

# Information Modeling in Multidatabase Systems: Beyond Data Modeling

(Invited Paper)

Amit Sheth  
Bellcore  
444 Hoes Lane  
Piscataway NJ 08854-4182  
*amit@ctt.bellcore.com*

Leonid Kalinichenko  
Russian Academy of Sc.  
Vavilova 30/6  
Moscow, V-334, 117900 Russia  
*leonidk@ipian15.ipian.msk.su*

## Abstract

We consider the information modeling issues in environments that range from *multidatabase manipulations* supporting direct DBMS level interfaces to multiple database, to *multisystem applications* involving activities among multiple application systems. Information modeling needs in such environments go far beyond the use of traditional data models to develop database schemas or application views. Besides modeling data managed by individual component systems and databases, we also need to model operations and activities, as well as interdependencies and constraints that span across the systems. Because of significant interplay between these different types of information during information management activities such as query processing and transaction management, a uniform framework for modeling, and preferably a single language for specifying, all these types of information is highly desirable. For an example multisystem application, we demonstrate important aspects of such a uniform and comprehensive modeling in the SYNTHESIS language.

## 1 Introduction

The multidatabase system environments that we consider range from *multisystem applications* to *multidatabase manipulation*. The former includes supporting applications that perform operations on mul-

multiple existing, autonomous, and heterogeneous component application systems. A component application system consists of application program(s) and its own database(s), and typically can be accessed only through its application level interface. A multidatabase manipulation involves direct DBMS level access by a multidatabase application to the databases managed by the (possibly heterogeneous) DBMSs.

To support applications and effective management of information in multidatabase systems characterized above, we need to go significantly beyond the traditional database modeling that involves modeling of structure, and sometimes semantics (i.e., meaning and use), of data objects that are stored in a database or used by an application. Additional types of information that should be modeled include modeling of multisystem operations (also called multisystem activities or work flows), and of multidatabase consistency requirements specified as consistency constraints among related data that are managed in multiple databases.

In Section 2, we characterize the three information management issues that have typically been addressed separately in the literature. We discuss some of the significant interactions among the three information management issues, and point to the advantages that may result from their comprehensive and uniform modeling. We then sketch an information repository that can support such a modeling. In Section 3, we use a simple multisystem application and the SYNTHESIS language to demonstrate one representation framework to comprehensively and uniformly model different types of information. More emphasis is given to the operation (transaction) execution semantics and consistency constraints – the types of information that has often not been modeled in the past. Section 4 provides a brief conclusion.

**Proceedings of the First Intl. Conf. on  
Information and Knowledge Management,  
Baltimore, MD, USA, November 1992.**

## 2 Three Dimensions of Information Management

We present a classification of information management issues in multidatabase system environments along three essential dimensions: multidatabase information modeling, multidatabase operation interdependencies, and multidatabase consistency constraints (see Figure 1). Almost all research to date has focused on one dimension at a time. However, we believe that issues along each dimension have significant interactions with information along other dimensions. Furthermore, a compatible or uniform representation of information along each of these three dimensions can help to beneficially exploit the interactions among them and to provide a more comprehensive framework for managing information and supporting multisystem applications in a multidatabase environment.

### Multidatabase Information Modeling

Multidatabase information modeling can be seen as an extension of the recent work on federated database schema architecture [SL90], information modeling (including conceptual data modeling for traditional databases and behavior/operation modeling in object oriented context), schema analysis and schema integration, and techniques to deal with semantic heterogeneity in multidatabase systems [S91]. Types of information about individual information systems that may be modeled along this dimension include:

- schemas that may consist of data structure definitions using data models needed for multidatabase manipulation and/or behavior/operation modeling needed for multisystem applications
- other meta-data such as dictionary and directory information and other information stored in a traditional information repository,
- meta-meta-data such as data modeling guidelines and data modeling constraints,
- data and application semantics including common sense and background knowledge and ontology, and other ways of capturing semantics (e.g., rules).

Following the modeling of individual information systems, a multidatabase information model needs to specify semantic similarities and schematic differences among information managed by different systems [SK92].

### Multidatabase Operation Interdependencies

This dimension presents an operation-centric or transaction-centric view of information management. It captures multisystem application semantics related to dependencies among multisystem application's operations or transactions performed by multiple component systems or dependencies among subtransactions of a multidatabase transaction. This dimension is related to research in relaxed or extended transaction models [E92] and frameworks [CR91]. These can be used to support multisystem work flows [DHL91][ANRS92]. In addition, it may include semantics of operations/transactions such as commutativity or one sided commutativity of operations [BR87][KRST92] or the levels of inconsistencies that can be tolerated by an operation. Managing a work flow involves specification and support of dependencies among operations performed by different systems and databases to support a multisystem application.

### Multidatabase Dependencies and Consistency

This dimension presents a data-centric view of information management. Specifically, it captures specification of semantically related data stored in different databases and their consistency in a multisystem application and multidatabase context. Its key component is specifying and enforcing consistency of inter-related data stored in databases managed by different DBMSs and/or different application systems. Recent research on active databases [DHL90] and research on interdependent data [RSK91] help in addressing this issue (also see [WQ87]). The framework for modeling interdependent data uses data dependency descriptors to capture structural dependency among related data stored in different databases, their consistency requirements involving both temporal and data state components, and procedures to restore consistency of related data.

#### 2.1 Interactions

Issues along these three dimensions are not fully orthogonal. We feel that at a conceptual level, it helps to consider each of them separately. This allows us to distinctly address the distribution, heterogeneity, and autonomy issues along each dimension. However, it is critical to understand and exploit the interactions among information along different dimensions to develop a comprehensive information management sys-

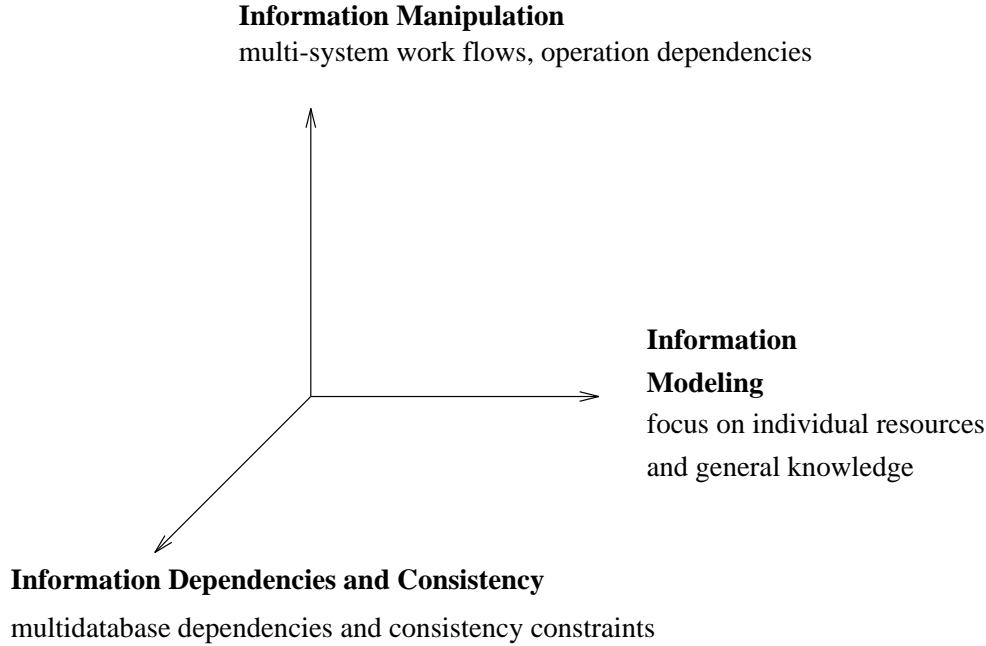


Figure 1: Three dimensions of information management in a multidatabase environment

tem. Some examples of interactions and relationships between the issues on the three dimensions follow:

- Modeling of data in information modeling can be extended to include modeling of operations and transactions (subtransactions, atomic transactions, multidatabase transactions). In fact an object hierarchy may include data objects and operation/transaction objects.
- Multidatabase consistency constraints and inter-transaction (intra-multidatabase transaction) dependencies can be considered to be aspects of constraints in information modeling. In Carnot[C91], all constraints together are termed as a declarative resource constraint base. Uniform specifications of correctness criteria used by various transaction models and consistency requirements of applications have been suggested in [KMS91] and that of database consistency constraints and transaction correctness properties have been investigated in [RC91]. Rule based specifications and triggers, such as E-C-A rules [DHL90] can be used to specify aspects of operation dependencies as well as data consistency requirements.

- Information along the information modeling dimension can be used for query processing, optimization, semantic enrichment, and retrieving approximate answers. Carnot[C91] performs semantic enrichment to retrieve objects related to the original query.
- Multisystem/multidatabase transaction management (e.g., polytransactions [SRK92]) may involve using information along the multidatabase operation interdependencies and operation semantics along the multidatabase manipulation dimension as well as information such as data dependency descriptors along the multidatabase dependency and consistency constraints dimension. Alternatively, polytransactions can be seen as augmenting traditional transactions or other relaxed multidatabase transactions to also enforce multidatabase consistency requirements.

## 2.2 An Information Repository

We propose that an information repository should maintain all these types of information, and that having a uniform representation would help in designing such a repository. A possible structure of an information repository that may hold information along all

three dimensions is shown in Figure 2. The next section addresses the issue of uniform representation.

Information is modeled using three types of objects: data, operations, and constraints. These types correspond to the information management dimension shown in Figure 1. Objects of each type have been organized in a three level classification hierarchy. Generalization-specialization is further used at each classification level. Each level is specialized into the objects of the three types. The meta-meta level provides a description of one or more models for each type of object. It can include one or more of the following:

- data models for modeling database schemas
- activity or relaxed transaction models for modeling activities, collections of operations, or multidatabase transactions<sup>1</sup>, and
- data dependency and consistency specification models (e.g., the data dependency descriptors requiring three specification components of dependency, consistency, and restoration [RSK91] for specifying multidatabase dependencies and consistency constraints).

Various models at the meta-meta level themselves can be organized into additional class (generalization) hierarchies not shown in this figure. For example, various relaxed transaction models can be defined in a class hierarchy to allow sharing of modeling features and interoperability of relaxed transactions defined at the meta-level.

Models defined as subclasses will inherit attributes and behavior from their transitive superclasses. Thus common features of relaxed transaction models and their interrelationships may be explicitly and homogeneously shown. Specific transactions are represented as instances of an appropriate class. For instance, we may assume that deferred transaction model [DHL91] is specified as a subclass of a nested transaction model, which in turn is a subclass of a more general transaction model. All relevant inter-transaction dependencies defined in terms of effects of transactions on other transactions are specified by the methods of these classes.

Each meta level object is described according to a relevant model described at the meta-meta level. Following types of objects are modeled at this level:

- individual database schemas (intensional aspects of the objects) are modeled as meta-data objects

---

<sup>1</sup>A framework such as ACTA [CR91] to provide building blocks for such models.

- various multisystem applications, activities, or multidatabase transactions are modeled as meta-operation objects, and
- various multidatabase and multi-system dependencies and consistency constraints are modeled as meta-constraint objects.

At the object level, we have individual data object extensions, codes for individual methods and transactions, and descriptions or implementations of individual components of constraint specifications.

### 3 Uniform Representation

A uniform representation can facilitate to understand and exploit interactions among the issues modeled along the three dimensions and develop a comprehensive information management strategy. In this section, we use the SYNTHESIS language [K91] to demonstrate the feasibility of using one representation system to uniformly model information along all three dimensions. A simple hotel reservation system is modeled. We model only the objects at the meta level of the classification hierarchy.

#### 3.1 An example of a simple hotel reservation system

The computerized hotel reservation system of a franchise hotel consists of local offices (LOs) located at each franchise location and the central reservation office (CO). A customer or travel agent can directly contact an LO or call the CO using its 800 number to either make a new reservation, change the existing reservation, or cancel the existing reservation. The LO has the master copy of the reservation records and availability of rooms at that location. The CO keeps a copy of the records of each LO. This copy can be up-to 24 hours old. The CO calls up each LO every night to perform the following operations.

1. Exchange with the LO the respective changes in reservation records during the last period.
2. Get from the LO a quota of 25% of the remaining vacancies. This quota may be used by the CO during the next day to satisfy new reservation requests.

The database state at the CO is considered to become abnormal if:

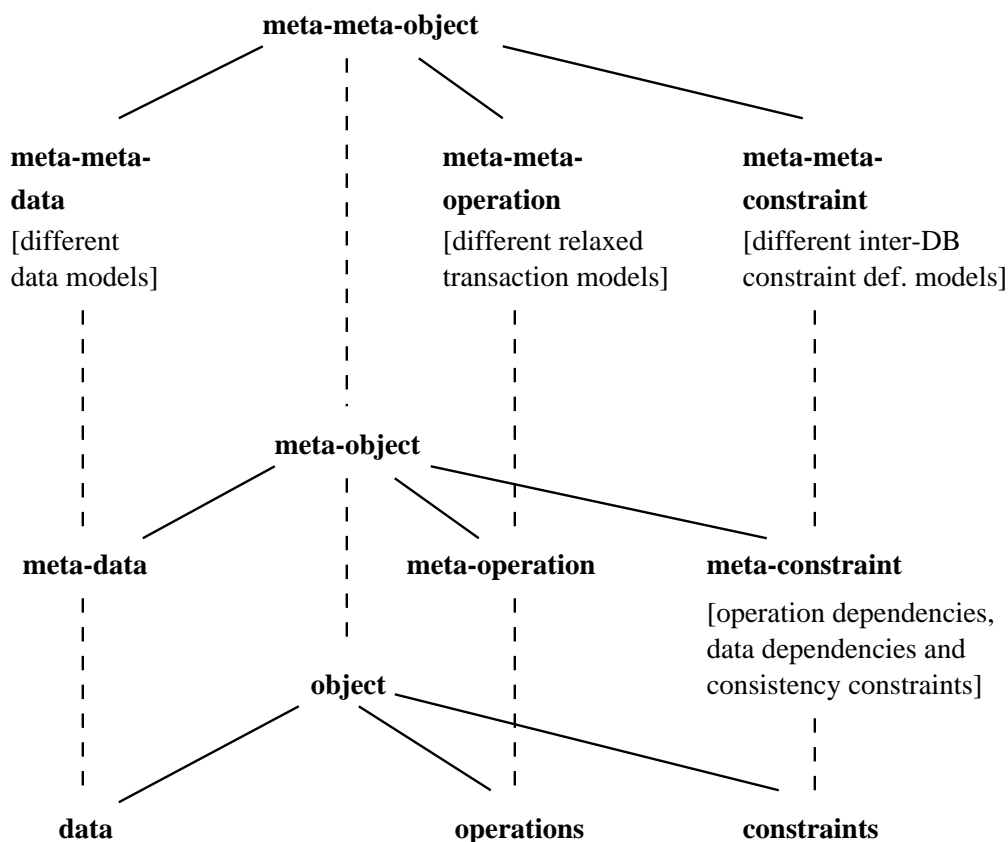


Figure 2: An information repository structure covering the three dimensions

1. a quota for a hotel managed by the CO is exhausted;
2. a quota for a hotel managed by the CO exceeds the given 25% limit because of the cancellation of previous reservations.

In such cases the CO communicates with the LO immediately (in addition to the once-a-day normal exchange explained above), and exchanges information using the following rules:

1. If the quota for some LO is exhausted, the CO should communicate with this LO to get an additional quota only if the total amount of the CO reservations exceed 70% of the sum of all quotas (this sum is taken for all LOs served by the CO). It is expected that the CO can provide reasonable service if the CO still has sufficient amounts of quotas for other LOs. 70% is a heuristic threshold.
2. If the actual number of rooms available for some LO exceeds its quota (e.g., the extra quota may

be obtained due to cancellations of previous reservations) and the total amount of CO reservations is below 50% of the sum of all quotas, the CO should communicate with the LOs to release a certain amount of quota. This heuristics ensures that the quota assigned to the CO does not far exceed its ability of fill vacancies.

Our example ignores the situation when the quota at an LO is exhausted but the CO still has a vacancy for that LO.

Using this simple example, we now demonstrate a uniform representation of information along all three dimensions (Information Modeling, Operation Interdependence, Consistency Constraints) in the SYNTHESIS language.

### 3.2 Overview of the relevant SYNTHESIS language features

SYNTHESIS [K91] was designed as a *multipurpose language* to homogeneously describe heterogeneous information. Examples of the types of information that

can be modeled include formalized description of the application domains, conceptual design of pre-existing information resources, the application problem specification. In this paper we discuss only those features of the SYNTHESIS language that are necessary to demonstrate types of information occurring in our example.

The SYNTHESIS language uses *frames* to describe any entity, including the entities of the language itself, such as types, classes, functions, and assertions. A frame may be used as a structured symbolic model of some entity or some concept. The built-in slot *self* contains the frame's unique identifier. It is assigned by the system when the frame is created and is kept permanent during the life of the frame.

Syntactically, a frame is always represented in brackets { and }. Slots are separated by semicolons. A slot name and its values are separated by a colon. The values of a slot are separated by commas. Acceptable slot values include an atomic value, a frame, a collection of rules of logic programming language, and a set of values.

Each frame may be declared to belong to a class or a metaclass. Such frame membership is denoted by a slot in :< class or metaclass name list >. Examples of built-in metaclasses are: *class*, *metaclass*, *assertion*, *transaction*, and *function*.

Syntactically, each functional attribute of a class (such attributes are considered to be the methods) is defined by the description of a function:

```
< function description > ::=
    { < function identifier >;
      in : function;
      [params : { < formal parameter list > };]
      [< implementation >] }
< formal parameter identifier > ::=
    < parameter sort symbol > < typed variable >
< parameter sort symbol > ::= - | + | < empty >,
    where
    + specifies input parameter;
    - specifies output parameter; and
    < empty > specifies input & output parameter.
```

Implementation of a function may be given by either a logic program, by a predicative specification or by a program written in some programming language. A predicative specification is used to state mixed pre/post conditions defining the function. To describe a state transition, it is necessary to distinctly denote the variables that define the state before and after the function execution. In our syntax, an apostrophe is used to identify the variable referring to the

state after function execution.

Assertions, logic programs, and formulae appearing in the predicative specifications below are described as follows. A variant of the multisorted first order predicate logic language is used in SYNTHESIS [KZ91]. Each predicate, function, variable and constant in the formulae is typed. In simplified form formulae are either *atoms* or appear as:

$$\begin{aligned}
 &w_1 \ \& \ w_2 \ (w_1 \ \text{and} \ w_2) \\
 &w_1 \ | \ w_2 \ (w_1 \ \text{or} \ w_2) \\
 &\neg w_2 \ (\text{not } w_2) \\
 &w_1 \rightarrow w_2 \ (\text{if } w_1 \ \text{then } w_2) \\
 &\exists x/t \ (w) \ (\text{for some } x \ \text{of type } t, \ w) \\
 &\forall x/t \ (w) \ (\text{for all } x \ \text{of type } t, \ w)
 \end{aligned}$$

where  $w$ ,  $w_1$ ,  $w_2$  are formulae.

Existential and universal quantifiers are denoted by *ex* and *all*, respectively. Built-in predicates are used to simplify representation of the formulae, e.g.:

- *added*(< variable >, < class name >) - is true after the function execution if the variable defines a new object created as an instance of the given class;

- *removed*(< variable >, < class name >) - is true after the function execution if the variable defines an object removed from the given class.

Predicative specification of an operation involves defining properties and relationships that should be satisfied by the state transitions caused by execution of the operation. This involves using the predicates relating values of state variables before and after the operation execution (expressing mixed pre- and post-conditions). Thus, a predicative transaction specification uses predicates that express what properties the state transitions resulting for the transaction execution have, in terms of the properties of, and the relationships between the input and the output states, of the state transition. A complete predicative specification of a (multidatabase) transaction model involves considering all state transactions caused by the primitives (such as *Begin*, *Spawn*, *Commit*, *Abort*) of the transaction model.

We specify the transaction models in an object-oriented style. Typed first order logic is used to represent consistency constraints (sometimes called invariants) and predicative specification is used for operations. Primitives are applied to transactions interpreted as objects. The object-orientation allows specifying a transaction class hierarchy where each class represents a specific transaction model. Models defined as subclasses will inherit attributes and behavior from their transitive superclasses. Thus common features of multitransaction models and their interrelationship may be explicitly shown. Specific transac-

tions are represented as instances of an appropriate class.

### 3.3 Specification of the hotel reservation system in SYNTHESIS

Definitions of data objects and operation/transaction objects are given next and are self explanatory. Thereafter, we discuss specification of constraints and an execution model for data interdependencies in more detail.

#### Information modeling in an object-oriented style

```
{hotel;
in: class;
hotel_name: string;
quota: integer;
actual_resrv: integer;
};

{reservation;
in: class;
where: hotel;
for_whom: person;
for_period: time;
};
```

#### Operation/Transaction Modeling

```
{central_op;
in: transaction;
r: reservation;
book: {in: function;
      params: {+person, +hotel, +period, -r};
      {r.where='hotel & r.for_whom'=person &
r.for_period'=period & added(r,reservation) &
hotel.actual_resrv'=hotel.actual_resrv-1}}};
cancel: {in: function;
        params: {+r};
        {r.where.actual_resrv'=
          r.where.actual_resrv+1 &
removed(r,reservation)}}
};
```

#### 3.3.1 Consistency Constraint Modeling

We now discuss declarative specification of data interdependency constraints and an execution model to enforce them.

In SYNTHESIS, the *script* facility is used for describing a collection of activities. The script model of SYNTHESIS is based on the high level Petri nets. Scripts are defined as classes, each instance of a script class corresponds to a particular activation of an activity defined by the script. An activity may have some or all of the ACID properties of a traditional transaction. As an object such instance is characterized by an object identifier and a collection of the script attribute values. A script includes description of *states* that may be connected by *transition arcs*. The definition of a state may include assertions that should be satisfied in the state. Each transition arc connects the transition *input states* (specified by the from slot of a script) with the *output states* (specified by the to slot of a script). The transition arc is also defined by the *transition condition* and the *transition action*. The transition takes place if all its input states are active, assertions corresponding to the input states are satisfied and the transition condition is satisfied. After the transition action all input states of the transition are deactivated and all output states are activated.

As mentioned earlier, we propose to model a transaction class hierarchy. For example, the class specifying the deferred transaction model is a subclass of the class specifying the nested transaction model. Furthermore, execution model of each transaction model is specified as its subclass of type script class.

Only one instance of the script for each top level activity (in our case for each CO activity) should exist. This instance, besides representing a script, will also represent a top transaction in the deferred transaction class. Subtransaction instances may be created during the life of a top transaction as its children. We discuss execution model of the deferred transaction model below.

By declaring a script as a subclass of the deferred transaction class, we can use the methods associated with this class. The methods of a class specifying a transaction model define primitives of the corresponding transaction model. In the example we use two primitives of the deferred transaction model: *deferred\_spawn* and *cycle*.

*Deferred\_spawn* causes creation of a subtransaction that will be spawned and executed as a deferred subtransaction. The cycled deferred activity of subtransactions is organized using two sets of deferred subtransactions - *current\_wave* and *next\_wave*. Current wave contains subtransactions that were made ready during the previous cycle (the concept of cycle is defined below) for concurrent execution during this cycle. The next wave contains subtransactions that were created

by *deferred\_spawn* primitives during the execution of the top transaction or of the subtransactions of the current wave.

The *cycle* primitive is called in a state when the current wave is empty (all subtransactions of the current wave were executed). Execution of this primitive results in moving of all subtransactions from the next wave to the current wave and in setting the next wave set to empty. Subsequently concurrent execution of the current wave of subtransactions is started.

The execution of *deferred subtransaction* [DHL91] is delayed until the *cycle-0\_end*, which denotes end of the top transaction *T*. If more than one deferred subtransactions are created before *T* reaches its *cycle-0\_end*, then all these subtransactions are started as concurrent subtransactions in *cycle-1*. If the processing of subtransactions in *cycle-1* causes more deferred transactions to be created, the latter are started when all subtransactions in *cycle-1* have finished and are started as concurrent subtransactions of *T* in *cycle-2*. If no more deferred subtransactions are created during the current cycle, *T* can commit.

We now specify the consistency constraints for our example and their enforcement. A comment field of each specification explains that specification.

```
{CO_state_restoration_activity;
in: script;
superclass: deferred_transaction ;
initial: start;
tr1: deferred_transaction;
tr2: deferred_transaction;
states:
```

```
  {DDD1;
  assertions:
  {{ex hotel_name (available_reserv=0 &
    hotel(hotel_name))}},
  {{ current wave is empty }}
  },
```

```
{comment;
The assertion associated with the state DDD1 of the script detects the situation that the quota of some LO is exhausted.
```

In each cycle the DDDi state may be activated only when the current wave becomes empty. It happens when all subtransactions of the current wave will be executed.

```
  },
  {DDD2;
  assertions:
  {{ex hotel_name (available_reserv > quota &
    hotel(hotel_name))}},
```

```
    {{ current wave is empty }}
  };
```

```
{comment;
```

The assertion associated with the state DDD2 of the script detects the situation that the actual number of rooms available for some LO exceeds its quota.

```
};
```

```
transitions:
```

```
{from: start;
to: DDD1, DDD2, cycle;
action: },
```

```
{comment;
```

The state *cycle* is used to organization cycled execution of deferred subtransactions. The *cycle* primitive is used for that.

```
};
```

```
{from: DDD1;
to: D1;
condition: {{count(r, reservation(r/self)) >
  0.7 * total(quota, hotel(quota))}};
```

```
{comment;
```

The condition corresponds to the first rule of the hotel example: whether the total amount of CO reservations exceeds 70% of the sum of quotas of all LOs. Recall that the state DDD1 denotes the abnormal situation in which the quota available to the CO of the relevant LO has exhausted.

```
};
```

```
action:
  {{deferred_spawn(get_additional_quota, tr1)
  }}},
```

```
{comment;
```

The *deferred\_spawn* primitive above creates a new transaction belonging to the *get\_additional\_quota* class and includes the transaction *tr1* (declared as the output parameter of the primitive *deferred\_spawn*) into the next wave set. The transition leads to the intermediate state D1 state which is introduced to wait for the completion of *tr1*.

When script transition activity leading to spawning of subtransactions is completed, the transition corresponding the cycle input state will be fired. The *cycle* primitive starts executing already prepared wave of subtransactions. Commit of each such subtransaction leads to its exclusion from the current wave set. In particular, when *tr1* completes, the transition from D1 to DDD1 is fired. Finally, current wave becomes empty. Then DDDi states are again activated and new cycle begins (now next wave might not be empty: it may contain deferred subtransactions created during the current wave).

```

    },
    {from: DDD2;
     to: D2;
     condition: {{count(r, reservation(r/self)) <
                 0.5 * total(quota, hotel(quota))}};
    {comment;
     This condition checks whether the total amount of
     CO reservations is below 50% of the sum of quotas of
     all LOs.
    }
    };
    action:
    {{{deferred_spawn(release_extra_quota, tr2)
     }}}},
    {from: D1;
     to: DDD1;
     condition: {{ ¬((tr1 ∈ current_wave) ∨
                    (tr1 ∈ next_wave))}};
     action: },
    {from: D2;
     to: DDD2;
     condition: {{ ¬((tr2 ∈ current_wave) ∨
                    (tr2 ∈ next_wave))}};
     action: },
    {comment;
     Transitions from states D1 and D2 below take place
     when deferred subtransaction tr1 (tr2) generated on
     transition from DDD1 to D1 (DDD2 to D2) had been
     executed. To detect this situation it is sufficient to
     check that tr1 (tr2) does not belong to both cur-
     rent_wave and next_wave sets.
    }
    };
    {from: cycle;
     to: cycle;
     condition: {{ current wave is empty and next
                    wave is not empty, no transition is
                    possible except this one }}
     action: cycle(this) }
    {comment;
     Parameter this refers to the current instance of the
     script class. };
    }

```

The above example of a script-based transaction can also be seen as an example of a polytransaction [SRK92]. Polytransactions were introduced for maintaining consistency among interdependent data stored in multiple database systems. Interdatabase dependency and consistency requirements are specified using the Data Dependency Descriptors ( $D^3$ ) that include 1) an interdatabase dependency predicate  $P$ ; 2) a mutual

consistency predicate stating consistency requirements  $C$  that define when  $P$  must be satisfied; and 3) a collection  $A$  of consistency restoration procedures needed to restore consistency and to ensure that  $P$  is satisfied as specified by  $C$ . Our example includes two  $D^3$  expressed by the DDD1 and DDD2 states, associated assertions (corresponding to interdatabase states that model their  $P$ s) and transitions leading from DDD1, DDD2 that use consistency restoration procedures.

## 4 Conclusion

A classification of information management issues in multidatabase system environments is presented along three dimensions: multidatabase information modeling, multidatabase operation interdependencies, and multidatabase consistency constraints. Though these three dimensions are not fully orthogonal, we feel that at a conceptual level, it helps to consider each of them separately.

The concept of the information repository is presented. In the repository each dimension mentioned above is structured into objects of three different types. The meta-meta level provides a description of one or more models for each type of object. Each meta level object is described according to a relevant model described at the meta-meta level. At the object level, we have individual data object extensions, codes for individual methods and transactions, and descriptions or implementations of individual components of constraint specifications.

We believe that a comprehensive and uniform representation of information along each of these three dimensions can help to beneficially exploit the interactions among them for better information management. It can also provide a more comprehensive design, development, and operation of the multisystem applications and individual components (application systems and/or databases) in a multidatabase environment. A uniform and comprehensive representation of all these types of information is required to exploit significant interplay between these different types of information during information management activities such as information system design providing for semantic interoperation, query processing, and transaction and consistency constraint management. To demonstrate the feasibility of a comprehensive and uniform representation, we used the SYNTHESIS language that was designed for developing heterogeneous interoperable information resource environment. Using the SYNTHESIS language, we modeled the relevant types of information for supporting an example multisystem

application. Modeling of transaction execution semantics and consistency constraints were demonstrated in more detail.

## References

- [ANRS92] M. Ansari, L. Ness, M. Rusinkiewicz, and A. Sheth. Using Flexible Transactions to Support Multi-system Telecommunication Applications. In *Proceedings of the 18th Intl. Conf. on Very Large Data Bases*, Vancouver, Canada, August 1992.
- [BR87] B.R. Badrinath and K. Ramamritham. Semantics-based concurrency control: Beyond commutativity. In *Proceedings of the ACM SIGMOD Conference*, 1987.
- [C91] P. Cannata. The Irresistible Move Towards Interoperable Database Systems. In *Proc. of the Int'l Wrkshp on Interoperability in Multidatabase Systems*, Kyoto, Japan, April 1991.
- [CR91] P. P. Chrysanthis and K. Ramamritham. A Formalism for Extended Transaction Models. *Proceedings of the 17th International Conference on Very Large Data Bases*, September 1991.
- [DHL90] U. Dayal, M. Hsu, and R. Ladin. Organizing Long-Running Activities with Triggers and Transactions. In *Proceedings of ACM SIGMOD Conference on Management of Data*, 1990.
- [DHL91] U. Dayal, M. Hsu, R. Ladin. A transactional model for long-running activities *Proceedings of the 17th International Conference on Very Large Data Bases*, September 1991.
- [E92] A. Elmagarmid, Ed. Database Transaction Models for Advanced Applications, Morgan-Kaufmann, 1992.
- [K91] L. Kalinichenko. SYNTHESIS: a language for description, design and programming of interoperable information resource environment. Technical Report, Institute of Problems of Informatics of the USSR Academy of Sciences, September 1991 (in Russian).
- [KZ91] L. Kalinichenko, V. Zadorozhny. A generalized information resource query language and basic query evaluation technique. In *Proceedings of the Second International Conference on Deductive and Object-oriented Databases*, Munich, Germany, December 1991.
- [KMS91] K.C.Lee, W. Mansfield, and A. Sheth. An Interactive Transaction Model for Distributed Cooperative Tasks. *Data Engineering Bulletin*, Vol. 14, No. 1, March 1991.
- [KRST92] P. Krychniak, M. Rusinkiewicz, A. Sheth, and G. Thomas. Bounding the Effects of Compensation under Relaxed Multi-level Serializability. Bellcore Technical Memorandum, TM-TSV-021509/1, June 15, 1992.
- [RC91] K. Ramamritham and P. Chrysanthis. In Search of Acceptability Criteria: Database Consistency Requirements and Transaction Correctness Properties. Technical Report TR 91-92, Computer Science Department, University of Massachusetts, December 1991.
- [RSK91] M. Rusinkiewicz, A. Sheth and G. Karabatis. Specifying Interdatabase Dependencies in a Multidatabase Environment. *IEEE Computer*, December 1991.
- [S91] A. Sheth, Ed. Special Issue on Semantic Issues in Multidatabase Systems. *SIGMOD Record*, December 1991.
- [SK92] A. Sheth and V. Kashyap. Too Far (Schematically) yet too Close (Semantically). *invited paper, to appear in the Proc. of the Intl. Conf. on Semantics of Interoperable Systems (DS-5)*, November 1992.
- [SL90] A. Sheth and J. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3), September 1990.
- [SRK92] A. Sheth, M. Rusinkiewicz, and G. Karabatis. Using Polytransactions to Manage Interdependent Data. Chapter 14, in [E92].
- [WQ87] G. Wiederhold and X. Qian. Modeling Asynchrony in Distributed Databases. In *Intl. Conf. on Data Engineering*, February 1987.