE Personal Communications*, 4(5):58–64, 1997.

[41] P. Brusilovsky. Methods and Techniques of Adaptive Hypermedia. *User Modeling and User-Adapted Interaction*, 6:87–129, 1996.

[42] P. Brusilovsky, E. W. Schwarz, and G. Weber. ELM-ART: An Intelligent Tutoring System on World Wide Web. In *Proceedings of International Conference on Intelligent Tutoring Systems, June 12-14, 1996, Montréal, Canada*, pages 261–269, 1996.

[43] J. A. Bubenko. The Temporal Dimension in Information Modeling. In *Architecture and Models in Data Base Management Systems (Proceedings of IFIP Working Conference on Modelling in Data Base Management Systems, January 3-7, 1977, Nice, France)*, pages 93–118, 1977.

[44] J. A. Bubenko. Information Modelling in the Context of System Development. In *Information Processing 80 (Proceedings of IFIP Congress 80, October 6-9, 1980, Tokyo, Japan, October 14-17, 1980, Melbourne, Australia)*, pages 395–411, 1980.

[45] P. Büchler. XSLGui – A Graphical Template Editor for Web Content Managament. Diploma thesis, Swiss Federal Institute of Technology Zurich (ETH Zurich), Zurich, Switzerland, 2003.

[46] J. Burrell and G. K. Gay. Collectively Defining Context in a Mobile, Networked Computing Environment. In *Proceedinsg of ACM SIGCHI Conference on Human Factors in Computing Systems, March 31-April 5, 2001, Seattle, WA, USA*, pages 231–232, 2001.

[47] J. Burrell and G. K. Gay. E-Graffiti: Evaluating Real-World Use of a Context-Aware System. *Interacting with Computers*, 14(4): 301–312, 2002.

[48] P. Butterworth, A. Otis, and J. Stein. The GemStone Object Database Management System. *Communications of the ACM*, 34(10): 64–77, 1991.

[49] M. J. Carey and D. J. DeWitt. A Data Model and Query Language for EXODUS. *ACM SIGMOD Record*, 17(3):413–423, 1988.

[50] S. Casteleyn, O. De Troyer, and S. Brockmans. Design Time Support for Adaptive Behavior in Web Sites. In *Proceedings of ACM Symposium on Applied Computing, March 9-12, 2003 Melbourne, FL, USA*, pages 1222–1228, 2003.

[51] P. Castro and R. R. Muntz. Managing Context Data for Smart Spaces. *IEEE Personal Communications*, 7(5):44–46, 2000.

[52] P. Castro, P. Chiu, T. Kremenek, and R. R. Muntz. A Probabilistic Room Location Service for Wireless Networked Environments. In *Proceedings of International Conference on Ubiquitous Computing, September 30-October 2, Atlanta, GA, USA*, pages 18–34, 2001.

[53] D. Caswell and P. Debaty. Creating Web Representations for Places. In *Proceedings of International Symposium on Handheld and Ubiquitous Computing, September 25-27, 2000, Bristol, England, UK*, pages 114–126, 2000.

[54] S. Ceri, P. Fraternali, and S. Paraboschi. Data-Driven, One-To-One Web Site Generation for Data-Intensive Applications. In *Proceedings of International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 615–626, 1999.

[55] S. Ceri, P. Fraternali, and A. Bongio. Web Modeling Language (WebML): A Modeling Language For Designing Web Sites. *Computer Networks*, 33(1-6):137–157, 2000.

[56] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing Data-Intensive Web Applications.* The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann Publishers Inc., 2002.

[57] S. Ceri, F. Daniel, and M. Matera. Extending WebML for Modeling Multi-Channel Context-Aware Web Applications. In *Proceedings of International Workshop on Multichannel and Mobile Information Systems, December 13, 2003, Roma, Italy*, pages 225–233, 2003.

[58] S. Ceri, P. Dolog, M. Matera, and W. Nejdl. Adding Client-Side Adaptation to the Conceptual Design of e-Learning Web Applications. *Journal of Web Engineering*, 4(1):21–37, 2005.

[59] S. Ceri, F. Daniel, and F. M. Facca. Modeling Web Applications Reacting to User Behaviors. *Computer Networks*, 50(10):1533–1546, July 2006.

[60] S. Ceri, F. Daniel, F. M. Facca, and M. Matera. Model-driven Engineering of Active Context-Awareness. *World Wide Web*, 2007. On-line first: `http://dx.doi.org/10.1007/s11280-006-0014-5`.

[61] S. Ceri, F. Daniel, M. Matera, and F. M. Facca. Model-driven Development of Context-Aware Web Applications. *ACM Transactions on Internet Technology*, 7(2), 2007.

[62] H. Chen, T. Finin, and A. Joshi. An Ontology for Context-Aware Pervasive Computing Environments. *The Knowledge Engineering Review*, 18(3):197–207, 2003.

[63] H. L. Chen. *An Intelligent Broker Architecture for Pervasive Context-Aware Systems.* PhD thesis, University of Maryland, Baltimore County, Baltimore, MD, USA, 2004.

[64] P. P.-S. Chen. The Entity-Relationship Model – Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[65] K. Cheverst, N. Davies, K. Mitchell, and A. Friday. Experiences of Developing and Deploying a Context-Aware Tourist Guide: The GUIDE Project. In *Proceedings of International Conference on Mobile Computing and Networking, August 6-11, 2000, Boston, MA, USA*, pages 20–31, 2000.

[66] K. Cheverst, N. Davies, K. Mitchell, A. Friday, and C. Efstratiou. Developing a Context-Aware Electronic Tourist Guide: Some Issues and Experiences. In *Proceedings of ACM SIGCHI Conference on Human Factors in Computing Systems, April 1-6, 2000, The Hague, The Netherlands*, pages 17–24, 2000.

[67] P. Chiu, J. Boreczky, A. Girgensohn, and D. Kimber. LiteMinutes: An Internet-based System for Multimedia Meeting Minutes. In *Proceedings of International World Wide Web Conference, May 1-5, 2001, Hong Kong, Hong Kong*, pages 140–149, 2001.

[68] J. Clark, editor. XML Stylesheet Language Transformations (XSLT). `http://www.w3.org/TR/xslt`, November 1999.

[69] J. Clifford and A. U. Tansel. On an Algebra for Historical Relational Databases: Two Views. In *Proceedings of ACM SIGMOD International Conference on Management of Data, May 28-31, 1985, Austin, TX, USA*, pages 247–265, 1985.

[70] J. Clifford and D. S. Warren. Formal Semantics for Time in Databases. *ACM Transactions on Databases Systems*, 8(2):214–254, 1983.

[71] CODASYL Data Base Task Group. *April 1971 Report.* Association for Computing Machinery (ACM), 1971.

[72] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387, 1970.

[73] M. H. Coen. Design Principles for Intelligent Environments. In *Proceedings of National Conference on Artificial Intelligence, July 2630, 1998, Madison, WI, USA*, pages 547–554, 1998.

[74] R. Conradi and B. Westfechtel. Version Models for Software Configuration Management. *ACM Computing Surveys*, 30(2):232–282, 1998.

[75] J. Coutaz, J. L. Crowley, S. Dobson, and D. Garlan. Context is Key. *Communications of the ACM*, 48(3):49–53, 2005.

[76] R. D. Cronk. Tributaries and Deltas. *BYTE*, 17(1):177–186, 1992.

[77] P. Dadam, V. Y. Lum, and H.-D. Werner. Integration of Time Versions into a Relational Database System. In *Proceedings of International Conference on Very Large Data Bases, August 27-31, 1984, Singapore, Republic of Singapore*, pages 509–522, 1984.

[78] P. De Bra and L. Calvi. AHA! An Open Adaptive Hypermedia Architecture. *The New Review of Hypermedia and Multimedia*, 4 (26):115–139, 1998.

[79] P. De Bra, G.-J. Houben, and H. Wu. AHAM: A Dexter-based Reference Model for Adaptive Hypermedia. In *Proceedings of ACM Conference on Hypertext and Hypermedia, February 21-25, 1999, Darmstadt, Germany*, pages 147–156, 1999.

[80] P. De Bra, A. Aerts, B. Berden, B. de Lange, B. Rousseau, T. Santic, D. Smits, and N. Stash. AHA! The Adaptive Hypermedia Architecture. In *Proceedings of ACM Conference on Hypertext and Hypermedia, August 26-30, 2003, Nottingham, England, UK*, pages 81–84, 2003.

[81] A. de Spindler, M. C. Norrie, M. Grossniklaus, and B. Signer. Spatio-Temporal Proximity as a Basis for Collaborative Filtering in Mobile Environments. In *Proceedings of International Workshop on Ubiquitous Mobile Information and Collaboration Systems, June 5-6, 2006, Luxembourg, Grand Duchy of Luxembourg*, pages 912–926, 2006.

[82] O. De Troyer and S. Casteleyn. Designing Localized Web Sites. In *Proceedings of International Conference on Web Information Systems Engineering, November 22-24, 2004, Brisbane, Australia*, pages 547–558, 2004.

[83] O. De Troyer and C. J. Leune. WSDM: A User-Centered Design Method for Web Sites. *Computer Networks and ISDN Systems*, 30 (1-7):85–94, 1998.

[84] R. De Virgilio and R. Torlone. A General Methodology for Context-Aware Data Access. In *Proceedings of ACM International Workshop on Data Engineering for Wireless and Mobile Access, June 12, 2005, Baltimore, MD, USA*, pages 9–15, 2005.

[85] R. De Virgilio and R. Torlone. Management of Heterogeneous Profiles in Context-Aware Adaptive Information System. In *Proceedings of On the Move to Meaningful Internet Systems Workshops, October 31-November 4, 2005, Agia Napa, Cyprus*, pages 132–141, 2005.

[86] R. De Virgilio and R. Torlone. Modeling Heterogeneous Context Information in Adaptive Web Based Applications. In *Proceedings of the International Conference on Web Engineering, July 11-14, 2006, Palo Alto CA, USA*, pages 56–63, 2006.

[87] R. De Virgilio, R. Torlone, and G.-J. Houben. A Rule-based Approach to Content Delivery Adaptation in Web Information Systems. In *Proceedings of the International Conference on Mobile Data Management, May 9-13, 2006, Nara, Japan*, pages 21–24, 2006.

[88] N. M. Delisle and M. D. Schwartz. Neptune: A Hypertext System for CAD Applications. In *Proceedings of ACM SIGMOD International Conference on Management of Data, May 28-30, 1986, Washington, D.C., USA*, pages 132–143, 1986.

[89] N. M. Delisle and M. D. Schwartz. Contexts – A Partitioning Concept for Hypertext. *ACM Transactions on Information Systems*, 5 (2):168–186, 1987.

[90] A. K. Dey, G. D. Abowd, and D. Salber. A Context-based Infrastructure for Smart Environments. In *Proceedings of International Workshop on Managing Interactions in Smart Environments, December 13-14, 1999, Dublin Ireland*, pages 114–128, 1999.

[91] K. R. Dittrich and R. A. Lorie. Version Support for Engineering Database Systems. *IEEE Transactions on Software Engineering*, 14(4):429–437, 1988.

[92] K. R. Dittrich, W. Gotthard, and P. C. Lockemann. DAMOKLES – A Database System for Software Engineering Environments. In *Proceedings of an International Workshop on Advanced Programming Environments, June 16-18, 1986, Trondheim, Norway*, pages 353–371, 1986.

[93] P. Dourish. *Where the Action Is: The Foundations of Embodied Interaction.* MIT Press, 2001.

[94] D. J. Ecklund, E. F. Ecklund, Jr., R. O. Eifrig, and F. M. Tonge. DVSS: A Distributed Version Storage Server for CAD Applications. In *Proceedings of International Conference on Very Large Data Bases, September 1-4, 1987, Brighton, UK*, pages 443–454, 1987.

[95] R. Elmasri, G. T. J. Wuu, and Y.-J. Kim. The Time Index: An Access Structure for Temporal Data. In *Proceedings of International Conference on Very Large Data Bases, August 13-16, 1990, Brisbane, Queensland, Australia*, pages 1–12, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[96] F. Espinoza, P. Persson, A. Sandin, H. Nyström, E. Cacciatore, and M. Bylund. GeoNotes: Social and Navigational Aspects of Location-Based Information Systems. In *Proceedings of International Conference on Ubiquitous Computing, September 30-October 2, 2001, Atlanta, GA, USA*, pages 2–17, 2001.

[97] J. Estublier. Work Space Management in Software Engineering Environments. In *Proceedings of Workshop on System Configuration Management, March 25-26, 1996, Berlin, Germany*, pages 127–138, 1996.

[98] J. Estublier and R. Casallas. Three Dimensional Versioning. In *Selected papers from the ICSE SCM-4 and SCM-5 Workshops on Software Configuration Management*, pages 118–135, 1995.

[99] F. M. Facca, S. Ceri, J. Armani, and V. Demaldé. Building Reactive Web Applications. In *Special Interest Tracks and Posters of International World Wide Web Conference, May 10-14, 2005, Chiba, Japan*, pages 1058–1059, 2005.

[100] P. Fahy and S. Clarke. CASS – Middleware for Mobile Context-Aware Applications. In *Proceedings of Workshop on Context Awareness, June 6, 2004, Boston, MA, USA*, 2004.

[101] M. Fernández, D. Florescu, J. Kang, A. Levy, and D. Suciu. STRUDEL: A Web Site Management System. In *Proceedings of ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, AZ, USA*, pages 549–552, 1997.

[102] M. Fernández, D. Suciu, and I. Tatarinov. Declarative Specification of Data-intensive Web Sites. In *Proceedings of Conference on*

Domain-Specific Languages, October 3-6, 1999, Austin, TX, USA, pages 135–148, 1999.

[103] M. Fernández, D. Florescu, A. Levy, and D. Suciu. Declarative Specification of Web Sites with STRUDEL. *The VLDB Journal*, 9 (1):38–55, 2000.

[104] Z. Fiala and G.-J. Houben. A Generic Transcoding Tool for Making Web Applications Adaptive. In *Proceedings of CAISE'05 Forum, June 13-17, 2005, Porto, Portugal*, pages 15–20, 2005.

[105] Z. Fiala, M. Hinz, K. Meissner, and F. Wehner. A Component-based Approach for Adaptive, Dynamic Web Documents. *Journal of Web Engineering*, 2(1-2):58–73, 2003.

[106] Z. Fiala, F. Frăsincar, M. Hinz, G.-J. Houben, P. Barna, and K. Meissner. Engineering the Presentation Layer of Adaptable Web Information Systems. In *Proceedings of International Conference on Web Engineering, July 26-30, 2004, Munich, Germany*, pages 459–472, 2004.

[107] Z. Fiala, M. Hinz, G.-J. Houben, and F. Frăsincar. Design and Implementation of Component-based Adaptive Web Presentations. In *Proceedings of Symposium on Applied Computing, March 14-17, 2004, Nicosia, Cyprus*, pages 1698–1704, 2004.

[108] G. W. Fitzmaurice. Situated Information Spaces and Spatially Aware Palmtop Computers. *Communications of the ACM*, 36(7): 39–49, 1993.

[109] A. Fitzpatrick, G. Biegel, S. Clarke, and V. Cahill. Towards a Sentient Object Model. In *Proceedings of Workshop on Engineering Context-Aware Object-Oriented Systems and Environments, November 5, 2002, Seattle, WA, USA*, 2002.

[110] A. Fox and E. A. Brewer. Reducing WWW Latency and Bandwidth Requirements by Real-time Distillation. *Computer Networks and ISDN Systems*, 28(7-11):1445–1456, 1996.

[111] A. Fox, S. D. Gribbe, E. A. Brewer, and E. Amir. Adapting to Network and Client Variability via On-Demand Dynamic Distillation. *Computer Architecture News*, 24(Special Issue):160–170, 1996.

[112] F. Frăsincar. *Hypermedia Presentation Generation for Semantic Web Information Systems.* PhD thesis, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, June 2005.

[113] F. Frăsincar, P. Barna, G.-J. Houben, and Z. Fiala. Adaptation and Reuse in Designing Web Information Systems. In *Proceedings of International Conference on Information Technology: Coding and Computing, April 2-4, 2004, Las Vegas, NV, USA*, pages 387–391, 2004.

[114] F. Frăsincar, G.-J. Houben, and P. Barna. Hera Presentation Generator. In *Special Interest Tracks and Posters of International World Wide Web Conference, May 10-14, 2005, Chiba, Japan*, pages 952–953, 2005.

[115] S. K. Gadia. A Homogeneous Relational Model and Query Languages for Temporal Databases. *ACM Transactions on Database Systems*, 13(14):418–448, December 1988.

[116] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Longman Publishing Co., 1995.

[117] F. Garzotto, P. Paolini, and D. Schwabe. HDM – A Model-Based Approach to Hypertext Application Design. *ACM Transactions on Informations Systems*, 11(1):1–26, 1993.

[118] I. P. Goldstein and D. G. Bobrow. Extending Object Oriented Programming in Smalltalk. In *Proceedings of ACM Conference on LISP and Functional Programming, August 25-27, 1980, Stanford University, CA, USA*, pages 75–81, 1980.

[119] M. Grossniklaus. CMServer – An Object-Oriented Framework for Website Development and Content Management. Diploma thesis, Swiss Federal Institute of Technology Zurich (ETH Zurich), Zurich, Switzerland, 2001.

[120] M. Grossniklaus and M. C. Norrie. Information Concepts for Content Management. In *Proceedings of International Workshop on Data Semantics and Web Information Systems, December 11, 2002, Singapore, Republic of Singapore*, pages 150–159, 2002.

[121] M. Grossniklaus, M. C. Norrie, and P. Büchler. Metatemplate Driven Multi-Channel Presentation. In *Proceedings of International Workshop on Multichannel and Mobile Information Systems, December 13, 2003, Roma, Italy*, pages 234–242, 2003.

[122] M. Grossniklaus, M. C. Norrie, B. Signer, and N. Weibel. Putting Location-Based Services on the Map. In *Proceedings of International Symposium on Web and Wireless Geographical Information Systems, December 4-5, 2006, Hong Kong, China*, pages 1–11, 2006.

[123] T. Gu, H. K. Pung, and D. Q. Zhang. A Middleware for Building Context-Aware Mobile Services. In *Proceedings of Vehicular Technology Conference (Spring), May 17-19, 2004, Milan, Italy*, pages 2656–2660, 2002.

[124] R. V. Guha and J. McCarthy. Varieties of Contexts. In *Proceedings of International and Interdisciplinary Conference on Modeling and Using Context, June 23-25, 2003, Stanford, CA, USA*, pages 164–177, 2003.

[125] B. Gulla, E.-A. Karlsson, and D. Yeh. Change-Oriented Version Descriptions in EPOS. *Software Engineering Journal*, 6(6):378–386, 1991.

[126] R. M. Gustavsen. Condor – An Application Framework for Mobility-Based Context-Aware Applications. In *Proceedings of Workshop on Concepts and Models for Ubiquitous Computing, September 29, 2002, Göteborg, Sweden*, 2002.

[127] A. Haake. CoVer: A Contextual Version Server for Hypertext Applications. In *Proceedings of ACM European Conference on Hypertext Technology, November 30-December 4, 1992, Milan, Italy*, pages 43–52, 1992.

[128] A. Haake. Under CoVer: The Implementation of a Contextual Version Server for Hypertext Applications. In *Proceedings of ACM European Conference on Hypermedia Technology, September 19-23, 1994, Edinburgh, Scotland, UK*, pages 81–93, 1994.

[129] A. Haake and J. M. Haake. Take CoVer: Exploiting Version Support in Cooperative Systems. In *Proceedings of ACM SIGCHI*

Conference on Human Factors in Computing Systems, April 24-29, 1993, Amsterdam, The Netherlands, pages 406–413, 1993.

[130] A. Haake and D. L. Hicks. VerSE: Towards Hypertext Versioning Styles. In *Proceedings of ACM Conference on Hypertext, March 16-20, 1996, Bethesda, MD, USA*, pages 224–234, 1996.

[131] A. N. Habermann and D. Notkin. Gandalf: Software Development Environments. *IEEE Transactions on Software Engineering*, 12 (12):1117–1127, 1986.

[132] F. Halasz and M. Schwartz. The Dexter Hypertext Reference Model. *Communications of the ACM*, 37(2):30–39, 1994.

[133] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The Anatomy of a Context-Aware Application. *Wireless Networks*, 8 (2-3):187–197, 2002.

[134] R. L. Haskin and R. A. Lorie. On Extending the Functions of a Relational Database System. In *Proceedings of ACM SIGMOD International Conference on Management of Data, June 2-4, 1982, Orlando, FL, USA*, pages 207–212, 1982.

[135] C. Heath and P. Luff. Disembodied Conduct: Communication Through Video in a Multi-Media Office Environment. In *Proceedings of ACM SIGCHI Conference on Human Factors in Computing Systems, April 27-May 2, 1991, New Orleans, LA, USA*, pages 99–103, 1991.

[136] S. Helal, B. Winkler, C. Lee, Y. Kaddoura, L. Ran, C. Giraldo, S. Kuchibhotla, and W. Mann. Enabling Location-Aware Pervasive Computing Applications for the Edlerly. In *Proceedings of IEEE International Conference on Pervasive Computing and Communications*, pages 531–536, 2003.

[137] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura, and E. Jansen. The Gator Tech Smart House: A Programmable Pervasive Space. *Computer*, 38(3):50–60, 2005.

[138] G. D. Held, M. R. Stonebraker, and E. Wong. INGRES – A Relational Data Base System. In *Proceedings of National Computer Conference, May 19-22, 1975, Anaheim, CA, USA*, pages 409–416, 1975.

[139] R. Hennicker and N. Koch. A UML-Based Methodology for Hypermedia Design. In *Proceedings of International Conference on the Unified Modeling Language, October 2-6, 2000, York, England, UK*, pages 410–424, 2000.

[140] K. Henricksen. *A Framework for Context-Aware Pervasive Computing Applications*. PhD thesis, University of Queensland, Brisbane, Australia, 2003.

[141] D. L. Hicks, J. J. Leggett, P. J. Nürnberg, and J. L. Schnase. A Hypermedia Version Control Framework. *ACM Transactions on Information Systems*, 16(2):127–160, 1998.

[142] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, and W. Retschitzegger. Context-Awareness on Mobile Devices – The Hydrogen Approach. In *Proceedings of Annual Hawaii International Conference on System Sciences (Track 9)*, pages 10–19, 2003.

[143] F. Hohl, U. Kubach, A. Leonhardi, K. Rothermel, and M. Schwehm. Next Century Challenges: Nexus – An Open Global Infrastructure for Spatial-Aware Applications. In *Proceedings of ACM/IEEE International Conference on Mobile Computing and Networking, August 15-20, 1999, Seattle, WA, USA*, pages 249–255, 1999.

[144] J. I. Hong and J. A. Landay. A Context/Communication Information Agent. *Personal and Ubiquitous Computing*, 5(1):78–81, 2001.

[145] J. I. Hong and J. A. Landay. An Infrastructure Approach to Context-Aware Computing. *Human Computer Interaction*, 16(2-4):287–303, 2001.

[146] G.-J. Houben, P. Barna, F. Frăsincar, and R. Vdovják. Hera: Development of Semantic Web Information Systems. In *Proceedings of International Conference on Web Engineering, July 14-18, 2003, Oviedo, Spain*, pages 529–538, 2003.

[147] S. S. Intille. Designing a Home of the Future. *IEEE Pervasive Computing*, 1(2):76–82, 2002.

[148] G. Jaeschke and H.-J. Schek. Remarks on the Algebra of Non First Normal Form Relations. In *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March 29-31, 1982, Los Angeles, CA, USA*, pages 124–138, 1982.

[149] C. S. Jensen and R. T. Snodgrass. Temporal Data Management. *IEEE Transactions on Knowledge and Data Engineering*, 11(1): 36–44, 1999.

[150] Y. Jin, S. Decker, and G. Wiederhold. OntoWebber: Model-Driven Ontology-Based Web Site Management. In *Proceedings of International Semantic Web Working Symposium, July 29-August 1, 2001, Stanford University, CA, USA*, pages 529–547, 2001.

[151] S. Jones and P. J. Mason. Handling the Time Dimension in a Data Base. In *Proceedings of International Conference on Data Bases, July 2-4, 1980, Aberdeen, Scotland*, pages 65–83, 1980.

[152] R. José and N. Davies. Scalable and Flexible Location-Based Services for Ubiquitous Information Access. In *Proceedings of International Symposium on Handheld and Ubiquitous Computing, September 27-29, 1999, Karlsruhe, Germany*, pages 52–66, 1999.

[153] J. W. Kaltz and J. Ziegler. A Conceptual Model for Context-Aware Web Engineering. In *Proceedings of International Workshop on Modelling and Retrieval of Context, September 20-21, 2004, Ulm, Germany*, 2004.

[154] D. Kaplan. On the Logic of Demonstratives. *Journal of Philosophical Logic*, 8(1):81–98, 1979.

[155] G. Kappel, W. Retschitzegger, and W. Schwinger. Modeling Customizable Web Applications – A Requirement's Perspective. In *Proceedings of International Conference on Digital Libraries, November 13-16, 2000, Kyoto, Japan*, pages 168–179, 2000.

[156] R. H. Katz. Toward a Unified Framework for Version Modeling in Engineering Databases. *ACM Computing Surveys*, 22(4):375–409, 1990.

[157] R. H. Katz and E. E. Chang. Managing Change in a Computer-Aided Design Database. In *Proceedings of International Conference*

on Very Large Data Bases, September 1-4, 1987, Brighton, UK, pages 455–462, 1987.

[158] R. H. Katz and T. J. Lehman. Database Support for Versions and Alternatives of Large Design Files. *IEEE Transactions on Software Engineering*, SE-10(2):191–200, 1984.

[159] R. H. Katz, M. Anwarrudin, and E. E. Chang. A Version Server for Computer-Aided Design Data. In *Proceedings of ACM/IEEE Conference on Design Automation, June 1986, Las Vegas, NV, USA*, pages 27–33, 1986.

[160] C. D. Kidd, R. Orr, G. D. Abowd, C. G. Atkeson, I. A. Essa, B. MacIntyre, E. D. Mynatt, T. Starner, and W. Newstetter. The Aware Home: A Living Laboratory for Ubiquitous Computing Research. In *Proceedings of International Workshop on Cooperative Buildings, Integrating Information, Organization, and Architecture, October 1-2, 1999, Pittsburgh, PA, USA*, pages 191–198, 1999.

[161] T. Kindberg, J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino, B. Serra, and M. Spasojevic. People, Places, Things: Web Presence for the Real World. *Mobile Networks and Applications*, 7(5): 365–376, 2002.

[162] M. Kirchhof and S. Linz. Component-based Development of Web-enabled eHome Services. *Personal Ubiquitous Computing*, 9(5): 323–332, 2005.

[163] C. Kiss, editor. Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 2.0. `http://www.w3.org/TR/CCPP-struct-vocab2/`, April 2007.

[164] P. Klahold, G. Schlageter, R. Unland, and W. Wilkes. A Transaction Model Supporting Complex Applications in Integrated Information Systems. In *Proceedings of ACM SIGMOD International Conference on Management of Data, May 28-31, 1985, Austin, TX, USA*, pages 388–401, 1985.

[165] P. Klahold, G. Schlageter, and W. Wilkes. A General Model for Version Management in Databases. In *Proceedings of International*

Conference on Very Large Data Bases, August 25-28, 1986, Kyoto, Japan, pages 319–327, 1986.

[166] M. R. Klopprogge. Term: An Approach to Include the Time Dimension in the Entity-Relationship Model. In *Proceedings of International Conference on Entity-Relationship Approach, October 12-14, 1981, Washington, DC, USA*, pages 477–508, 1981.

[167] A. Kobler. *The eXtreme Design Approach.* PhD thesis, Swiss Federal Institute of Technology Zurich (ETH Zurich), Zurich, Switzerland, 2001.

[168] A. Kobler and M. C. Norrie. OMS Java: Lessons Learned from Building a Multi-Tier Object Management Framework. In *Proceedings of Workshop on Java and Databases: Persistence Options, November 2, 1999, Denver, CO, USA*, 1999.

[169] A. Kobler, M. C. Norrie, B. Signer, and M. Grossniklaus. OMS Java: Providing Information, Storage and Access Abstractions in an Object-Oriented Framework. In *Proceedings of International Conference on Object Oriented Information Systems, August 27-29, 2001, Calgary, Canada*, pages 25–34, 2001.

[170] A. Kobsa, D. Müller, and A. Nill. KN-AHS: An Adaptive Hypertext Client of the User Modeling System BGP-MS. In *Readings in Intelligent User Interfaces*, pages 372–380. Morgan Kaufmann, 1998.

[171] N. Koch. *Software Engineering for Adaptive Hypermedia System.* PhD thesis, Ludwig-Maximilians-University Munich, Munich, Germany, 2000.

[172] N. Koch and M. Wirsing. The Munich Reference Model for Adaptive Hypermedia Applications. In *Proceedings of International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, May 29-31, Malaga, Spain*, pages 213–222, 2002.

[173] P. Korpipää, J. Mäntyjärvi, J. Kela, H. Keränen, and E.-J. Malm. Managing Context Information in Mobile Devices. *IEEE Pervasive Computing*, 2(3):42–51, 2003.

[174] G. Kortuem, M. Bauer, and Z. Segall. NETMAN: The Design of a Collaborative Wearable Computer System. *Mobile Networks and Applications*, 4(1):49–58, 1999.

[175] B. Kreller, D. Carrega, J. Shankar, P. Salmon, S. Bottger, and T. Kassing. A Mobile-Aware City Guide Application. In *Proceedings of ACTS Mobile Communications Summit, June 8-11, 1998, Rhodes, Greece*, pages 60–65, 1998.

[176] M. Lacroix and P. Lavency. Preferences: Putting More Knowledge into Queries. In *Proceedings of International Conference on Very Large Data Bases, September 1-4, 1987, Brighton, England, UK*, pages 217–225, 1987.

[177] C. Lamb, G. Landis, J. Orenstein, and D. Weinreb. The ObjectStore Database System. *Communications of the ACM*, 34(10):50–63, 1991.

[178] A. Lampen and A. Mahler. An Object Base for Attributed Software Objects. In *Proceedings of European Unix Systems User Group Conference, October 3-7, 1988, Cascais, Portugal*, pages 95–106, 1988.

[179] P. Lavency and M. Vanhoedenaghe. Knowledge Based Configuration Management. In *Proceedings of Annual Hawaii International Conference on System Sciences (Software Track), January, Kailua-Kona, HI, United States*, pages 83–92, 1988.

[180] D. B. Leblang and R. P. Chase, Jr. Computer-Aided Software Engineering in a Distributed Workstation Environment. *ACM SIGSOFT Software Engineering Notes*, 9(3):104–112, 1984.

[181] A. Leonhardi, U. Kubach, K. Rothermel, and A. Fritz. Virtual Information Towers – A Metaphor for Intuitive, Location-Aware Information Access in a Mobile Environment. In *Proceedings of IEEE International Symposium on Wearable Computers, October 18-19, 1999, San Francisco, CA, USA*, pages 15–20, 1999.

[182] A. Lombardoni. *Towards a Universal Information Platform: An Object-Oriented, Multi-User, Information Store.* PhD thesis, Swiss Federal Institute of Technology Zurich (ETH Zurich), Zurich, Switzerland, 2006.

[183] N. A. Lorentzos and R. G. Johnson. TRA: A Model for Temporal Relational Algebra. In *Temporal Aspects of Information Systems (Proceedings of IFIP Working Conference on Temporal Aspects in Information Systems, May 13-15, Sophia-Antipolis, France)*, pages 95–108, 1988.

[184] V. Y. Lum, P. Dadam, R. Erbe, J. Günauer, P. Pistor, G. Walch, H.-D. Werner, and J. Woodfill. Designing DBMS Support for the Temporal Dimension. In *Proceedings of ACM SIGMOD International Conference on Management of Data, June 18-21, 1984, Boston, MA, USA*, pages 115–130, 1984.

[185] A. Mahler. Variants: Keeping Things Together and Telling Them Apart. In *Configuration Management*, pages 73–97, 1995.

[186] N. Marmasse and C. Schmandt. Location-Aware Information Delivery with ComMotion. In *Proceedings of International Symposium on Handheld and Ubiquitous Computing, September 25-27, 2000, Bristol, England, UK*, pages 157–171, 2000.

[187] M. Marx and C. Schmandt. CLUES: Dynamic Personalized Message Filtering. In *Proceedings of ACM Conference on Computer Supported Cooperative Work, November 16-20, 1996, Boston, MA, USA*, pages 113–121, 1996.

[188] S. Meyer and A. Rakotonirainy. A Survey of Research on Context-Aware Homes. In *Proceedings of Australasian Information Security Workshop Conference on ACSW Frontiers, February 1, 2003, Adelaide, Australia*, pages 159–168, 2003.

[189] D. B. Miller, R. G. Stockton, and C. W. Krueger. An Inverted Approach to Configuration Management. *ACM SIGSOFT Software Engineering Notes*, 14(7):1–4, 1989.

[190] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller. Introduction to WordNet: An On-line Lexical Database. *International Journal of Lexicography*, 3(4):235–244, 1990.

[191] M. Milosavljevic and J. Oberlander. Dynamic Hypertext Catalogues: Helping Users to Help Themselves. In *Proceedings of ACM Conference on Hypertext and Hypermedia, June 20-24, 1998, Pittsburgh, PA, United States*, pages 123–131, 1998.

[192] R. Mohan, J. R. Smith, and C.-S. Li. Adapting Multimedia Internet Content for Universal Access. *IEEE Transactions on Multimedia*, 1(1):104–114, 1999.

[193] R. Motschnig-Pitrik and L. Nykl. The Role and Modeling of Context in a Cognitive Model of Rogers' Person-Centred Approach. In *Proceedings of International and Interdisciplinary Conference on Modeling and Using Context, July 27-30, 2001, Dundee, Scotland, UK*, pages 275–289, 2001.

[194] M. C. Mozer. The Neural Network House: An Environment that Adapts to its Inhabitants. In *Proceedings of AAAI Spring Symposium on Intelligent Environments, March 23-25, 1998, Stanford University, Palo Alto, CA, USA*, pages 110–114, 1998.

[195] B. P. Munch, J.-O. Larsen, B. Gulla, R. Conradi, and E.-A. Karlsson. Uniform Versioning: The Change-Oriented Model. In *Proceedings of International Workshop of Software Configuration Management, May 21-22, 1993, Baltimore, MD, USA*, pages 188–196, 1993.

[196] E. D. Mynatt, I. Essa, and W. Rogers. Increasing the Opportunities for Aging in Place. In *Proceedings of Conference on Universal Usability, November 16-17, 2000, Arlington, VA, USA*, pages 65–71, 2000.

[197] K. S. Nagel, C. D. Kidd, T. O'Connell, A. K. Dey, and G. D. Abowd. The Family Intercom: Developing a Context-Aware Audio Communication System. In *Proceedings of International Conference on Ubiquitous Computing, September 30-October 2, 2001, Atlanta, GA, USA*, pages 176–183, 2001.

[198] S. B. Navathe and R. Ahmed. A Temproal Relational Model and a Query Language. *Information Sciences*, 49(1–3):147–175, 1989.

[199] P. J. Nicklin. Managing Multi-Variant Software Configuration. In *Proceedings of International Workshop on Software Configuration Management, June 12-14, 1991, Trondheim, Norway*, pages 53–57, 1991.

[200] M. C. Norrie. An Extended Entity-Relationship Approach to Data Management in Object-Oriented Systems. In *Proceedings of Inter-*

national Conference on the Entity-Relationship Approach, Arlington, TX, USA, pages 390–401, 1994.

[201] M. C. Norrie. Distinguishing Typing and Classification in Object Data Models. In H. Kangassalo, H. Jaakkola, S. Ohsuga, and B. Wangler, editors, *Information Modelling and Knowledge Bases VI*, pages 399–412. IOS Press, 1995.

[202] M. C. Norrie and A. Palinginis. From State to Structure: An XML Web Publishing Framework. In *Proceedings of CAiSE'03 Forum, June 16-20, 2003, Klagenfurt/Velden, Austria*, pages 137–140, 2003.

[203] M. C. Norrie and A. Palinginis. Empowering Databases for Context-Dependent Information Delivery. In *Proceedings of International Workshop on Ubiquitous Mobile Information and Collaboration Systems, June 16-17, 2003, Klagenfurt/Velden, Austria*, pages 90–101, 2003.

[204] M. C. Norrie and A. Palinginis. Versions for Context Dependent Information Services. In *Proceedings of Conference on Cooperative Information Systems, November 3-7, 2003, Catania-Sicily, Italy*, pages 503–515, 2003.

[205] M. C. Norrie, B. Signer, and N. Weibel. General Framework for the Rapid Development of Interactive Paper Applications. In *Proceedings of Workshop on Collaborating over Paper and Digital Documents, November 4, 2006, Banff, Canada*, pages 9–12, 2005.

[206] M. C. Norrie, B. Signer, and N. Weibel. Print-n-Link: Weaving the Paper Web. In *Proceedings of the ACM Symposium on Document Engineering, October 10-13, 2006, Amsterdam, The Netherlands*, pages 34–43, 2006.

[207] M. C. Norrie, B. Signer, M. Grossniklaus, R. Belotti, C. Decurtins, and N. Weibel. Context-Aware Platform for Mobile Data Management. *Wireless Networks*, 2007. On-line first: `http://dx.doi.org/10.1007/s11276-006-9858-y`.

[208] Objectivity, Inc. Objectivity/DB. `http://www.objectivity.com`, 2000.

[209] Open Text Corporation. Livelink Enterprise Content Management. `http://www.opentext.com`, 2006.

[210] R. Oppermann and M. Specht. A Context-Sensitive Nomadic Exhibition Guide. In *Proceedings of International Symposium on Handheld and Ubiquitous Computing, September 25-27, 2000, Bristol, England, UK*, pages 127–142, 2000.

[211] K. Østerbye. Structural and Cognitive Problems in Providing Version Control for Hypertext. In *Proceedings of ACM European Conference on Hypertext Technology, November 30-December 4, 1992, Milan, Italy*, pages 33–42, 1992.

[212] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object Exchange Across Heterogeneous Information Sources. In *Proceedings of International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan*, pages 251–260, 1995.

[213] J. Pascoe. The stick-e Note Architecture: Extending the Interface Beyond the User. In *Proceedings of International Conference on Intelligent User Interfaces, January 6-9 1997, Orlando, FL, USA*, pages 261–264, 1997.

[214] J. Pascoe. Adding Generic Contextual Capabilities to Wearable Computers. In *Proceedings of IEEE International Symposium on Wearable Computers, October 19-20, 1998, Pittsburgh, PA, USA*, pages 92–99, 1998.

[215] J. Pascoe, D. R. Morse, and N. Ryan. Developing Personal Technology for the Field. *Personal and Ubiquitous Computing*, 2(1): 28–36, 1998.

[216] J. Pascoe, N. Ryan, and D. Morse. Issues in Developing Context-Aware Computing. In *Proceedings of International Symposium on Handheld and Ubiquitous Computing, September 27-29, 1999, Karlsruhe, Germany*, pages 208–221, 1999.

[217] P. F. Patel-Schneider, P. Hayesand, and I. Horrocks, editors. OWL Web Ontology Language: Semantics and Abstract Syntax. `http://www.w3.org/TR/owl-absyn/`, February 2004.

[218] P. Persson, F. Espinoza, P. Fagerberg, A. Sandin, and R. Cöster. GeoNotes: A Location-based Information System for Public

Spaces. In K. Höök, D. Benyon, and A. Munro, editors, *Designing Information Spaces: The Social Navigation Approach*, pages 151–173. Springer Verlag, 2003.

[219] M. Petrovic, M. Grossniklaus, and M. C. Norrie. Role-Based Modelling of Interactions in Database Applications. In *Proceedings on International Conference on Advanced Information Systems Engineering, June 5-9, 2006, Luxembourg, Grand Duchy of Luxembourg*, pages 63–77, 2006.

[220] N. Pissinou, K. Makki, and Y. Yesha. On Temporal Modeling in the Context of Object Databases. *ACM SIGMOD Record*, 22(3): 8–15, 1993.

[221] J. Plaice and W. W. Wadge. A New Approach to Version Control. *IEEE Transactions on Software Engineering*, 19(3):268–276, 1993.

[222] A. Ranganathan and R. H. Campbell. An Infrastructure for Context-Awareness Based on First Order Logic. *Personal Ubiquitous Computing*, 7(6):353–364, 2003.

[223] A. Ranganathan, R. H. Campbell, A. Ravi, and A. Mahajan. ConChat: A Context-Aware Chat Program. *IEEE Pervasive Computing*, 1(3):51–57, 2002.

[224] T. Reichenbacher. The World in Your Pocket – Towards a Mobile Cartography. In *Proceedings of International Cartography Conference, August 6-10, 2001, Bejing, China*, volume 4, pages 2514–2521, 2001.

[225] T. Reichenbacher. Adaptive Concepts for a Mobile Cartography. *Journal of Geographical Sciences*, 11(Supplement):43–53, 2001.

[226] C. Reichenberger. VOODOO – A Tool for Orthogonal Version Management. In *Selected Papers from the ICSE SCM-4 and SCM-5 Workshops on Software Configuration Management*, pages 61–79, 1995.

[227] F. Rice. Introducing the Office (2007) Open XML File Formats. `http://msdn.microsoft.com/en-us/library/ms406049.aspx`, May 2006.

[228] G. Rivera. *From File Pathnames to File Objects.* PhD thesis, Swiss Federal Institute of Technology Zurich (ETH Zurich), Zurich, Switzerland, 2001.

[229] J. F. Roddick and J. D. Patrick. Temporal Semantics in Information Systems – A Survey. *Information Systems*, 17(3):249–267, 1992.

[230] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. A Middleware Infrastructure for Active Spaces. *IEEE Pervasive Computing*, 1(4):74–83, 2002.

[231] G. Rossi, D. Schwabe, and R. Guimarães. Designing Personalized Web Applications. In *Proceedings of International World Wide Web Conference, May 1-5, 2001, Hong Kong, China*, pages 275–284, 2001.

[232] D. Rotem and A. Segev. Physical Organization of Temporal Data. In *Proceedings of International Conference on Data Engineering, February 3-5, 1987, Los Angeles, CA, USA*, pages 547–553, 1987.

[233] J. R. Rumbaugh, M. R. Blaha, W. Lorensen, F. Eddy, and W. Premerlani. *Object-Oriented Modeling and Design.* Prentice Hall, 1990.

[234] J. R. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual.* Addison-Wesley Longman, 1999.

[235] N. Ryan, J. Pascoe, and D. R. Morse. Enhanced Reality Fieldwork: The Context-Aware Archaeological Assistant. In *Proceedings of Conference on Computer Applications and Quantitative Methods in Archaeology, April, 1997 Birmingham, England, UK*, 1997.

[236] N. Ryan, J. Pascoe, and D. R. Morse. FieldNote: A Handheld Information System for the Field. In *Proceedings of International Workshop on Telegeoprocessing, May 6-7, 1999, Lyon, France*, pages 156–163, 1999.

[237] D. Salber, A. K. Dey, and G. D. Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In *Proceedings of ACM SIGCHI Conference on Human Factors in Computing Systems, May 15-20, 1999, Pittsburgh, PA, USA*, pages 434–441, 1999.

[238] N. Sarnak, R. L. Bernstein, and V. Kruskal. Creation and Maintenance of Multiple Versions. In *Proceedings of International Workshop on Software and Configuration Control, January 27-29, 1988, Grassau, Germany*, pages 264–275, 1988.

[239] N. Sawhney and C. Schmandt. Nomadic Radio: Speech and Audio Interaction for Contextual Messaging in Nomadic Environments. *ACM Transactions on Computer-Human Interaction*, 7(3): 353–383, 2000.

[240] H.-J. Schek and P. Pistor. Data Structures for an Integrated Data Base Management and Information Retrieval System. In *Proceedings of the International Conference on Very Large Data Bases, September 8-10, 1982, Mexico City, Mexico*, pages 197–207, 1982.

[241] K.-D. Schewe and B. Thalheim. Modeling Interaction and Media Objects. In *Proceedings of International Conference on Applications of Natural Language to Information Systems, June 28-29, 2001, Madrid, Spain – Revised Papers*, pages 313–324, 2001.

[242] K.-D. Schewe and B. Thalheim. Reasoning About Web Information Systems Using Story Algebras. In *Proceedings of East-European Conference on Advances in Databases and Information Systems, September 22-25, 2004, Budapest, Hungary*, pages 54–66, 2004.

[243] B. N. Schilit and M. M. Theimer. Disseminating Active Map Information to Mobile Hosts. *IEEE Network*, 8(5):22–32, 1994.

[244] B. N. Schilit, N. Adams, and R. Want. Context-Aware Computing Applications. In *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications, December 8-9, 1994, Santa Cruz, CA, USA*, pages 85–90, 1994.

[245] B. N. Schilit, D. M. Hilbert, and J. Trevor. Context-Aware Communication. *IEEE Wireless Communications*, 9(5):46–54, 2002.

[246] C. Schmandt, N. Marmasse, S. Marti, N. Sawhney, and S. Wheeler. Everywhere Messaging. *IBM Systems Journal*, 39(3-4):660–677, 2000.

[247] A. Schmidt. Implicit Human Computer Interaction Through Context. *Personal and Ubiquitous Computing*, 4(2-3):191–199, 2000.

[248] A. Schmidt, M. Beigl, and H.-W. Gellersen. There Is More to Context Than Location. *Computer and Graphics*, 23(6):893–901, 1999.

[249] A. Schmidt, A. Takaluoma, and J. Mäntyjärvi. Context-Aware Telephony Over WAP. *Personal and Ubiquitous Computing*, 4(4): 225–229, 2000.

[250] D. Schwabe and G. Rossi. An Object Oriented Approach to Web-based Applications Design. *Theory and Practice of Object Systems*, 4(4):207–225, 1998.

[251] D. Schwabe, G. Rossi, and S. D. J. Barbosa. Systematic Hypermedia Application Design with OOHDM. In *Proceedings of ACM Conference on Hypertext, March 16-20, 1996, Washington, DC, USA*, pages 116–128, 1996.

[252] C. Schwank. A Content Management Component for EdFest. Semester thesis, Swiss Federal Institute of Technology Zurich (ETH Zurich), Zurich, Switzerland, 2005.

[253] B. Schwarzentrub. Multi-Variant Programming. Semester thesis, Swiss Federal Institute of Technology Zurich (ETH Zurich), Zurich, Switzerland, 2006.

[254] E. Sciore. Using Annotations to Support Multiple Kinds of Versioning in an Object-Oriented Database System. *ACM Transactions on Database Systems*, 16(3):417–438, 1991.

[255] E. Sciore. Versioning and Configuration Management in an Object-Oriented Data Model. *The VLDB Journal*, 3(1):77–106, 1994.

[256] A. Segev and A. Shoshani. Logical Modeling of Temporal Data. In *Proceedings of ACM SIGMOD International Conference on Management of Data, May 27-29, 1987, San Francisco, CA, USA*, pages 454–466, 1987.

[257] Y. Shi, W. Xie, G. Xu, R. Shi, E. Chen, Y. Mao, and F. Liu. The Smart Classroom: Merging Technologies for Seamless Tele-Education. *IEEE Pervasive Computing*, 2(2):47–55, 2003.

[258] F. Siegemund. A Context-Aware Communication Platform for Smart Objects. In *Proceedings of International Conference on Pervasive Computing, April 18-23, 2004, Linz/Vienna, Austria*, pages 69–86, 2004.

[259] B. Signer. *Fundamental Concepts for Interactive Paper and Cross-Media Information Spaces.* PhD thesis, Swiss Federal Institute of Technology Zurich (ETH Zurich), Zurich, Switzerland, 2006.

[260] B. Signer, M. Grossniklaus, and M. C. Norrie. Java Framework for Database-Centric Web Engineering. In *Proceedings of Workshop on Web Engineering, May 1, 2001, Hong Kong, China*, pages 42–49, 2001.

[261] B. Signer, M. C. Norrie, M. Grossniklaus, R. Belotti, C. Decurtins, and N. Weibel. Paper-Based Mobile Access to Databases. In *Demonstration Proceedings of ACM SIGMOD International Conference on Management of Data, June 27-29, Chicago, IL, USA*, pages 763–765, 2006.

[262] A. Singh, A. Trivedi, K. Ramamritham, and P. Shenoy. PTC: Proxies that Transcode and Cache in Heterogeneous Web Client Environments. *World Wide Web*, 7(1):7–28, 2004.

[263] J. R. Smith, R. Mohan, and C.-S. Li. Transcoding Internet Content for Heterogeneous Client Devices. In *Proceedings of International Symposium on Circuits and Systems, May 31-June 3, 1998, Monterey, CA, USA*, pages 599–602, 1998.

[264] R. Snodgrass. The Temporal Query Language TQuel. In *Proceedings of ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, April 2-4, 1984, Waterloo, Ontario, Canada*, pages 204–213, 1984.

[265] R. Snodgrass and I. Ahn. A Taxonomy of Time Databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data, May 28-31, 1985, Austin, TX, USA*, pages 236–246, 1985.

[266] L. F. G. Soares, G. L. D. S. Filho, R. F. Rodrigues, and D. C. Muchaluat. Versioning Support in the HyperProp System. *Multimedia Tools and Applications*, 8(3):325–339, 1999.

[267] Y. Stavrakas and M. Gergatsoulis. Multidimensional Semistructured Data: Representing Context-Dependent Information on the Web. In *Proceedings of International Conference on Advanced Information Systems Engineering, May 27-31, 2002, Toronto, Canada*, pages 183–199, 2002.

[268] Y. Stavrakas, M. Gergatsoulis, and P. Rondogiannis. Multidimensional XML. In *Proceedings of International Workshop on Distributed Communities on the Web, June 19-21, 2000, Quebec City, Canada*, pages 100–109, 2000.

[269] Y. Stavrakas, K. Pristouris, A. Efandis, and T. Sellis. Implementing a Query Language for Context-Dependent Semistructured Data. In *Proceedings of East-European Conference on Advances in Databases and Information Systems, September 22-25, 2004, Budapest, Hungary*, pages 173–188, 2004.

[270] A. Steiner. *A Generalisation Approach to Temporal Data Models and their Implementations*. PhD thesis, Swiss Federal Institute of Technology Zurich (ETH Zurich), Zurich, Switzerland, 1998.

[271] G. Stevenson, P. Nixon, and S. Dobson. Towards a Reliable, Wide-Area Infrastructure for Context-Based Self-Management of Communications. In *Proceedings of IFIP Workshop on Autonomic Communications, October 3-5, 2005, Athens, Greece*, pages 115–128, 2005.

[272] T. Strang and C. Linnhoff-Popien. A Context-Modeling Survey. In *Proceedings of Workshop on Advanced Context Modelling, Reasoning and Management, September 7, 2004, Nottingham, England, UK*, 2004.

[273] Y. Sumi, T. Etani, S. Fels, N. Simonet, K. Kobayashi, and K. Mase. C-MAP: Building a Context-Aware Mobile Assistant for Exhibition Tours. In *Community Computing and Support Systems, Social Interaction in Networked Communities (based on the Kyoto Meeting on Social Interaction and Communityware, June 8-10, 1998, Kyoto, Japan)*, pages 137–154, 1998.

[274] M.-R. Tazari, M. Grimm, and M. Finke. Modelling User Context. In *Proceedings of International Conference on Human-Computer Interaction, June 22-27, 2003, Crete, Greece*, pages 293–297, 2003.

[275] B. Thalheim and A. Düsterhöft. SiteLang: Conceptual Modeling of Internet Sites. In *Proceedings of International Conference on Conceptual Modeling, November 27-30, 2001, Yokohama, Japan*, pages 179–192, 2001.

[276] R. H. Thomason. Type Theoretic Foundations for Context, Part 1: Contexts as Complex Type-Theoretic Objects. In *Proceedings of International and Interdisciplinary Conference on Modeling and Using Context, September 9-11, 1999, Trento, Italy*, pages 351–360, 1999.

[277] V. J. Tsotras and B. Gopinath. Optimal Versioning of Objects. In *Proceedings of International Conference on Data Engineering, February 3-7, 1992, Tempe, AZ, USA*, pages 358–365, 1992.

[278] TYPO3 Association. TYPO3. `http://www.typo3.org`, 2005.

[279] R. Vdovják. *A Model-driven Approach for Building Distributed Ontology-based Web Applications*. PhD thesis, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, June 2005.

[280] R. Vdovják, F. Frăsincar, G.-J. Houben, and P. Barna. Engineering Semantic Web Information Systems in Hera. *Journal of Web Engineering*, 1(1-2):3–26, 2003.

[281] Versant, Corp. Versant Object Database. `http://www.versant.com`, 2004.

[282] Vignette Corporation. Vignette Enterprise Content Management. `http://www.vignette.com`, 1996.

[283] W. W. Wadge, G. Brown, M. C. Schraefel, and T. Yildirim. Intensional HTML. In *Proceedings of International Workshop on Principles of Digital Document Processing, March 29-30, 1998, Saint Malo, France*, pages 128–139, 1998.

[284] R. Want, A. Hopper, V. Falcão, and J. Gibbons. The Active Badge Location System. *ACM Transaction on Information Systems*, 10 (1):91–102, 1992.

[285] J. Warmer and A. Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley Longman, 1999.

[286] M. J. Weal, D. T. Michaelides, M. K. Thompson, and D. C. DeRoure. The Ambient Wood Journals: Replaying the Experience. In *Proceedings of ACM Conference on Hypertext and Hypermedia, August 26-30, 2003, Nottingham, England, UK*, pages 20–27, 2003.

[287] J. N. Weatherall and A. Hopper. Predator: A Distributed Location Service and Example Applications. In *Proceedings of International Workshop on Cooperative Buildings, Integrating Information, Organization, and Architecture, October 1-2, 1999, Pittsburgh, PA, USA*, pages 127–139, 1999.

[288] Web Models s.r.l. WebRatio Site Development Studio. `http://www.webratio.com`, 2001.

[289] M. Weiser. The Computer for the Twenty-First Century. *Scientific American*, 265(3):94–104, 1991.

[290] A. Wexelblat. Communities Through Time: Using History for Social Navigation. In *Community Computing and Support Systems, Social Interaction in Networked Communities (based on the Kyoto Meeting on Social Interaction and Communityware, June 8-10, 1998, Kyoto, Japan)*, pages 281–298, 1998.

[291] E. J. Whitehead, Jr. Goals for a Configuration Management Network Protocol. In *Proceedings of International Symposium on System Configuration Management, September 5-7, 1999, Toulouse, France*, pages 186–203, 1999.

[292] E. J. Whitehead, Jr. *An Analysis of the Hypertext Versioning Domain.* PhD thesis, University of California, Irvine, CA, USA, 2000.

[293] D. Widdows. A Mathematical Model for Context and Word-Meaning. In *Proceedings of International and Interdisciplinary Conference on Modeling and Using Context, June 23-25, 2003, Stanford, CA, USA*, pages 369–382, 2003.

[294] J. F. H. Winkler. Version Control in Families of Large Programs. In *Proceedings of International Conference on Software Engineering, March 30-April 2, 1987, Monterey, CA, USA*, pages 150–161, 1987.

[295] T. Winogard. Architectures for Context. *Human Computer Interaction*, 16(2-4):401–419, 2001.

[296] A. P. Würgler. *OMS Development Framework: Rapid Prototyping for Object-Oriented Databases*. PhD thesis, Swiss Federal Institute of Technology Zurich (ETH Zurich), Zurich, Switzerland, 2000.

[297] A. Zeller and G. Snelting. Unified Versioning Through Feature Logic. *ACM Transactions on Software Engineering and Methodology*, 6(4):398–441, 1997.

[298] Zope Corporation. Zope Content Management Framework. `http://www.zope.org`, 2006.

[299] S. Zweifel. Graphical Authoring Tools for Web Content Management. Diploma thesis, Swiss Federal Institute of Technology Zurich (ETH Zurich), Zurich, Switzerland, 2002.