A Contextualised Object Data Model based on Semantic Values

Alex G. Büchner[*], David A. Bell[†], John G. Hughes[*]

[*]Northern Ireland Knowledge Engineering Laboratory, University of Ulster
[†] School of Information and Software Engineering, University of Ulster
email: {ag.buchner, da.bell, jg.hughes}@ulst.ac.uk

Abstract

Context mediation based on semantic values is a promising approach to resolve semantic heterogeneity in multidatabase systems. To represent contextualised data, the concept of contexts is introduced, which encompasses context identity and semantic values, contextualised equivalence specifications for atomic and complex types, context mediation, as well as a hierarchical context topology. The introduced concepts are embedded in an object data model, viz. ODMG, and its object definition and query language are extended accordingly.

Keywords: semantic interoperability, semantic values, context mediation, object data models.

1 Introduction

A promising approach to resolve semantic heterogeneities among heterogeneous information systems is the concept of semantic values and context mediation [1]. Each local data source expresses its semantics declaratively with reference to the underlying ontology. Context information includes properties about a data value and is represented by the concept of semantic values, which are a synergy of a traditional value and a context. When a data value, as opposed to a semantic value, is being exchanged from a source to a receiver, it also migrates from the source context to the receiver context. To perform this migration, a conversion mechanism has to be declared which is, like semantic values and contexts themselves, part of the database schema.

This approach, although very promising and powerful, lacks some fundamental issues, which are being tackled in this paper. Firstly, the concept lacks the handling of complex objects. Secondly, due to the lattice-like representation, the order of complexity of conversion functions is quadratic. Lastly, it lacks a formal specification of the introduced concepts.

The contribution of this work is a contextualised object data model which is based on the concept of semantic values. The outline of this paper is follows. §2 introduces the concept of contexts which deals with context identity and semantic values, contextualised equivalence specifications for atomic and complex types, context mediation, and the hierarchical organisation of contexts. In §§3-5, we extend the ODMG model with these concepts, as well as its object definition and query language. §6 compares related work, before conclusions are drawn and further research is outlined in §7. Throughout the paper, we use an internationalised literature database to illustrate the proposed concepts.

2 The Concept of Contexts

2.1 Context Identity and Semantic Values

A context represents behavioural aspects which are shared by attributes of the same ontology. Assuming an underlying internationalisation ontology, possible properties are the exchange rate of a currency, the scale of a numeric value and the granularity of any attribute.

Definition 1. A **context** c contains a set of properties $P_c = \{p_{c_1}, p_{c_2}, p_{c_3}, \ldots, p_{c_n}\}$ where $P_c \subseteq P$ and $P = \{p_1, p_2, p_3, \ldots, p_m\}$, which represents an application-specific ontology O. A **context identifier** cid is a unique representation of a context c, which is chosen from a set of contexts $C = \{c_1, c_2, c_3, \ldots, c_n\}$.

The general idea of the context identity concept is that every single attribute instance is being allotted an additional attribute context identifier. Thus, each attribute instance is represented by a semantic value, which is defined as following.

Definition 2. A **semantic value** s consists of a type t, a value v, and a context identifier cid, such that each contextualised entity in an information space is represented as the triple $s = (t, v, cid)$.

This approach is a generalisation of proposals, in which every attribute – but not each attribute instance – has a specific context allotted to it. Such tagging methods have been used in a wide range of applications, such as class variables in object-oriented systems, meta-knowledge in expert systems, or rendering information in word processors and mark-up languages. Consequently, every context contains type- *and* context-sensitive semantics and, thus, controls the behaviour of these opaque data types. Behaviour in a generic database sense means that equality has to be re-defined to allow a uniform representation of new functionality in order to handle information in contextualised (multi)databases. Therefore, traditional comparison operations have to be replaced by contextualised ones, i.e. the concept of equality has to be substituted by that of equivalence (also referred to as semantic equality). A type-specific semantics is required to express the behaviour of the taxonomies of operations which can occur when dealing with contextualised data.

2.2 Contextualised Comparison Operations

Information from different data sources is interpreted differently by different data receivers, and thus comparisons between and among values of different contexts have to be specified.

Definition 3. Let $s_1 = (t_1, v_1, cid_1)$ and $s_2 = (t_2, v_2, cid_2)$ be two semantic values and q be a **comparison operator** such that $q \in \{=, <, <=, >, >=, <>\}$. Then the expression $e = s_1\, q\, s_2$ is valid iff $t_1 = t_2$ or t_1 can be converted to t_2 and vice versa[1].

Dyadic operations upon data with identical type and context can be performed traditionally; dealing with data from different contexts depends crucially on q-operations. With dyadic operations from different contexts, the operands are either in canonical forms, have to be transformed to such, or a particular q-operation has to be computed which is sensitive to both contexts c_1 and c_2. While unary operations are straightforward, operations involving comparison require a mechanism which guarantees semantic equivalence among values from different contexts.

2.3 Context Comparison Specification

Formulating comparisons to resolve semantic heterogeneity among values from different contexts cannot be expressed in a single mechanism. Examples of the diversity of comparisons are collation sequences (including complex orders, like multiscript sorting), transformations of different calendaric systems, or operations upon exchange rates in different currencies. To perform equivalence operations as specified in Definition 3, each value has to be converted into a form in which traditional q-operations can be computed. Due to the fact that there *is* no canonical form of contextualised values of certain types, a more sophisticated approach is required, which takes the contexts of values into account. Sciore, et. al. [1] have identified five different ways of declaring semantic value specifications, namely rules, predicates in logic, functional expressions, tables, and tagged attribute properties. While numeric-based data types, e.g., currencies or dates, can usually be transformed into a context-independent canonical form, this is does not hold for strings, abstract and complex data types, which might behave inconsistently in different source contexts. Additionally, different receivers might want to specify individual comparison operations.

The maximum number of specifications for each type is of complexity $O(|C|) = |C|^2$, where $|C|$ denotes the cardinality of the set of contexts C. Although this quadratic order could theoretically lead to an explosion of comparison specifications, in reality often few specifications are required. Additionally, in §2.5, a hierarchical context inheritance mechanism will be introduced to reduce the number of comparison specifications. To specify the comparison of two contextualised values, a context mediator, which takes the semantic values and returns their type- and context-specific order, is being built.

1 Whether a semantic value can be converted will be specified in §2.6, when context mediation is introduced.

Definition 4. A **context mediator** *med* is an evaluation mechanism that takes the semantic values s_1 and s_2 as input and returns the order of that value pair, such that

$$med\ (s_1, s_2) = \begin{cases} -1 & \text{if } s_1 > s_2 \\ 0 & \text{if } s_1 = s_2 \\ 1 & \text{if } s_1 < s_2 \\ \text{Null} & \text{if undefined} \end{cases}$$

The output of the mediator *med* is also called the order o of s_1 and s_2, which simplifies defining the specification of orders for complex data types (see §2.4). The specification of a generic context mediator is described in detail in §2.6. Facilitating the mediator as specified, although currently a black box, it is now possible to define the following.

Definition 5a. Let $s_1 = (t_1, v_1, cid_1)$ and $s_2 = (t_2, v_2, cid_2)$ be two semantic values and $e = s_1\, q\, s_2$ a valid expression. s_1 is **equal** to s_2 iff $(v_1 = v_2) \wedge (t_1 = t_2) \wedge (cid_1 = cid_2)$. s_1 is **equivalent** to s_2, iff $med(s_1, s_2) = 0 \wedge (cid_1 \neq cid_2)$.

Since context equivalence is a superset of context equality, the former can be embedded in the latter and all comparisons between two semantic values can be carried out using the context mediator *med* – independently of the content of the two context identifiers:

Definition 5b. Let $s_1 = (t_1, v_1, cid_1)$ and $s_2 = (t_2, v_2, cid_2)$ be two semantic values and $e = s_1\, q\, s_2$ a valid expression. s_1 is **semantically equal** to s_2, iff $med(s_1, s_2) = 0$.

2.4 Complex Data Types

In addition to atomic data types, the concept of context identity also has to be supported by complex types as well as any arbitrary user-defined abstract type. Also, it is insufficient to calculate semantic in/equality only, since an order of data of complex type is required for various database-related operations (sorting, grouping, indexing). To compute the order of two complex objects with the same structure, we define the following measure, which gives elements with a smaller index higher priority and which is normalised to the integer interval [-1,1] via the signum function.

Definition 6. Given two complex objects $o_1(s_{1_1}, s_{1_2}, s_{1_3}, ..., s_{1_n})$ and $o_2(s_{2_1}, s_{2_2}, s_{2_3}, ..., s_{2_n})$, the **object order** $o(o_1, o_2)$ is defined as

$$o(o_1, o_2) = \text{sgn}\left(\sum_{i=1}^{n} (n-i+1)^2 * med\ (s_{1_i}, s_{2_i}) \right)$$

Operations upon objects with different structure depend on the type of the (complex) semantic values. We now define generic orders for two kinds of complex types, which can be overridden in application-specific mediators. The collection types considered in here are sets and vectors. For sets, the results of the mediator for each individual element of the difference sets are calculated and mapped onto the result set $\{-1,0,1\}$.

Definition 7. Let $o_1^s = \{s_{1_1}, s_{1_2}, s_{1_3}, ..., s_{1_m}\}$ and $o_2^s = \{s_{2_1}, s_{2_2}, s_{2_3}, ..., s_{2_n}\}$ be two contextualised sets, and $\Delta_1 = o_1^s - o_2^s$ and $\Delta_2 = o_2^s - o_1^s$ be two difference sets, respectively. The **set order** $o(o_1^s, o_2^s)$ is defined as

$$o(o_1^s, o_2^s) = \begin{cases} \operatorname{sgn}(|\Delta_1| - |\Delta_2|) & \text{if } \Delta_1 = \varnothing \vee \Delta_2 = \varnothing \\ \operatorname{sgn}\left(\sum_{i=1}^{|\Delta_1|} \sum_{j=1}^{|\Delta_2|} med\left(s_{1_i}, s_{2_j}\right) \right) & \text{otherwise} \end{cases}$$

For set operations ($\cap$, $\cup$, $-$, $\supset$, $\subset$, $\in$ and derivatives thereof), as well as composite predicates ($\forall$ and $\exists$), the existing context mediators are used to evaluate the results. The second collection type to be defined is type vector, which re-uses the lexicographic object order of Definition 6 for physically overlapping components and adds a fraction for the remaining part which is only relevant in case of equality.

Definition 8. Let $o_1^v = \langle s_{1_1}, s_{1_2}, s_{1_3}, ..., s_{1_n} \rangle$ and $o_2^v = \langle s_{2_1}, s_{2_2}, s_{2_3}, ..., s_{2_n} \rangle$ be two contextualised vectors, and $m = \min(m, n)$**.** The **vector order** $o(o_1^v, o_2^v)$ is defined as

$$o(o_1^v, o_2^v) = \operatorname{sgn}\left(\left(\sum_{i=1}^{m} (m - i + 1)^2 * med\left(s_{1_i}, s_{2_i}\right) \right) + \frac{m-n}{\max(m,n)} \right)$$

2.5 Topological Organisation of Contexts

To minimise the redundancy of context declarations (e.g., linguistically related writing systems or dates in different time zones but from a single calendaric system), contexts are organised in a hierarchy. Thus, structure and behaviour can be inherited, and overloading and overriding mechanisms can be applied to contexts.

There are two ways in which to organise contexts hierarchically. One is by types, i.e. for every data type a context-based hierarchy is built (Figure 1a). The other is by contexts, i.e. contexts are organised in a hierarchy and for every type structure and mediators are specified (Figure 2b). Depending on the degree of fluctuation of types and contexts respectively, a system is easier to maintain, which has been indicated by the context-related shaded parts in each hierarchy. It is felt that it is more natural to group types around contexts, rather than the other way round, and hence, this approach has been chosen. Also, new contexts are usually added to a system, rather than new data types with *different* context characteristics. The root context node is always Context, which contains some basic generic structure (like the name and properties) as well as behaviour (e.g., the mediation execution mechanism and order computations) which is shared by all inheritors, unless overridden. The whole spanning context tree is representing the underlying ontology *O*.

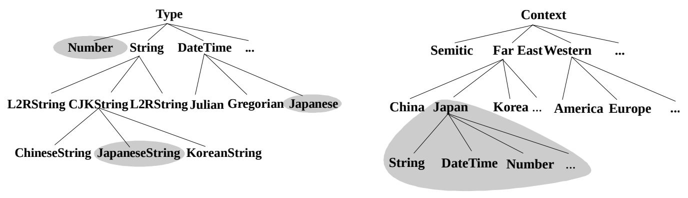

Figure 1. (a) Type-based Context Hierarchy
(b) Context-based Type Hierarchy

Definition 9. A **context hierarchy** *O* is an undirected, connected, acyclic graph which is defined as the tuple $O = (C,E)$, where $C = \{c_0, c_{1_1}, c_{1_2}, ..., c_{1_{|c_1|}}, c_{2_1}, c_{2_2}, ..., c_{2_{|c_2|}}, ..., c_{n_1}, c_{n_2}, ..., c_{n_{|c_n|}}\}$ and $E = \{e_1, e_2, e_3, ..., e_m\}$. Each e_k has the form $e_k = \{c_i, c_j\}$; $c_i, c_j \in C$, c_0 has indegree 0, $c_1 ... c_n$ have indegree 1. c_i is **subcontext** of c_j iff $c_i \subset c_j$; c_i is **supercontext** of c_j iff $c_j \subset c_i$.

A further advantage of the hierarchical arrangement of contexts is the possibility of packaging context sub-trees; we refer to these sub-trees as resources.

Definition 10. A **resource** *r* is defined as a subgraph of a context hierarchy *O*, such that $r \subset O$, i.e. $r \in O \wedge r \neq O$.

An example of a resource is the highlighted Japan subgraph in Figure 1b which contains the following properties:

```
JapaneseString(WritingSystem ∈ {Hiragana, Katakana, Kanji})
JapaneseDate(Calendar = Emperor; TZ = +8)
JapaneseNumber()
```

In this particular case, Japanese strings have one property, which can take three possible values, dates have two properties and numbers inherit all properties from its supercontext, FarEast. The operational part of contexts will be outlined in the next subsection.

2.6 A Primitive Context Mediator

The purpose of the context mediator is to compute the order of two contextualised semantic values. A variety of constructs has been suggested in the literature (see §6), but to reconcile context heterogeneity, functions (which encompass arithmetic expressions as well as rules) and tables are sufficient. Although functions have been designed to convert numbers and derivations thereof (like datetimes), they can also be used by strings or data of abstract type. Similarly, tables have originally been designed to represent collation sequences of characters, but they are not limited to them; data from any other type can be organised in tables, too. The generic context mediator skeleton, which is based on Definition 5b, has the following layout:

```
med ← s1, s2
    function <conversion_function> | table <collation>
med → {-1, 0, 1}
```

The context mediator *med* takes two semantic values s_1 and s_2; their types and contexts can be, but are not mandatorily, different. <conversion_function> is an implementation-specific set of linguistic programming constructs, which should cover minimal, but sufficient functionality, and <collation> is the name of an ordering table. The mediator returns the order of the two semantic input values where the flags –1, 0 and 1 represent 'less than', 'equal' and 'greater than', respectively. Note that the mediator does not return the context of the result. This will either be requested explicitly by a formulated query, or implicitly given by default contexts (see §4.1 and §5).

2.7 Recapitulation of Proposed Work

The topology independent layer view in Figure 2 graphically presents the concept of contexts. For simplicity, only a minimised context hierarchy is used from the internationalisation example, and only a few attribute instances are allocated to properties.

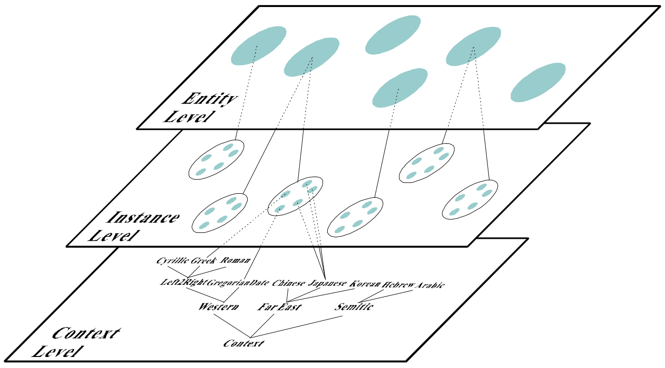

Figure 2. Topology-independent layer view

The four proposed fundamental constructs (context identity and semantic values, contextualised equivalence, inheritance of contexts, and context mediation) will now be embedded in an appropriate data model.

3 Contextualised Object Data Model

Embedding the concept of contexts in a data model requires the representation of context semantics and inheritance, as well as polymorphism mechanisms in order to build a context hierarchy. Traditional data models are inappropriate in providing these facilities, and so an object data model has been chosen. For the purposes of this paper, an evolutionary approach has been preferred, and the ODMG object model [2] has been used. However, none of the extensions made here are ODMG-specific; they can be applied to any object data model.

A new meta data type Context is created which contains information about contexts themselves. Its structural part consists of a unique name identifier (context identity), a set of properties and a set of super_contexts. To model single context inheritance, a recursive relationship, referred to as subcontextualising, has been embedded. Two methods to define and remove contexts (create and delete) as well as two methods to attach and detach mediators to a context (add_mediator and remove_mediator) are needed. To check the existence of a context, an exists? method is required.

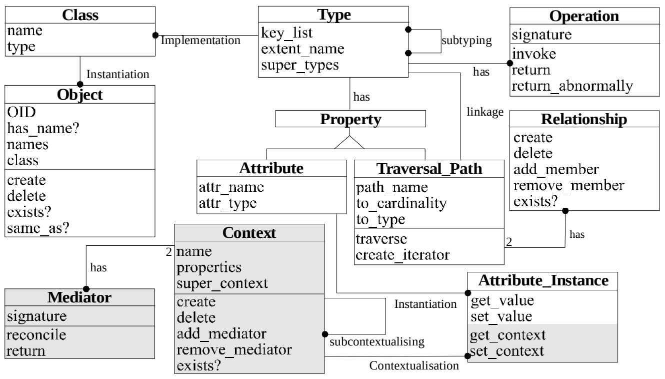

Figure 3. Localised ODMG meta model[2]

To connect the type Context, a 1:n relationship Contextualisation to the abstract type Attribute_Instance has been added. To keep the principle of encapsulation, the type Attribute_Instance has to be enhanced by the mutator method set_context to change the context of an attribute instance, and the accessor method get_context to retrieve it.

The structural part of the Mediator contains a signature, which consists of the mediator name, the mediator type (function or table), as well as the types and contexts of the two semantic input values analogous to the specification in §2.6. To send a message to a mediator, the reconcile method has to be called, which uses the return method for the response. The Mediator type is connected through a 2:n has-relationship to the Context type. Note that the mediator could have been modelled as part of the context, but for reasons of encapsulation and scalability it has been designed as a separate component.

The three outlined extensions (introduction of the Context and Mediator types as well as attribute instance enhancements) lead to the simplified ODMG meta model depicted in Figure 3. The diagram makes use of the graphical notation of OMT, highlighting the added contextualisation features. In addition, to represent context properties and behaviour transparently, the model is capable of packaging contexts and connected resources, and using them when needed.

4 Context Object Definition Language

To keep consistency with the ODMG model enhancements incorporated above, the ODMG-ODL will be extended in this section. Again, the extensions are not ODMG-specific; syntactical statements are just borrowed for convenience.

In general, in ODMG every type is defined as an interface, to be compatible with the OMG interface definition language IDL. An interface consists of an extent, a set of keys, a number of attributes and relationships, as well as a range of operation signatures. Only a few syntactical extensions are necessary to support all contextualised semantic constructs of the contextualised object data model.

[2] Modified from M.E.S. Loomis [3].

4.1 Attribute Declaration

An optional default context, which can be allotted to every attribute definition, is being added. Another optional constraint is the limitation of one attribute to a non-empty set of contexts. The modified BNF for specifying an attribute looks as following.

```
<attrib_dcl> ::=
      [readonly] attribute
      <domain_type> <identifier>
      [[<positive_int_const>]]
      [ContextDefault <context>]
      [ContextConstraint <context_set>

<context_set> ::=
      <context> | <context> , <context_set>

<context> ::= <identifier>
```

If both properties are assigned, ContextDefault must be a member of ContextConstraint. If a type has a supertype, context attribute properties are inherited and can be overridden. If either ContextDefault or ContextConstraint is overridden, clashes have to be detected automatically, i.e. ContextDefault still has to be a member of ContextConstraint. Allocation of a ContextDefault to an attribute of complex type can lead to misinterpretation, because it is unclear whether the context is meant to be default for the entire complex object, or for every single element. We assume that it is both, since it is a natural way of declaring default contexts for complex data types.

4.2 Context and Mediator Declaration

Due to the fact that contexts are treated similarly to types and mediators similarly to operations, they too have to be part of the database schema, and therefore syntactical constructs have to be provided for their declaration. Declaration of contexts consists of naming a context identifier and supercontexts, which are implicitly given in the specification of a context. Declaration of mediators requires a signature (identifier and a mediator type) as well as the mediator implementation. The BNF for Context and Mediator declaration are listed in the appendix.

The following is the declaration of the three contexts Western (subcontext of Context), USA (subcontext of Western) and USMountain (subcontext of USA) for the data type Date. WesternDate contains one property Gregorian which is the only permitted value, USADate is a virtual context without any additional properties, and USMountainDate inherits the Calendar property from the context Western and adds TZ as additional property.

```
Context Western (Date) : Context
     Properties Calendar
          PropertyDefault Gregorian
          PropertyConstraint Gregorian

Context USA (Date) : Western

Context USMountain (Date) : USA
     Properties TZ
          PropertyDefault -6
```

The specification of two simple alternative context mediators to specify the comparison from US$ to HK$ with and without 10% overhead for amounts over 10.000 HK$ can be defined as follows.

```
Mediator USA_HK Function USA(Currency) HK(Currency)
(
      if HK.Value > 10000
      then (HK.Value * 0.12) * 0.1
      else HK.Value * 0.12
)

Mediator USA_HK Table USA(Currency) HK(Currency)
(
      NYSE
)
```

As can be seen from the given example, values which can be accessed through the dot notation (as can all other properties) are implicitly converted to a temporary canonical form, which then can be evaluated by the mediator execution mechanism and the comparison result returned accordingly. It is important to note that it is possible that semantic values of different types (atomic, complex or abstract) can be compared, which is regularly required for semantic heterogeneity conflict resolution. Also, different users in different contexts can replace each mediator with their own perceptive version, which guarantees the autonomy of the source and receiver sites. After specifying syntactical extensions of the structural section, the operational part has to be enhanced, too.

5 Context Object Query Language

To represent context information within queries minor syntactical extensions have to be introduced to the ODMG-OQL. Every context property of every attribute has to be accessible, which is realised through the following notation, where attribute can be either an atomic or a complex object:

```
<attribute>@<property>
```

To guarantee orthogonality of the query language, context identity can be used with every attribute in all valid query statements. For instance, to retrieve all titles which are allocated to the Japanese context and written by American authors (authors who have been allotted to the context USA!), the following query has to be formulated:

```
SELECT  r.title, r.title@WritingSystem
FROM    References r
WHERE   r.author@WritingSystem = USA AND
        r.title@WritingSystem = Japan
```

Internally, the context information must be accessed through the defined accessor method get_context. Joins now have enhanced semantics, because *q*-operators check for semantic values. With the introduction of the context indicator @ and the usage of contextualised comparison operators all queries can be expressed, which also holds for aggregate functions. If no specific context information is given, the newly defined comparison operators are used to imitate these operators. But it might also be of interest to sort and/ or group a retrieved set by its context, e.g.,

```
SELECT     r.author, r.title
FROM       References r
GROUP BY   cheap:      r.price <= 20@Context = USA,
           normal:     r.price BETWEEN 20@Context AND
                        30@Context =USA,
           expensive:  r.price >= 30@Context = USA
ORDER BY   r.price@Context, r.author
```

The few examples above illustrate the usage of the contextualised OQL and show that only one simple extension, viz. the @ notation, is necessary to formulate queries dealing with context heterogeneities.

6 Related Work

Our research has been motivated by Sciore, Siegel and Rosenthal's [1] concept of semantic values to represent semantically heterogeneous information. As mentioned earlier, the system, which is based on the relational calculus, lacks the handling of complex objects ("*[...] context information needs to be attached to objects larger than single attributes; this may me be easier in object-oriented models.")* and, due to the lattice-like representation, leads to an explosion of conversion functions. There have been a large number of alternative endeavours to resolve semantic heterogeneity resolution. An almost exhaustive survey is given by R. Hull [4], which covers virtual and materialised integrated views, updateable mediators, workflow systems, as well as meta-data manipulation. The explicit consideration of contexts as part of the reconciliation of semantic heterogeneity is a relatively new research area.

Ouksel and Naiman [5] define context as the semantic knowledge which is required to perform queries in heterogeneous environments. Each context is specified as a set of inter-schema correspondence assertions describing the semantic relationship between two entities, each containing conflicts about naming, abstraction and level of heterogeneity. Our approach is currently not concerned about context building, but context representation and interchange. Kashyap and Sheth [6] represent context as a collection of contextual co-ordinates and values assigned to them, represented as description logic expressions. Semantic similarities are used to identify different levels of semantic proximity of two objects. Context is represented at an intensional level, as opposed to our extensional representation. The incorporation of context uncertainty in our object model is work in progress (see §7).

7 Conclusions and Further Work

We have presented a contextualised object data model which deals with semantic domain heterogeneity in a more flexible way than existing systems. The concept of context has been the basis of our work. This encompasses context identity and semantic values, localised equivalence specifications, context mediation and a context hierarchy. We have extended the ODMG data model with the proposed concepts, including its object definition and object query language.

Future work is twofold. We have developed a prototype in which mappings (mediator conversion functions as well as tables) between atomic contextualised values can be specified. First benchmark results have shown that execution times increase by a factor 10 to 12. The support of context hierarchies increases by another 8-10% per inheritance level. Thus, query optimisation strategies for contextualised semantics are required. The concept of contextualised reconciliation has been specified for resolvable properties, which is insufficient in certain situations. The data model as well as its definition and query language are currently extended with similarity measures to support unresolvable properties in the context of semantic heterogeneity.

8 References

[1] E. Sciore, M. Siegel, A. Rosenthal, "Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems", ACM Trans. on Database Systems, 19(2):254-290, 1994.

[2] R.G.G. Cattell, D. Barry, D. Bartels, M. Berler, J. Eastman, S. Gamerman, D. Jordan, A. Springer, H. Strickland, D. Wade (eds.), The Object Database Standard: ODMG-93 Release 2.0, Morgan Kaufmann Publishers, San Mateo, CA, 1997.

[3] M.E.S. Loomis, "The ODMG object model", Journal of Object-Oriented Programming, 6(3):64-69, 1993.

[4] R. Hull, "Managing Semantic Heterogeneity in Databases: A Theoretical Perspective", Proc. 16th ACM Symp. on Principles of Database Systems, pp. 51-61, 1997.

[5] A.M. Ouksel, C.F. Naiman, "Coordinating Context Building in Heterogeneous Information Systems", Journal of Intelligent Information Systems, 3:151-183, 1996.

[6] V. Kashyap, A. Sheth, "Semantic and Schematic Similarities between Database Objects: A Context-based Approach", The VLDB Journal, 5(4):276-304, 1996.

Appendix: Context and Mediator BNF

```
<context_dcl> ::=
  Context <identifier> ( <Type> ) [: <context>]
  [Properties <context_properties>]

<Type> ::= <atomic_types> | <complex_types> | <abstract_types>

<context_properties> ::=
  <context_property> | <context_property > , <context_properties>

<context_property> ::=
  <identifier> PropertyDefault <context> PropertyConstraint
  <context_set>


<mediator> ::=
  Mediator <mediator_signature>
  (
  <mediator_body>
  )

<mediator_signature> ::=
  <identifier> Function | Table <locale> (<Type>) <locale> (<Type>)

<mediator_body> ::=
  <mediator_function> | <mediator_table>

<mediator_function> ::= <mediator_conversion_function>

<mediator_table> ::= <identifier>
```