Schemas for Context-aware Content Storage

Hans-Werner Sehring
Namics
Hamburg, Germany
e-mail: hans-werner.sehring@namics.com

Abstract—**Data, to an increasing degree, is not used directly as content represented in documents, but it serves as a foundation for content tailored for and delivered to users working in different and varying contexts. To this end, the actual content is dynamically assembled from base data with respect to a certain context. This is particularly true for content management applications, e.g., for websites that are targeted at a user's context. The notion of context comprises various dimensions of parameters like language, location, time, user, and user's device. Most data modeling languages, including programming languages, are not well prepared to cope with variants of content, though. They are designed to manage universal, consistent, and complete sets of data. The Minimalistic Meta Modeling Language (M3L) as a language for content representation has proven particularly useful for modeling content in context. Towards an operational M3L execution environment, we are researching data schemas to efficiently store and utilize M3L models. Such schemas serve as a testbed to discuss context-aware data representation and retrieval in this paper. This is done by expressing context-aware models, in particular M3L statements, by means of traditional persistence technology.**

Keywords-data modeling; content modeling; context-aware data modeling; content; content management; context.

I. INTRODUCTION

In the digital society [1], data is required to represent all kinds of *content*, ranging from structured content of text documents to unstructured, typically binary representations of video and audio content. It is used for many purposes, the most obvious ones being information and commerce. Content is published by means of documents, often multimedia documents incorporating different media that are interrelated to form hypermedia networks. So-called publication channels offer the medium for one kind of publication, e.g., a website, a document file, or a mobile app. Content is typically represented in a channel-agnostic way in order to support multi- or even omni-channel publishing.

It is quite common to deliver content to users in a way that addresses the *context* in which they are when requesting the content. This may include the channel they are using, the working mode they are in, the history of previous usage scenarios, etc. Targeting content to users' contexts can range from simply arranging content in a specific way, over specifically assembled documents, to content that is synthesized for the current requests. Examples are a prominent display of teasers for content that is assumed to be of interest to the user, the production of documents matching a user's native language, adjustment of document quality based on the current network bandwidth and the receiving device, and creating content that represents some base data in knowledgeable form.

For such content targeting scenarios, data needs to be stored in a way that allows generating different views on the content, mainly by selecting content relevant in a certain context. Data representing all forms of content in such a system, therefore, needs to be attributed with the contexts in which it is applicable or preferred. Obviously, some notion of context is required for such representations [2].

Data modeling and programming languages typically do not exhibit features to represent context and to include it in evaluations. Database management systems, being the backbone of practically every information system, are particularly optimized for one connected set of data that is supposed to be consistent and complete. This means that they are not well equipped for dynamic content production, neither regarding content representation nor efficient context-dependent retrieval.

Data retrieval needs particular attention when content is dynamically assembled depending on some context in which it is requested. For the tasks of context-aware content management, complex collections of data to be used as content are requested frequently. A context-aware schema has to efficiently support the underlying queries that are employed to identify relevant content.

For the discussion of data models, we consider content in contexts as it is expressible using the Minimalistic Meta Modeling Language (M3L). This language allows expressing content in a straightforward way. Being a modeling language, there is no obvious mapping to established data structures, though.

The rest of this paper is organized as follows. Section II reviews related work in the area of context-aware data and content models. Section III gives a brief overview over the M3L and describes those parts of the language that are required for the discussion in this paper. Section IV presents a first conceptual model of an internal representation of M3L concepts. Section V makes this model more concrete by means of logical representations, comparable to the logical view on databases. Aspects of alternative implementations are touched in Section VI. The conclusion and acknowledgment close the paper.

 ISBN: 978-1-61208-611-8

II. Related Work

Context is important in the area of content management, but also other modeling domains. This section names some existing modeling approaches for contextual information.

A. Content Management Products

Most commercial content management products have introduced some notion of context in their models and processes. They utilize context information to *target* content to users. Some use the term *personalization*, which is similar to, but different from contextualization [3].

In most cases, there are publication *rules* associated with content, similar as discussed in [4]. These rules are based on so-called *segments*. Every user is assigned one or more segments. When requesting content, the rules are evaluated for the actual segment(s) in order to select suitable content.

Content authors and editors maintain the content rules. Segments are assigned to users automatically by the systems based on the users' behavior (user interactions), the user journey (e.g., previously visited sites and search terms used for finding the current website), and context information (e.g., device used and location of the user).

Segments offer a rather universal notion of context, though there is no explicit context model.

B. Context-aware Data Models

Parallel to the notion of context used for content, there exists some work on the influence of environments on running applications. In mobile usage scenarios, context refers mainly to such environmental considerations, e.g., network availability, network bandwidth, device, or location.

Context changes are incorporated dynamically into evaluations in these scenarios [5].

Context-awareness is not limited to data models. It is also used for adaptable or adaptive software systems, e.g., to map software configurations to execution environments [6], or to control the behavior of a generic solution [7].

C. Concept-oriented Content Management

Concept-oriented Content Management (CCM) [8] is an approach to manage content reflecting knowledge. Such content does not represent simple facts, but instead is subject to interpretation. Furthermore, the history of things is described by content, not just their latest state.

CCM is not directly concerned about modeling context. Instead, it aims to introduce a form of pragmatics into content modeling that allows users on the one hand to express differing views by means of individual content models, and on the other hand to still communicate by exchanging content between individualized models.

CCM uses a notion of personalization that goes far beyond the one of content management systems (see above).

It is similar to contextualized content usage, although the system does not know about the context of a user. Instead, users carry out personalization (in CCM terms) manually.

A CCM system reacts to model changes and relates model variants to each other. The basis for this is systems generation: based on the definitions of users, schemas, APIs, and software modules are generated.

Some aspects of the considerations presented in Section VI were gained from the research on the generation of CCM modules for persistence.

III. The Minimalistic Meta Modeling Language

The *Minimalistic Meta Modeling Language* (*M3L*, pronounced "mel") is a modeling language that is applicable to a range of modeling tasks. It proved particularly useful for context-aware content modeling [9].

For the purpose of this paper, we only introduce the static aspects of the M3L in this section. Dynamic evaluations that are defined by means of different rules are not presented here because – at least in the current state of investigation – they lay outside the scope of content models.

The descriptive power of M3L lies in the fact that the formal semantics is rather abstract. There is no fixed domain semantics connected to M3L definitions. There is also no formal distinction between typical conceptual relationships (specialization, instantiation, entity-attribute, aggregation, materialization, contextualization, etc.).

A. Concept Definitions and References

A M3L definition consists of a series of definitions or references. Each definition starts with a previously unused identifier that is introduced by the definition and may end with a semicolon, e.g.:

```
Person;
```

A reference has the same syntax, but it names an identifier that has already been introduced.

We call the entity named by such an identifier a *concept*.

The keyword `is` introduces an optional reference to a *base concept*, making the newly defined concept a *refinement* of it.

A specialization relationship as known from object-oriented modeling is established between the base concept and the newly defined derived concept. This relationship leads to the concepts defined in the context (see below) of the base concept to be visible in the derived concept.

The keyword `is` always has to be followed by either `a`, `an`, or `the`. The keywords `a` and `an` are synonyms for indicating that a classification allows multiple sub concepts of the base concept:

```
Peter is a Person; John is a Person;
```

There may be more than one base concept. Base concepts can be enumerated in a comma-separated list:

```
PeterTheEmployee is a Person, an Employee;
```

The keyword `the` indicates a closed refinement: there may be only one refinement of the base concept (the currently defined one), e.g.:

```
Peter is the FatherOfJohn;
```

Any further refinement of the base concept(s) leads to the redefinition ("unbinding") of the existing refinements.

Statements about already existing concepts lead to their redefinition. For example, the following expressions define the concept `Peter` in a way equivalent to the above variant:

```
Peter is a Person; Peter is an Employee;
```

B. Content and Context Definitions

Concept definitions as introduced in the preceding section are valid in a context. Definitions like the ones seen

so far add concepts the topmost of a tree of contexts. Curly brackets open a new context, e.g.:

```
Person { Name is a String; }
Peter is a Person{"Peter Smith" is the Name;}
Employee { Salary is a Number; }
Programmer is an Employee;
PeterTheEmployee is a Peter, a Programmer {
 30000 is the Salary; }
```

We call the outer concepts the *context* of the inner, and we call the set of inner concepts the *content* of the outer.

In this example, we assume that concepts `String` and `Number` are already defined. The subconcepts created in context are unique specializations in that context only.

As indicated above, concepts from the context of a concept are inherited by refinements. For example, `Peter` inherits the concept `Name` from `Person`.

M3L has visibility rules that correlate to both contexts and refinements. Each context defines a scope in which defined identifiers are valid. Concepts from outer contexts are visible in inner scopes. For example, in the above example the concept `String` is visible in `Person` because it is defined in the topmost scope. `Salary` is visible in `PeterTheEmployee` because it is defined in `Employee` and the context is inherited. `Salary` is not valid in the topmost context and in `Peter`.

C. Contextual Amendments

Concepts can be redefined in contexts. Implicitly, this happens by definitions as those shown above. For example, in the context of `Peter`, the concept `Name` receives a new refinement.

Concepts can be redefined in a context explicitly:

```
AlternateWorld {
 Peter is an Ape {
  "Peter Miller" is the Name; } }
```

We call a redefinition performed in a context different from that of the original definition a *conceptual amendment*.

In the above example, `Peter` is both a `Person` (inherited) and an `Ape` (additionally defined), while the name has been changed.

A redefinition is valid in the context it is defined in, in sub contexts, and in the context of refinements of the context (since the redefinition as part of the content is inherited).

IV. A Conceptual Model for Content Representations

A conceptual model, as known from database modeling, serves as a first step towards data models for context-aware content. The notion of "concept" is ambiguous here: The aim is a model of (M3L) concepts. A conceptual model for this allows us to abstract from the M3L as a language. The model is not supposed to address practical properties such as operational complexity.

A set of M3L concept definitions can be viewed as a graph with each node representing a concept, labeled with the name of the concept. There are two kinds of edges to represent specialization and contextualization. In fact, such a graph forms a hypergraph to account for contextualization. Every node can contain a graph reflecting definitions as the concept's content.

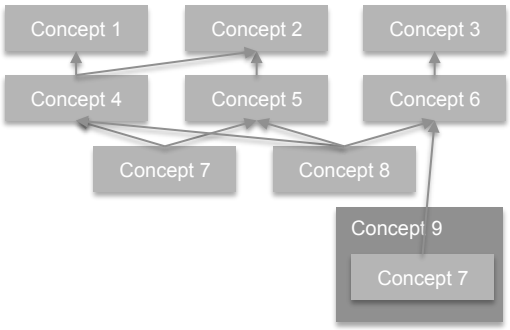

Figure 1. M3L concept refinements and contexts.

The following subsections detail specialization and contextualization relationships, as well as contextual redefinitions.

A. Representing Specialization

Conceptually, a specialization/generalization relationship can straightforward be seen as a many-to-many relationship between concepts. Fig. 1 shows an example.

Arrows with filled heads, directed from a concept to its base concepts, represent specialization relationships in the figure. For example, *Concept 4* is a refinement of *Concept 1* and *Concept 2*.

Fig. 1 furthermore indicates an amendment in a context, namely *Concept 9*. While *Concept 7* is a refinement of *Concept 4* and *Concept 5* in the default context, it is additionally a refinement von Concept 6 in the context of Concept 9 (if it is an `is a`/`is an` definition; otherwise, *Concept 7* would only be a refinement of *Concept 6* in the context of *Concept 9*).

B. Representing Context

Since contexts form a hierarchy, contextualization can be represented by a one-to-many relationship between concepts in the roles of context and content.

Fig. 2 represents such a hierarchy by nesting boxes shown for concepts. The contextualization relationship is thus visually represented by containment. For example, *Concept 2* is part of the content of *Concept 1*, or *Concept 2* is defined in the context of *Concept 1*.

The outermost context is the default context. There is no corresponding concept for this context.

C. Representing Contextual Information

Specialization and contextualization act together. Refinements of a concept inherit its content; concepts from that content are valid also in the context of the refinement. Each context allows concept amendments. These are a second way to add variations of concepts.

In order to represent contextualized redefinitions, we introduce two kinds of context definitions: *Initial Concept Definition* and *Contextual Concept Amendment*. Both can be placed in any context.

An initial context definition is placed in the topmost context in which a concept is defined. Redefinitions of concepts are represented by concept amendments inside the concept in whose context the redefinition is performed.

 ISBN: 978-1-61208-611-8

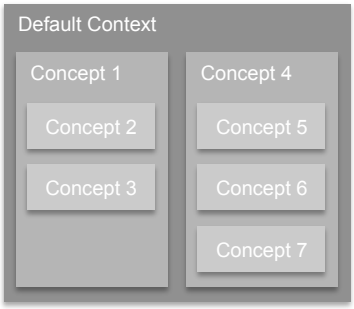

Figure 2. M3L concept definitions in contexts.

Fig. 3 illustrates this. As before, contexts are depicted as nested boxes. There is one *Context* and a *Sub Context*. Both show a *Concept* that has originally been defined as a refinement of *Base Concept* and is itself refined to *Refinement*. In the context on *Sub Context*, the concept gets the additional base concept *Base Concept 2*, and there is another refinement *Refinement 2*. These additions are recorded in the amendment in *Sub Context*.

Amendments have a reference to the next higher definition. This reference is called *Original*. In Fig. 3, it is shown by the dotted line.

Traversal of the original references allows collecting all definitions in order to determine the effective definition.

V. Logical Content Representation

This section refines contextual content representation models to a level similar to that of a logical data model. This way it discusses properties of data representations without taking implementation details into account.

The complexity of lookups is of major importance for the schema design. During the evaluation of M3L statements, many graph traversals are required to find all valid contexts, all base concepts (to determine content sets) and all refinements (to narrow down concepts before applying rules; this evaluation process is not laid out in this paper).

The most important design decision is the degree of (de)normalization of the schema. The basic assumption is that content is mainly queried, so that creation and update cost is less important than lookup cost.

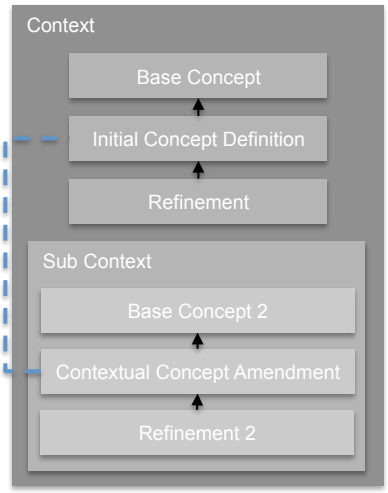

Figure 3. M3L concept amendments in contexts.

We consider two designs of denormalized schemas: materialization of reference sets and storage of relationships in way that allows efficient queries. Efficient storage is based on the usage of numeric IDs to reference concepts and computing relationships based on ID sets. An example of such an approach is the BIRD numbering scheme for trees [10] that allows range queries to determine sub trees.

A. Storing Refinements

Compared to the straightforward conceptual model, the logical schema is denormalized in order to avoid repeated navigation of specialization relationships when collecting the set of (transitive) base concepts or refinements of a concept.

Two approaches are investigated: aggregated concepts and transitive refinement relationships.

Aggregated data collects necessary information to avoid nested queries for refinements. All base concepts and all refinements are stored in an object representing the concept definition. Context-dependent content is added in contextual concept amendments (s.a.) that are stored as part of the context hierarchy.

The description objects additionally reference each other via original references.

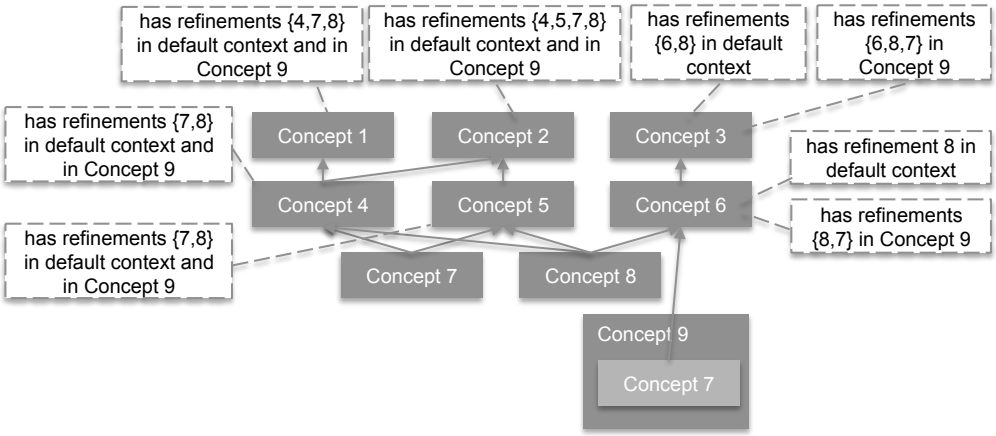

Figure 4. Representation of refinements using materialized transitive refinement relationships.

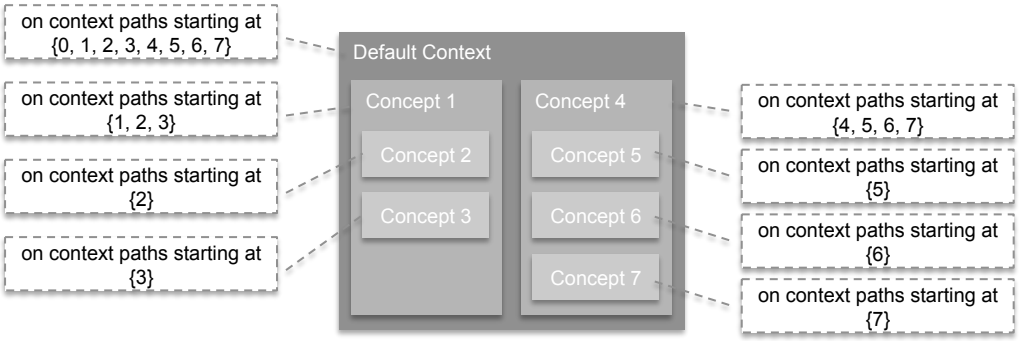

Figure 5. Representation of context hierarchies my materializing paths.

Alternatively, just transitive refinement relationships are materialized for every concept in every context. This way, transitive refinements are directly available, and base concepts can be collected using a simple query.

Fig. 4 shows an example for the sample from Fig. 1. The dashed boxes show the transitive refinements per relevant context. Base concepts can be determined by queries.

For example, the (transitive) base concepts of *Concept 4* are those concepts that have this concept as a refinement. Specifically, these are *Concept 1* and *Concept 2* (in both the default context and in the context of *Concept 9*).

Storing the context together with the refinement relationships is vital for handling singleton (`is the`) relationships, in particular the unbinding of concepts.

B. Storing Context Hierarchies

Performance is particularly important for the retrieval of the hierarchy of contexts a concept is defined or amended in. The effective definition of a concept (including aggregated base concepts and content) relies on this concept hierarchy.

By blending in the context information into the transitive refinements, as shown in the previous subsection, the situation is leveraged to a large degree. Still, the content that a concept has in a certain context is also relevant to concept evaluations.

As for the specialization/generalization relationships, two approaches are discussed here: materialized content collections in all contexts and information about paths in the context hierarchy.

The materialization of contextual definitions works the same way as that of refinements: with every concept definition amendment, we store the effective content in the respective context. This has to be computed on definition.

For the second approach, Fig. 5 illustrates the attribution of paths to the schematic example of Fig. 2. For each concept, we note down the concepts lying on the path in the context hierarchy from the default context to a specific context. For example, *Concept 1* lies on the paths from the default context to itself, to *Concept 2*, and to *Concept 3*.

We used numeric IDs to reference the concept (with the ID 0 given to the pseudo-concept for the default concept). IDs have to be ordered from the default context to sub contexts. By querying for all concepts on the path of a concept, ordered by ID, we retrieve the path to that concept.

VI. Physical Content Storage Models

This section briefly discusses some implementation approaches of context-aware content models. Specifically, we present the basics of a mapping to relational databases and one to a document-oriented database.

A. Mapping M3L to a Relational Database

There is a range of approaches for storing trees and graphs in relational databases [11]. On the basis of these, we add materialized transitive relationships as described above.

Relational tables for the transitive context hierarchy can be defined by statements like (with numeric type INT):

```
CREATE TABLE concept (id INT PRIMARY KEY);
CREATE TABLE paths (
 concept_id       INT REFERENCES concept(id),
 terminal_concept INT REFERENCES concept(id),
 PRIMARY KEY (concept_id, terminal_concept)
);
```

The table *concept* holds concepts (both initial definitions and amendments) with artificial, numeric IDs (other data is omitted here). The second table holds the path information as indicated in Fig. 5. *concept_id* refers to the concept, *terminal_concept* refers to the concept on whose path the concept lies.

Data stored this way can be queried by, e.g.,

```
SELECT c.* FROM concept c, paths p
 WHERE c.id = p.concept_id
 AND p.terminal_concept = i
 ORDER BY p.concept_id DESC;
```

to retrieve the path to concept *i*.

Transitive refinements can be stored in a table:

```
CREATE TABLE transitive_refinements (
 base_concept_id INT REFERENCES concept(id),
 refinement_id   INT REFERENCES concept(id),
 context_id      INT REFERENCES concept(id),
 PRIMARY KEY (base_concept_id, refinement_id,
              context_id));
```

The base concepts of, e.g., *Concept 4* can be queried by:

```
SELECT base_concept_id
 FROM transitive_refinements
 WHERE refinement_id = 4 AND context_id = 0;
```

in the default context (with ID 0), or by:

```
SELECT base_concept_id
 FROM transitive_refinements
 WHERE refinement_id = 4 AND context_id = 9;
```

for the context of *Concept 9*.

 ISBN: 978-1-61208-611-8

```
db.concept.insert({ name: "Default Context", content: [
 { name: "Concept 1", baseConcepts: null,                                    content: null },
 { name: "Concept 2", baseConcepts: null,                                    content: null },
 { name: "Concept 3", baseConcepts: null,                                    content: null },
 { name: "Concept 4", baseConcepts: ["Concept 1", "Concept 2"],              content: null },
 { name: "Concept 5", baseConcepts: ["Concept 2"],                           content: null },
 { name: "Concept 6", baseConcepts: ["Concept 3"],                           content: null },
 { name: "Concept 7", baseConcepts: ["Concept 4", "Concept 5"],              content: null },
 { name: "Concept 8", baseConcepts: ["Concept 4", "Concept 5", "Concept 6"], content: null },
 { name: "Concept 9", baseConcepts: null,                                    content: [
  {name: "Concept 7", baseConcepts: ["Concept 4", "Concept 5", "Concept 6"], content: null,
   original: "Concept 7" } ] } ] })
db.concept.aggregate([
{$unwind:"$content"},{$replaceRoot:{newRoot:"$content"}},{$match:{name:"Concept 9"}},
{$unwind:"$content"},{$replaceRoot:{newRoot:"$content"}},{$match:{baseConcepts:"Concept 6"}}])
```

Figure 6. Document definitions to map M3L to MongoDB and a sample query.

B. Mapping M3L to a Document Database

As an example of so-called NoSQL approaches, we conduct ongoing experiments with MongoDB, a widely used document-oriented database management system.

The definition of concept relationships is done a similar way as in relational databases: records have IDs, and records store IDs for references. There are no distinct relation structures, though. References are stored as document fields.

In contrast to a purely relational structure, documents allow representing nested contexts in a natural manner by embedded documents.

As an example of a schema, the *insert* statement shown in Fig. 6 stores the whole graph of Fig. 1.

This structure can be queried as required. For example, to find concepts with base concept *Concept 6* in the context of *Concept 9*, the *aggregate* statement in Fig. 6 can be applied.

VII. CONCLUSION

This section sums up the paper and gives an outlook on future work.

A. Summary

In this paper, we laid out approaches to context-aware content management, in particular using the Minimalistic Meta Modeling Language (M3L).

Though it is easily possible to map context representations to existing data management approaches, care has to be taken to achieve efficient implementations.

A logical schema for the representation of contextual content is presented, and first implementations are conducted. These demonstrate the feasibility of the schemas.

B. Outlook

The work on the data model mappings for M3L concept definitions is ongoing work; there is ample room for further optimizations of the relational database schema. The mapping to document-oriented database needs much more elaboration before comparisons can be made.

The utilization of databases to support M3L concept evaluation is an open issue. Practical rule sets will guide the investigations in the future.

Experiments with different implementations are ongoing. Data models have yet to be rated based on practical results.

ACKNOWLEDGMENT

Though the ideas presented in this paper are in no way related to Namics, the author is thankful to his employer for letting him follow his research ambitions based on experience made in customer projects. The discussions with colleagues, partners, and customers are highly appreciated.

REFERENCES

[1] M. Gutmann, “Information Technology and Society,” Swiss Federal Institute of Technology Zurich / Ecole Centrale de Paris, 2001.

[2] C. Bolchini, C. A. Curino, E. Quintarelli, F. A. Schreiber, and L. Tanca, “A Data-oriented Survey of Context Models,” ACM SIGMOD Record, vol. 36, pp. 19-26, December 2007.

[3] A. Zimmermann, M. Specht, and A. Lorenz, “Personalization and Context Management,” User Modeling and User-Adapted Interaction, vol. 15, pp. 275-302, Aug. 2005.

[4] S. Trullemans, L. Van Holsbeeke, and B. Signer, “The Context Modelling Toolkit: A Unified Multi-layered Context Modelling Approach,” Proc. ACM Human-Computer Interaction (PACMHCI), vol. 1, June 2017, pp. 7:1-7:16.

[5] G. Orsi and L. Tanca, “Context Modelling and Context-Aware Querying (Can Datalog Be of Help?),” Proc. First International Conference on Datalog Reloaded (Datalog '10), Mar. 2010, pp. 225-244.

[6] D. Ayed, C. Taconet, and G. Bernard, “A Data Model for Context-aware Deployment of Component-based Applications onto Distributed Systems,” GET/INT, 2004.

[7] S. Vaupel, D. Wlochowitz, and G. Taentzer, “A Generic Architecture Supporting Context-Aware Data and Transaction Management for Mobile Applications”, Proc. International Conference on Mobile Software Engineering and Systems (MOBILESoft '16), May 2016, pp. 111-122.

[8] J. W. Schmidt and H.-W. Sehring, “Conceptual Content Modeling and Management,” Perspectives of System Informatics, vol. 2890, M. Broy and A.V. Zamulin, Eds. Springer-Verlag, pp. 469-493, 2003.

[9] H.-W. Sehring, “Content Modeling Based on Concepts in Contexts,” Proc. Third Int. Conference on Creative Content Technologies (CONTENT 2011), pp. 18-23, Sep. 2011.

[10] F. Weigel, K. U. Schulz, and H. Meuss, “The BIRD Numbering Scheme for XML and Tree Databases – Deciding and Reconstructing Tree Relations using Efficient Arithmetic Operations,” Proc. Third international conference on Database and XML Technologies (XSym'05), Aug. 2005, pp. 49-67.

[11] V. Tropashko, SQL Design Patterns: The Expert Guide to SQL Programming. Rampant Techpress, 2006.

 ISBN: 978-1-61208-611-8