VALUE RECONCILIATION IN MEDIATORS OF HETEROGENEOUS INFORMATION COLLECTIONS APPLYING WELL-STRUCTURED CONTEXT SPECIFICATIONS

D. O. Briukhov, L. A. Kalinichenko, N. A. Skvortsov, S. A. Stupnikov
Institute for Problems of Informatics RAS

Abstract: Method for value reconciliation in Local as Views (LAV) mediators of heterogeneous information collections applying well-structured specifications is presented. This approach extends a procedure for heterogeneous information sources registration at subject mediators with LAV organization. Conflicts in value semantics and representation in contexts of the mediator and a collection should be recognized and explicitly specified. According to the proposed method, value semantics contexts should be defined for the mediator and for a collection. In such context for each value kind a type is to be specified that includes a generic function converting values from a collection context into the mediator context (and/or back). Each type attribute having the respective value semantics is to be typed with such type definition. Such structuring of type attribute semantics definition gives significant economy in development of value conversion functions needed to reconcile values between the mediator and collection contexts.

1. Introduction

Mediation architecture introduced in [14] defines an idea of a middleware positioned between information collections (information providers) and information consumers. Mediators support modelling facilities and methods for conversion of unorganized, non-systematic population of autonomous information collections kept by different information providers into a well-structured information source defined by the integrated uniform specifications. Mediators provide a uniform query interface to multiple data collections, thereby freeing the user from having to locate the relevant

collections, query each one in isolation, and combine manually the information from them. In an approach to the mediator architecture known as Local as Views (LAV) [4] schemas exported by collections are considered as materialized views over virtual classes of the mediator schema. Queries are expressed in terms of the mediator schema. Query evaluation is done by query planning making its rewriting in terms of the source schemas [11, 3]. The LAV architecture is designed to cope with a dynamic, possibly incomplete set of collections. Collections may change their exported schemas, become unavailable from time to time. LAV is potentially scalable with respect to a number of collections involved.

According to another approach, called Global as View (GAV) [5, 13], the global schema is constructed by several layers of views above the schemas exported by pre-selected sources. Queries are expressed in terms of the global schemas and are evaluated similarly to the conventional federated database approaches [12].

The work reported in this paper has been done in frame of a research project (supported by the grant of the Russian Foundations for Basic Research N 01-07-90084) devoted to investigation of the LAV mediator architecture, models and algorithms intended for the subject mediation approach. This approach is applicable to various subject domains in science, cultural heritage, mass media, e-commerce, etc. A mediator schema (including terminological, ontological, structural and behavioral specifications) defines a context specific for concrete subject domains. Various information collections can be registered at the mediator at any time. Users may not know anything about the registration process and about the collections that have been registered. To query information in the subject, users should know only the subject domain definition – the mediator schema. In this project for the canonical model of a mediator the SYNTHESIS language [8] that is a hybrid semi-structured/object model [9] has been chosen.

In [2] methods and tools required to support information collections registration at the mediator were presented. For the LAV mediation strategy it was proposed to apply the information source registration similarly to the process of compositional information systems development [1]. Local source metainformation definitions are treated as specifications of requirements and classes of the mediator level with the related metainformation - as specifications of pre-existing components. To get local classes definitions as materialized views above the mediated level, complete type specifications are assumed. Ontological specifications are used for identification of the mediator classes semantically relevant to a source class. Maximal subset of source information relevant to the mediator classes is identified (due to use of the most common reducts to identify maximal commonality between a source and federated level class specifications). Concretizing types are defined so

that the mediator classes instance types could be refined by the source instance types. This is a natural direction supporting a plan rewriting a mediator query in terms of registered sources. Such refining direction (bottom-up) is in contrast to conventional compositional development [1] where specification of requirements is to be refined by specifications of components (top-down). Such inversion is natural for registration process in the LAV mediator architecture: a materialized view (requirements) is constructed over virtual specifications (components).

Heterogeneous information collections are autonomous units created in different information contexts. The contexts depend on a subject domain of a collection, concepts definitions and real world models applied, value representation used. Heterogeneity of contexts manifests in different understanding of concepts of subject domains, different types definitions, different value appearance. The method [2] considered contextualization of collections in the mediator's context. Ontologies, structural and behavioral type and class specification were emphasized in this method.

After ontological and structural specifications of collections are contextualized in the mediator, the value semantics and representation in collections and the mediator may remain different. It means that during registration, various *features* of values should be taken into consideration. Different scales, measurement units, formats of data representation are just a few of the examples of such features. Heterogeneity of object values may be resolved with the help of value transformation rules applied on moving data from one context to another.

A systematic approach for resolving of the value heterogeneity is needed. In every case where values of object attributes in the mediator and collection contexts are not compatible it is required to develop a respective reconciliation function. In this paper an approach is introduced for providing systematic specifications during registration of information collections at subject mediators intended for reconciliation of value semantics and representation conflicts in different contexts.

A specifically structured description of type attribute semantics is proposed. For this purpose in the mediator context classes of values having specific semantics should be identified. For each of such classes a concept defining value semantics in terms of characterizing the value features should be specified. Features may have different values for concepts in different contexts. For proper features evaluation in a concrete context a specific function is to be included as a part of the value semantic concept. Finally, for each such class of values a type is to be specified that includes a generic function converting values between contexts. This function is controlled by the respective value features in the contexts involved. Each type attribute having the respective value semantics is to be typed with such type

definition. Similarly a value context is to be defined for each collection during collection registration. In such local contexts the concepts defining values semantics are defined as sub-concepts of the respective mediator concepts. Generally, a redefinition of feature evaluation function for a local context may be required. The proposed structuring of type attribute semantics definition gives significant economy in development of value conversion functions reconciling different contexts.

The paper is structured as follows. After brief characterization of the related work, a short introduction into the technique of metatype specification in SYNTHESIS is given. The intention of this introduction is to make further examples in the text better readable. Main contribution of the paper is contained in sections presenting an approach for well-structured context definitions in mediators and collections and their application to value conflicts reconciliation between the mediator and a collection contexts. The conclusion summarizes the results of the paper.

2. Related Work

Similar issues of value conflicts reconciliation were treated in the COIN project [6] that addressed problems of logical connectivity (the ability to exchange meaningful information) among disparate data sources. A multiplicity of both sources and receivers was considered. Data semantics in different contexts need to be defined applying specific structuring of specifications separating schema and context definitions.

In essence, each information source in the architecture is tagged with a context; a context being a set of axioms which describe certain assumptions about the data. For instance, if *A* is a database containing information on a company's finances and the dates in which those finances were recorded, then the context associated with database *A* would contain any underlying information needed to properly interpret those financial figures and dates, such as the scale factor, the currency, and the format of the date. Using this context information, the COIN architecture is able to resolve semantic conflicts.

Besides schema definition, the context axioms for the source and conversion functions are defined. Type definitions in a schema include signatures for the modifiers in the system. A relationship between type instances and modifiers that characterize representation of the type instances is established. Modifiers have varying interpretations depending on the context.

Context axioms are a set of definitions for the modifiers of each type given in the schema. The values returned by a modifier depend on a given

context in which they are created. Modifiers can also be defined intentionally. Conversion functions define how the value of a given semantic object can be derived in the current context, given that its value is known with respect to a different context. Query mediation within the COIN framework is accomplished using a top-down logic evaluation strategy resulting in a query plan which describes what sources and conversion operations are needed to mediate data exchanges at each step.

An approach presented in this paper differs in its orientation on the LAV mediators and object paradigm.

In [7] object attributes were treated as type definitions. Association metatypes establishing properties of association types were introduced. By means of an instance section of an association metatype meta-attributes of an association type may be defined. Similar approach to interpret meta-attributes in the object model is used in this paper.

3. Technique of Metatype Specifications in SYNTHESIS

The fundamental concept of the SYNTHESIS object model [8] is an abstract value. All data components in SYNTHESIS are abstract values. Abstract values are instances of abstract data types (ADT). Objects specialize abstract values with new properties - an ability to have a unique identifier and a potential ability to belong to some class (classes).

A type in the language is treated as a first-class value. Such values may be defined in specifications, on typing of variables, may be produced by type functions and type expressions. A class in SYNTHESIS is treated as a subtype of the set type. Class and type specifications are abstract and completely separated of their implementations. A number of various correct implementations (concretizations) may correspond to such specifications.

All operations over typed data in the SYNTHESIS language are represented by functions. Functions are given by predicative specifications that are given by mixed pre- and post-conditions. To express conditions related to changing of the information resource states, an ability is needed to denote variables expressing a resource state before and after function execution. For that a reference to a value of a variable after execution of a function is denoted by primed variables.

Predicative specifications are expressed using formulae of the SYNTHESIS object calculus. To specify formulae a variant of a typed (multisorted) first order predicate logic language is used. Every predicate, function, constant and variable in formulae is typed. Predicates in formulae correspond to classes, collections and functions. Variables, constants and function designators are used as terms. Each term has a well-defined type.

In the SYNTHESIS language the type specifications are syntactically represented by frames, their attributes - by slots of the frames. Additional information related to attributes can be included into metaslots. Syntactically frames are included into figure brackets { and }, slots are represented as pairs *<slot name> : <slot value>* (a frame can be used as a slot value), slots in a frame are separated by semi-colons. Metaslots (that are represented by frames) are written immediately after the slots to which they are related.

The multilevel type system of the SYNTHESIS language is organized as follows. On the level of types the type objects are located providing for definition of concrete and generic types. On the second level (the level of "types of types") the metatype objects are located that include as their instances the types of the first level. On the third level the metatypes objects are located that include the metatypes of the second level as their instances, and so forth.

Thus the multilevel type system sets a classification relationship on types that is orthogonal to the subtype relationship. Metatypes behave like (meta)classes that in their turn are organized as follows. A class specification combines information about two kinds of objects: about a class as an object itself and about objects - instances of the class.

Generally metaclasses provide for introducing of generic concepts and of common attributes (or of their categories) for similar classes, for introducing of common consistency constraints and deductive rules for such classes and their attributes. Metaclasses provide for proper grouping of application domain information and for proper differentiation of various application domains.

In the language the attribute specifications of objects may be treated in their turn also as types of association objects establishing a correspondence between a set of objects in an association domain and a set of objects in an association range. Thus a specification of a type attribute may be considered as a specification of an association type.

Treating of an object attribute as an association type motivates an introduction of association metatypes establishing properties of association types. It is said that an object attribute (as an association type) belongs to a particular attribute category that is explicitly introduced by an association metatype.

By means of an instance section of an association metatype additional attributes of an association type may be defined as it was explained above, thus "attributes of attributes" can be defined (e.g., for an attribute price attributes of this type can be introduced, such as price status (e.g., season price), currency). If for a particular type attribute a metaslot is declared and it is defined that the attribute belongs to a certain attribute category, (categories) then a merge of the specifications given in the metaslot and in an instance sections of the respective association metatypes is formed.

4. Definition of Contexts

The proposed approach to value reconciliation between local collection and mediator contexts consists in applying well-structured type attribute specifications. Conflicts in data representation in different contexts are frequently met due to the simplifications of attribute type schemas. Type *string* or *integer* may be used to represent values having different meaning (e.g., salary, budget, date, currency may have different semantics and formats in different application contexts). A certain context may impose specific assumptions about data that may not be explicitly defined in a schema.

To provide attribute specifications in each context with more value semantics a concept defining value semantics in terms of *features* characterizing the value should be defined. This concept is defined as an association metatype with an instance type containing value features specifications.

For example, for attributes meaning *money amount* an association metatype can be defined to specify that such attribute values will be characterized by *currency* and *scale factor* features. The instance type may contain also a function that makes possible to assign to these features values specific for a concrete context.

Further, for such class of values (*money amount*) a type is to be specified having an attribute that is declared as an instance of the association metatype introduced. This type includes also a generic function converting values between different contexts. This function is controlled by the respective features values in the contexts involved. Each type attribute having the respective value semantics is to be typed with such type definition.

The specification example in SYNTHESIS follows. We start with specifying of the mediator context.

```
{Rate;
    in: type;
    fromCurrency: string;
    toCurrency: string;
    exchangeRate: float
}
{rate;
    in: class;
    instance_section: Rate
}

{Currencies;
    in: type;
    country: string;
```

```
    currency: string
}
{currencies;
    in: class;
    instance_section: Currencies
}

{MoneyFeatures;
    in: metatype, association;

    instance_section: {MF;
      in: type;
      currency: string;
      scaleFactor: float;

      context_curr: {in: function;
        params: {+country/string};
        {{currencies(z/Currencies) & z.country = country &
          this.currency’ = z.currency &
          ((z.currency = 'JPY' & this.scaleFactor’ = 1000)|
          (z.currency <> 'JPY' & this.scaleFactor’ = 1))}}
      }
    }
}

{MoneyAmount;
    in: type;
    sum: float;
      metaslot
        in: MoneyFeatures
      end;
    currency_convert: {in: function;
      params: {+T/type, +newMA/T, -returns/T};
      {{(meet(MoneyAmount,T) = MoneyAmount) &
        ex x/Rate (rate(x) &
        x.fromCurrency = this.sum.currency &
        x.toCurrency = newMA.sum.currency &
        returns.sum’= this.sum * x.exchangeRate *
        newMA.sum.scaleFactor / this.sum.scaleFactor)}}
    }
}

{Company;
    in: type;
    name: string;
    revenue: MoneyAmount;
```

```
    country: string
}
{company;
    in: class;
    instance_section: Company
}
```

Association metatype *MoneyFeatures* defines the concept of money amount notion for the mediator attributes. In this specification the instance type is defined containing attributes (features) *currency* and *scaleFactor* and the *context_curr* function. This function is intended for the *currency* and *scaleFactor* value assignment in the mediator context. *MoneyAmount* type attribute *sum* is defined as an instance of metatype *MoneyFeatures* acquiring thus its semantics. Method *currency_convert* specifies generic function converting values between contexts

Mediator type *Company* has three attributes: *name*, *country* and *revenue*. The attribute *revenue* is typed with the *MoneyAmount* type. The type *Rate* and class *rate* are auxiliary data specifications used in *currency_convert* function for conversion of currencies.

Specification of local collection is formed during collection registration at the mediator. Process of registration is defined in [2]. Here we consider additional actions required for value semantics definition and reconciliation. Association metatype *LocalMoneyFeatures* is defined to represent semantics of money amount similarly to that of the respective mediator metatype. Instance type of local metatype is defined as a subtype of the instance type of the mediator metatype *MoneyFeature*. Note that in the local context the *context_curr* function is redefined to reflect specificity of the local context.

```
{LocalMoneyFeatures;
    in: metatype, association;
    instance_section: {LMF;
      in: type;
      supertype: MF;

      context_curr: {in: function;
        params: {+country/string};
        {{currencies(z/Currencies) & z.country = country &
          this.currency’ = z.currency &
          ((z.currency = 'JPY' & this.scaleFactor’ = 100) |
          (z.currency = 'RUR' & this.scaleFactor’ = 30) |
          (z.currency <> 'JPY' & z.currency <> 'RUR' &
          this.scaleFactor’ = 1))}}
      }
    }
```

```
}

{MoneyAmt;
    in: type;
    supertype: moneyAmount;
    sum: float;
      metaslot
        in: LocalMoneyFeatures
      end
}

{Organization;
    in: type;
    name: string;
    revenue: MoneyAmt;
    state: string
}
{organization;
    in: class;
    instance_section: Organization
}
```

Local collection specifications in the example include type *Organization* that is ontologically relevant to type *Company* of the mediator. Semantic meaning of a schema specification element is provided by linking this element to ontological concept defined for a given context (of a collection or a mediator). If an element of specification is semantically related to a concept then it becomes an instance of the ontological class corresponding to this concept. It is used for finding of ontologically relevant elements of specifications. Ontological relevance of specification elements in different contexts is identified during collection registration.

Attributes *name*, *revenue* and *state* of type *Organization* are semantically relevant to the respective attributes of *Company*. *MoneyAmt* type is registered as a subtype of *MoneyAmount*. Its attribute *sum* is redefined so that it acquires semantics from *LocalMoneyFeature* concept that is valid in the collection context. The attribute *revenue* of type *Organization* is defined with *MoneyAmt* type.

Conversion function *currency_convert* is constructed so that its parameter is an instance of *T* type. *T* also is passed as a parameter and should satisfy the precondition: *meet(MoneyAmount,T) = MoneyAmount*. According to this type expression [10], *MoneyAmount* and its subtypes are admissible input types. The value of this type contains also meta-attributes values that have been evaluated for the respective context. The function returns reconciled value of the *T* type. Another instance involved is *this* instance that is of type

MoneyAmount if the function is called from the mediator context or of type *MoneyAmt* if the function is called from a collection context. The function converts *this.sum* from the respective context into the *returns.sum* in the opposite context. Using auxiliary class *rate* the function finds exchange rate for the source and target currencies. Then the function calculates *returns.sum* from *this.sum* modifying it with the exchange rate found applying different scale factors of attribute in different contexts.

5. Application of Context Definitions to Attribute Value Reconciliation

We consider both way of value reconciliation – from a collection context to the mediator context and in a reverse order.

During registration of the local type *Organization*, this type becomes ontologically related to the mediator type *Company* and the common reduct for these types *R_Company_Organization* is specified. This reduct is such subspecification of *Company* type that there exists a reduct of *Organization* type that refines this reduct of *Company*. In our example the common reduct looks simply as follows:

```
{R_Company_Organization;
    in: reduct;
      metaslot
        of: Company;
        reduct: CR_Company_Organization;
        taking: {name, revenue, country}
      end
}
```

A slot *of* refers to the reduced mediator type *Company*. A list of attributes of the reduced type in the slot *taking* contains names of its attributes that are to be included into the reduct. In our case the reduct includes all attributes of *Company*. Concretizing reduct *CR_Company_Organization* is intended to specify how *R_Company_Organization* instance should be interpreted by an instance of the *Organization* type:

```
{CR_Company_Organization;
    in: c_reduct;
      metaslot
        of: Organization;
        reduct: R_Company_Organization;
        taking: {name, revenue, state}
```

```
      end
    simulating: {
      R_Company_Organization.name ~
          CR_Company_Organization.name;
      R_Company_Organization.country ~
          CR_Company_Organization.state;
      R_Company_Organization.revenue ~ cvt_revenue
    };
    cvt_revenue:{in: function;
      params: {+ext/CR_Company_Organization,
               -returns/MoneyAmount};
      {{ex o/Organization (o/CR_Company_Organization = ext &
        new(m/MoneyAmount) & m.sum.context_curr(o.state) &
        returns'=o.revenue.currency_convert(MoneyAmount,m)}}
    }
}
```

Slot *reduct* refers to the respective common reduct. Predicate *simulating* shows how the common reduct state is interpreted by an instance of the collection. E.g., it defines that attribute *name* of reduct *R_Company_Organization* is refined by attribute *name* of concretizing reduct *CR_Company_Organization* and value of this common reduct attribute is to be taken from the concretizing reduct attribute value. It also defines that attribute *revenue* of the common reduct *R_Company_Organization* is modeled by the function *cvt_revenue* resolving the value conflict. This function creates an instance of *MoneyAmount* type, contextualizes this instance in the mediator context and calls *currency_convert* function as a method of *revenue* attribute in *organization* class object identified by the concretizing reduct value. Note that the attribute *o.revenue* is of type *MoneyAmt* and exists in the collection context.

Now we consider a reverse way of attribute value reconciliation – starting with a value in the mediator context and delivering a value in a collection context. Assume that as a part of the mediator query we get

```
country = 'Japan' & revenue = 10000 & company(comp)
```

The problem is how to rewrite the constant *10000* that has the mediator context semantics to a collection context applying the same *currency_convert* function. To do that two money amount value instances need to be created – the first one (*m*) is in the mediator context that actually is *10000* extended with the mediator context money amount features and the second one is in the local context (*ml*) having the respective collection semantics. The following is the call of *currency_convert* together with the money amount instances creation and their semantics evaluation in the proper contexts.

```
new(m/MoneyAmount) & m.sum.context_curr('Japan')
& m.sum == 10000 & new(ml/MoneyAmt) &
ml.sum.context_curr('Japan') &
returns' = m.sum.currency_convert(MoneyAmt, ml)
```

6. Conclusion

The paper presents a method for value reconciliation in mediators of heterogeneous information collections applying well-structured specifications. Such value reconciliation specifications are formed during registration of information collection at a LAV mediator. According to the proposed approach, value semantics contexts should be defined in the mediator and in a collection. The contexts are formed by concepts defining value semantics of each kind of specific value in terms of characterizing the value features. For proper features evaluation in a concrete context a specific function is to be included as a part of the value semantic concept. Technically such concept definitions are specified as association metatypes and extend the ontological specifications of the mediator and a collection. In such context for each value kind a type is to be specified that includes a generic function converting values from a collection context into the mediator context (and/or back). Each type attribute having the respective value semantics is to be typed with such type definition. Such structuring of type attribute semantics definition gives significant economy in development of value conversion functions needed to reconcile values between the mediator and collection contexts.

References

[1] Briukhov, D.O., Kalinichenko, L.A. Component-based information systems development tool supporting the SYNTHESIS design method. In Proceedings of the East European Symposium on Advances in Databases and Information Systems (ADBIS'98), Springer, LNCS No. 1475, 1998.

[2] Briukhov, D.O., Kalinichenko, L.A., Skvortsov, N.A. Information sources registration at a subject mediator as compositional development. In Proceedings of the Fifth East European Symposium on Advances in Databases and Information Systems (ADBIS'01), Springer-Verlag, 2001, pp. 70-83.

[3] Duschka, O.,and Genesereth, M. Answering Queries Using Recursive Views. In Principles Of Database Systems (PODS), 1997.

[4] Friedman, M., Levy, A. and Millstein, T. Navigational Plans for Data Integration, In Sixteenth National Conference on Artificial Intelligence (AAAI-99), Orlando, Florida, 1999.

[5] Garcia-Molina, H., Hammer, J., Ireland, K., Papakostantinou, Y., Ullman, J. and Widom J. The Tsimmis Approach to Mediation: Data Models and Languages. Journal of Inteligient Information Systems, 1997.

[6] Goh, C., Bressan, S., Lee, T., Madnick, S., Siegel, M. A Procedure for the Context Mediation of Queries to Disparate Sources. International Logic Programming Symposium, 1997.

[7] Kalinichenko, L.A. Rule-based concept definitions intended for reconciliation of semantic conflicts in the interoperable information systems. In Proceedings of the Second International Baltic Workshop on DB and IS, Tallinn, June 1996.

[8] Kalinichenko, L. A. SYNTHESIS: the language for description, design and programming of the heterogeneous interoperable information resource environment. Institute for Problems of informatics, Russian Academy of Sciences, Moscow, 1995.

[9] Kalinichenko, L. A. Integration of Heterogeneous Semistructured Data Models in the Canonical One. In Proceedings of the First All-Russian Conference on Digital Libraries, St. Petersburg, 1999.

[10] Kalinichenko, L.A. Compositional Specification Calculus for Information Systems Development. In Proceedings of the East-European Symposium on Advances in Databases and Information Systems (ADBIS'99), Springer-Verlag, LNCS, 1999.

[11] Levy, A., Rajaraman, A. and Ordille, J. Querying Heterogeneous Information Sources using Source Descriptions. In Proceedings of the 22nd Conference on Very Large Databases, 1996, pp. 251-262.

[12] Sheth, A. and Larson, J. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Database. ACM Computing Surveys, 1990.

[13] Subrahmanian, V. S. Hermes: a Heterogeneous Reasoning and Mediator System. http://www.cs.umd.edu/projects/hermes/publications/postscripts/tois.ps

[14] Wiederhold, G. Mediators in the Architecture of Future Information Systems. IEEE Computer, 1992.