Transaction-Based Specification of Database Evolution

Lars Bækgaard
Department of Information Science, the Aarhus School of Business
Fuglesangs Allè 4, DK-8210 Aarhus V, Denmark
Email: lb@hha.dk

Abstract: We present a two-layer language for the specification of database evolution in terms of transaction-based, dynamic integrity constraints. The first language layer is based on first-order logic and it is used to express dynamic constraints in terms of queries on the transaction history of a database. The second layer uses a customizable combination of text and graphics and its semantics are defined in terms of the first-order language. Our language is orthogonal to state-based constraint languages and it can be used as a supplement to these. Also, our language can be used in combination with all object-based or entity-based data models. We use examples to illustrate the use of the specification language.

Key Terms: Database evolution, database integrity, dynamic constraint specifications, transaction-based constraint specifications.

1. Introduction

In this paper we present a language for the specification of the evolution of operational databases. Typically, operational databases are characterized by a high volume of update transactions each modifying a few database objects. Potentially, each update transaction is capable of violating database integrity. In order to maintain database integrity it is necessary to specify and enforce a set of integrity constraints (Formica and Missikoff 1992), (Hall and Gupta 1991), (Martin, Abida et al. 1992), (Motro 1989), (Mück and Vinek 1989), (Ngu 1990), (Simon and Valduriez 1984), (Stonebraker 1975).

Traditionally, integrity constraints have been specified in terms of database states. Some approaches support the specification of integrity constraints in terms of one database state (Brodie 1980) whereas other approaches support the specification of constraints in terms of relationships between two or more database states (Castilho, Casanova et al. 1982), (Cervasato and Eick 1992), (Chomicki 1992), (Kung 1985), (Lipeck 1986), (Lipeck and Saake 1987) (Vianu 1988)], (Wieringa, Meyer et al. 1989).

The transaction-based specification of object life cycles in terms of transaction histories is supported by systems development methods like JSD (Jackson 1983). In such approaches regular expressions are used to define legal transaction sequences (Bækgaard 1993), (Kappel and Schrefl 1991), (Rosenquist 1982). Transaction-based approaches are characterized by a much more loose coupling between constraint specifications and database schemes than are state-based specification languages.

However, we argue that regular languages are inappropriate as general specification languages.

Our specification approach is transaction-based and it supports the specification of regular expressions over transaction names as a special case. We use a two-layer, customizable language. The first layer is textual and based on first-order logic that is used to formulate constraints in terms of queries on transaction histories. The second layer uses customizable graphical symbols and its semantics are defined in terms of first-order queries on transaction histories.

Our specification approach is flexible in a number of ways. *First*, our specification language is independent of the structure of database objects. It can be used to supplement object-based models (Kim and Lochocsky 1989), (Kim 1995) and entity-based data models (Hammer and McLeod 1981), (Shipman 1981). It can even be used to supplement record-based models like the relational model (Codd 1970) if certain attributes are used as immutable surrogates (Khoshafian and Copeland 1986). *Second*, our language is independent of the specification of transaction bodies (Brodie and Ridjanovic 1984), (Gray 1981), (Leonard and Luong 1981), (Ngu 1984), (Qian 1988). *Third*, our specification approach supports the definition of customized graphical specification symbols. Consequently, the specification language can be tailored to specific specification needs.

The paper is organized as follows. In Section 2 we illustrate the limitations of regular expressions and we motivate the need for a more powerful specification approach. In Section 3 we present a data model that supports the specification of transaction preconditions in terms of queries on transaction histories. In Section 4 we define the syntax and semantics of a customizable graphical specification language. In Section 5 we illustrate the use of our specification language by means of examples. In Section 6 we conclude the paper and suggest directions for future research.

2. Motivation

Systems development methods like JSD (Jackson 1983) supports the specification of transaction-based, dynamic constraints in terms of regular expressions over transaction names.

In Figure 1 we have specified the dynamics of library borrowers as a graph that is equivalent to a regular expression over the transaction names Join, Borrow, Return, and Quit. Figure 1 is interpreted in the following way. The sequence of transactions in which a particular Borrower object participates must define a path in the specified graph. The first transaction must be Join and the last transaction must be Quit. In-between Join and Quit a particular Borrower object can participate in any sequence of Borrow and Return transactions. The fact that Join and Quit are underlined means that a specific Borrower object can participate in at most Join transaction and at most one Quit transaction.

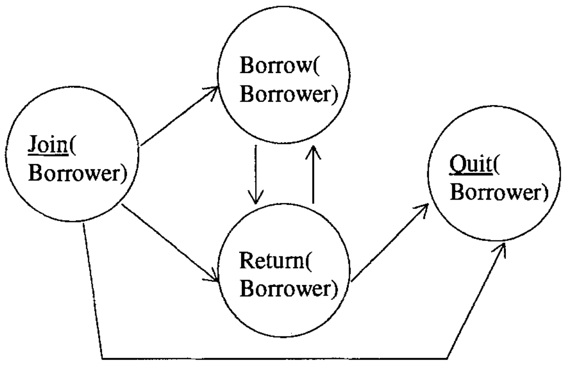

Figure 1 Library borrower as regular expression.

In Figure 2 we have used our specification approach to define the dynamics of library borrowers. In addition to the four transaction types in Figure 1 we have added a Reserve transaction.

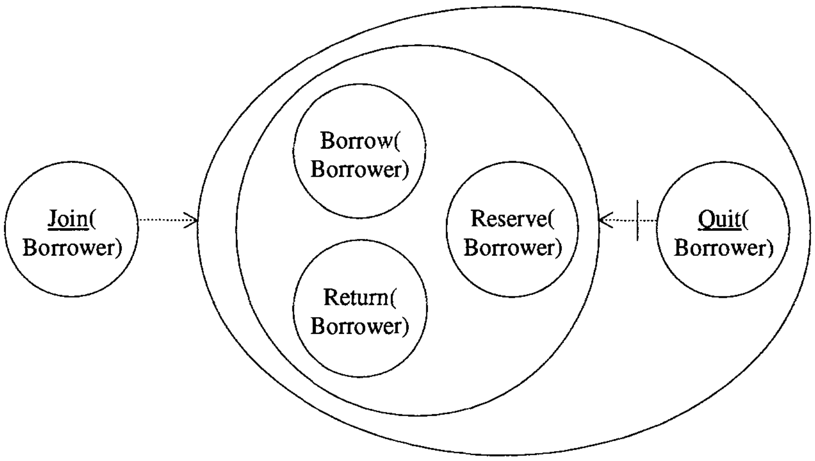

Figure 2 Library borrower.

Figure 2 is interpreted as follows. The transaction Quit is admissible for a particular Borrower object, *o*, if *o* has participated in a Join transaction in the past. The transactions Borrow, Return, and Reserve are admissible for a particular Borrower object, *o*, if *o* has participated in a Join transaction in the past and if *o* has not participated in a Quit transaction in the past.

A comparison of Figure 1 and Figure 2 indicates the fact that regular expressions are inappropriate in situations where the transactions are relatively independent. In such situations the regular expression approach forces the designer to specify transaction independence in terms of a possible huge set of explicit paths. This will become evident if the Reserve transaction as added to the specification in Figure 1.

In Section 3-4 we present a specification language that facilitates specifications like the one in Figure 2. Our language facilitates the specification of regular expressions over sets of transaction names as a special case.

3. Queries on Transaction Histories

In this section we present a first-order logic language that can be used to formulate transaction preconditions in terms of queries on transaction histories. We use such preconditions to constrain the sequencing of the transactions that create, modify, and destroy objects. In Section 4 we use the first-order logic language to specify the semantics of graphical symbols like the ones used in Section 2.

3.1 States and Transactions

We model database evolution as a state/transaction sequence where transitions between successive database states are caused by atomic update transactions. This is illustrated in Figure 3 where each s_i represents an explicit database state and each t_j represents an update transaction causing a transition from state s_{j-1} to state s_j. The state s_0 is an empty, initial state.

$$\begin{tikzcd} \boxed{s_0} \arrow[r, "t_1"] & \boxed{s_1} \arrow[r, "t_2"] & \cdots \arrow[r, "t_{n-1}"] & \boxed{s_{n-1}} \arrow[r, "t_n"] & \boxed{s_n} \end{tikzcd}$$

Figure 3 Database evolution.

A database state is a collection of object classes. An object class is a named set of objects. An object is composed of an immutable, globally unique object identifier and a body. The expression *o*.OID denotes the object identifier of the object *o*. The symbol $\text{Borrower}_{\text{id}}$ denotes the set of object identifiers for objects in the object class named Borrower.

Our specification language is independent of the specification of object bodies. If the underlying data model is a semantic data model like SDM (Hammer and McLeod 1981) or a structural object-oriented model like Telos (Mylopoulos 1992) the object body contains a set of state attributes. If the underlying data model is a behavioral object-oriented data model (Kim 1995) the object body contains the possible object behavior specified as a set of services in addition to the state attributes. Each object is a member of one or more classes. Objects can join and quit object classes dynamically.

We use our specification language to formulate dynamic constraints in terms of queries on the transaction histories of database objects. A database transaction is an execution of a transaction definition. Each transaction definition is composed of a type and a body. Our specification language is independent of the specification of transaction bodies (Brodie and Ridjanovic 1984), (Gray 1981), (Leonard and Luong 1981), (Ngu 1984), (Qian 1988). The type of a transaction is composed of a name and a set of formal parameters. The parameters define the possible transaction participants. For example, the expression Borrow (x: $\text{Borrower}_{\text{id}}$, y: Book_{id}) defines the type of a borrow transaction in a library database.

3.2 Transaction Histories

We use queries on transaction histories to facilitate the specification of transaction preconditions. Each object has a transaction history that can be described in terms of the sequence of transactions that has affected the object. We describe the transaction history as a set with the following structure.

HIST:	{TRANS}
TRANS:	(TNAME, TIME, OBJECTS)
OBJECTS:	{(OID, ONAME)}

Each element in HIST represents a historical transaction occurrence. For such an element, *trans*, the expressions *trans*.TNAME, *trans*.TIME, and *trans*.OBJECTS denote the name of the transaction type, the commit time, and the set of identifiers (*trans*.OBJECTS.OID) and class names (*trans*.OBJECTS.ONAME) for the participating objects, respectively. We assume that time stamps are unique.

HIST is updated whenever a transaction commits. For example, when the transaction Borrow (oid_{13}, oid_{22}) commits at time 070797.1154 the following element is added to HIST.

("Borrow", 070797.1154, {(oid_{13}, "Borrower"), (oid_{22}, "Book")})

3.3 Preconditions

We define preconditions as functions with the following type.

$$\text{Oid} \rightarrow \text{Boolean}$$

Precondition functions have this type because we use them to control the evolution of individual objects. All free variables in the definition of such a function must refer to Oid or to an element in HIST.

We use the following notation to specify a precondition for a specific transaction type/object class pair. The symbols *tname*, *oname*, and *p* denotes a transaction type name, an object class name, and a precondition function, respectively.

PRE (*tname*, *oname*) = *p*

Such a specification is interpreted in the following way. *p*(*o*.OID) must evaluate to true in order for an object, *o*, from the object class named *oname* to participate in a transaction of the type named *tname*.

When a transaction is submitted for execution the specified preconditions for all participating objects are evaluated. The transaction is admitted if and only if all these preconditions evaluate to true.

In order to facilitate the definition of precondition functions we define function generators with the following type. The domain Value denotes one of the domains Boolean, Integer, Real, or String.

Parameters $\rightarrow$ (Oid $\rightarrow$ Value)

All free variables in the definition of a function generator must refer to Parameters, to Oid, or to an element in HIST.

3.4 Sample Function Generators

The following examples illustrate the definition of function generators. The symbols *oname*, *tname*, *oid*, and *trans* denote an object class name, a transaction type name, an object identifier, and a transaction history element, respectively.

The function generator COUNT has the following type and definition.

COUNT: (String, String) $\rightarrow$ (Oid $\rightarrow$ Integer)

COUNT[*tname*, *oname*](*oid*) =
CARDINALITY{*trans*∈HIST | (*oid*, *oname*)∈*trans*.OBJECTS ∧ *trans*.TNAME=*tname*)}

For example, COUNT["Borrow", "Borrower"] is a function that returns the number of times a specific object has participated in a Borrow transaction as a Borrower object. COUNT["Borrow", "Borrower"] has the following type and definition.

COUNT["Borrow", "Borrower"]: Oid $\rightarrow$ Integer

COUNT["Borrow", "Borrower"](*oid*) =
CARDINALITY{*trans*∈HIST | (*oid*, "Borrower")∈*trans*.OBJECTS ∧ *trans*.TNAME="Borrow")}

The function generator LAST has the following type and definition.

LAST: (String, String) $\rightarrow$ (Oid $\rightarrow$ Boolean)

LAST[$tname_1$, $oname_1$] (oid) =
$\{trans_1 \in$ HIST $\mid (oid, oname_1) \in trans_1$.OBJECTS $\wedge$
$trans_1$.TNAME= $tname_1 \wedge trans$.TIME =
MAXIMUM$\{time \mid \exists\ trans_2 \in$ HIST, $oname_2$: $time=trans$.TIME $\wedge$
$(oid, oname_2) \in trans_2$.OBJECTS$\}\}\neq\emptyset$

For example, LAST[“Return”, “Book”] is a function that returns true if and only if the most recent transaction in which a specific object, o, has participated is a Return transaction and if o has participated as a Book object. LAST[“Return”, “Book”] has the following type and definition.

LAST[“Return”, “Book”]: Oid $\rightarrow$ Boolean

LAST[“Return”, “Book”](oid) =
$\{trans_1 \in$ HIST $\mid (oid,$ “Book”$) \in trans_1$.OBJECTS $\wedge$
$trans_1$.TNAME=“Return” $\wedge\ trans$.TIME =
MAXIMUM$\{time \mid \exists\ trans_2 \in$ HIST, $oname_2$: $time=trans$.TIME $\wedge$
$(oid, oname_2) \in trans_2$.OBJECTS$\}\}\neq\emptyset$

The function generator EXISTS has the following type and definition.

EXISTS: ((String, String) $\rightarrow$ (Oid $\rightarrow$ Boolean))

EXISTS[$tname$, $oname$](oid) =
$\{trans \in$ HIST $\mid (oid, oname) \in trans$.OBJECTS $\wedge$
$trans$.TNAME=$tname$ $\}\neq\emptyset$

For example, EXISTS[“Borrow”, “Book”] is a function that returns true if and only if a specific object has participated in a Borrow transaction as a Book object in the past. EXISTS[“Borrow”, “Book”] has the following type and definition.

EXISTS[“Borrow”, “Book”]: Oid $\rightarrow$ Boolean

EXISTS[“Borrow”, “Book”](oid) =
$\{trans \in$ HIST $\mid (oid,$ “Book”$) \in trans$.OBJECTS $\wedge$
$trans$.TNAME=“Borrow”$\}\neq\emptyset$

3.5 Aggregation of Function Generators

We use the rules in Table 1 to aggregate precondition functions.

Input expression	Aggregated expression
(Oid $\rightarrow$ Boolean$_1$) $\wedge$ (Oid $\rightarrow$ Boolean$_2$)	Oid $\rightarrow$ (Boolean$_1$$\wedge$Boolean$_2$)
(Oid $\rightarrow$ Boolean$_1$) $\vee$ (Oid $\rightarrow$ Boolean$_2$)	Oid $\rightarrow$ (Boolean$_1$$\vee$Boolean$_2$)
$\neg$(Oid $\rightarrow$ Boolean)	Oid $\rightarrow$ $\neg$(Boolean)
(Oid $\rightarrow$ Nummber$_1$) + (Oid $\rightarrow$ Number$_2$)	Oid $\rightarrow$ (Number$_1$+Number$_2$)
(Oid $\rightarrow$ Number$_1$) - (Oid $\rightarrow$ Number$_2$)	Oid $\rightarrow$ (Number$_1$-Number$_2$)
(Oid $\rightarrow$ Number$_1$) * (Oid $\rightarrow$ Number$_2$)	Oid $\rightarrow$ (Number$_1$*Number$_2$)
(Oid $\rightarrow$ Number$_1$) / (Oid $\rightarrow$ Number$_2$)	Oid $\rightarrow$ (Number$_1$/Number$_2$)
-(Oid $\rightarrow$ Number)	Oid $\rightarrow$ -(Number)
(Oid $\rightarrow$ Value$_1$) > (Oid $\rightarrow$ Value$_2$)	Oid $\rightarrow$ (Value$_1$>Value$_2$)
(Oid $\rightarrow$ Value$_1$) < (Oid $\rightarrow$ Value$_2$)	Oid $\rightarrow$ (Value$_1$<Value$_2$)
(Oid $\rightarrow$ Value$_1$) = (Oid $\rightarrow$ Value$_2$)	Oid $\rightarrow$ (Value$_1$=Value$_2$)

Table 1 Aggregation of precondition functions.

The following example illustrates the use of Table 1.

EXISTS["Buy", "Book"] $\wedge$ LAST["Return", "Book"](*oid*) =
({*trans*$\in$HIST | (*oid*, "Book")$\in$*trans*.OBJECTS $\wedge$
trans.TNAME="Buy"}$\neq\emptyset$) $\wedge$
({*trans$_1$*$\in$HIST | (*oid*, "Book")$\in$*trans$_1$*.OBJECTS $\wedge$
trans$_1$.TNAME="Return" $\wedge$ *trans*.TIME =
MAXIMUM{*time* | $\exists$ *trans$_2$*$\in$HIST, *oname$_2$*: *time*=*trans*.TIME $\wedge$
(*oid*, *oname$_2$*)$\in$*trans$_2$*.OBJECTS}}$\neq\emptyset$)

This aggregated precondition function returns true if and only if a specific object, *o*, has participated in a Buy transaction as a Book object in the past and if the most recent transaction in which *o* participated was a Return transaction in which *o* participated as a Book object.

4. A Graphical Specification Language

In this section we show how to specify the semantics of a graphical constraint language in terms of preconditions specified as queries on transaction histories.

Figure 4 defines the syntax and semantics of a graphical symbol for the specification of preconditions. For each graphical symbol the syntax is defined above the line and the semantics are defined below the line. Each *p* denotes a precondition function as defined in Section 3, each *oname* denotes an object class name, and each *tname* denotes a transaction type name.

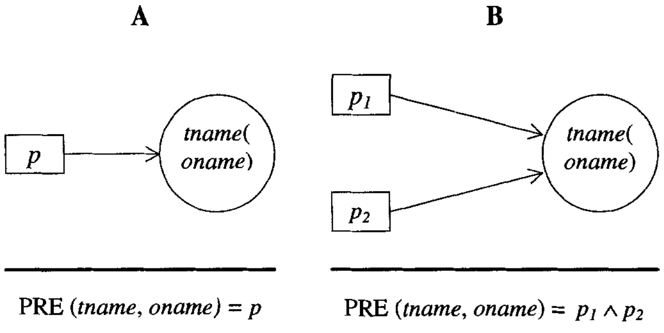

Figure 4 Precondition.

Figure 4 (A) is interpreted in the following way. An *oname* object, *o*, can participate in a *tname* transaction if and only if *p*(*o*.OID) evaluates to true. Two or more preconditions are implicitly combined using logical and ($\wedge$) as defined by the aggregation rules in Table 1.

Figure 5 defines the syntax and semantics of a graphical symbol for the specification of immediate predecessors. Figure 5 (A) is interpreted in the following way. An *oname* object, *o*, can participate in a *tname* transaction if and only if the most recent transaction in which *o* participated was a $tname_1$ transaction in which *o* participated as an $oname_1$ object. Figure 5 (B) is interpreted in the following way. An *oname* object, *o*, can participate in a *tname* transaction if and only if the most recent transaction in which *o* participated was a $tname_1$ transaction in which *o* participated as an $oname_1$ object or if the most recent transaction in which *o* participated was a $tname_2$ transaction in which *o* participated as an $oname_2$ object.

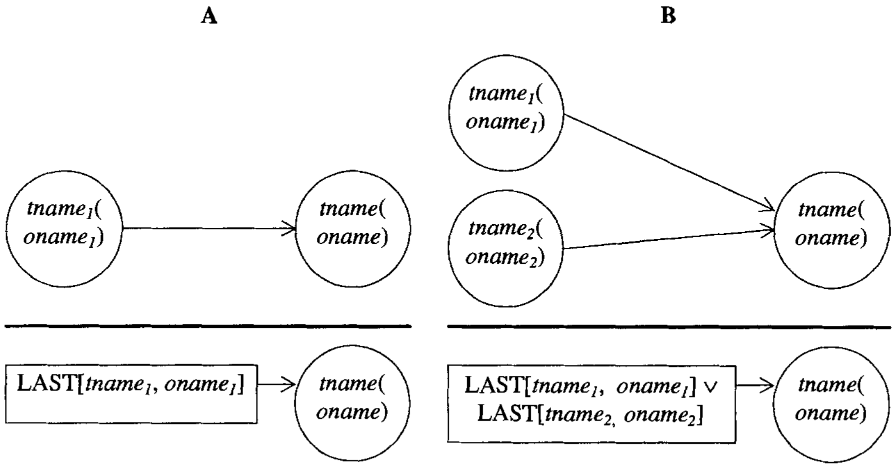

Figure 5 Immediate predecessors.

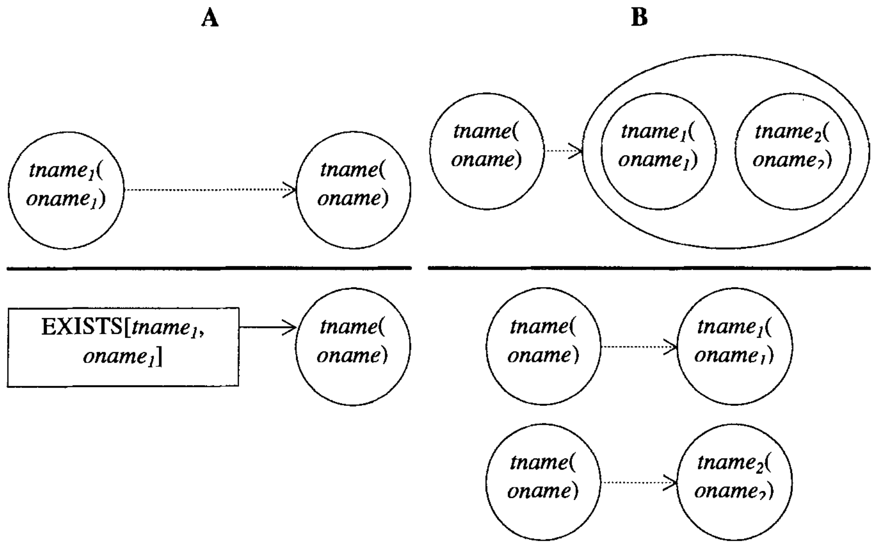

Figure 6 Enabling, non-immediate predecessors.

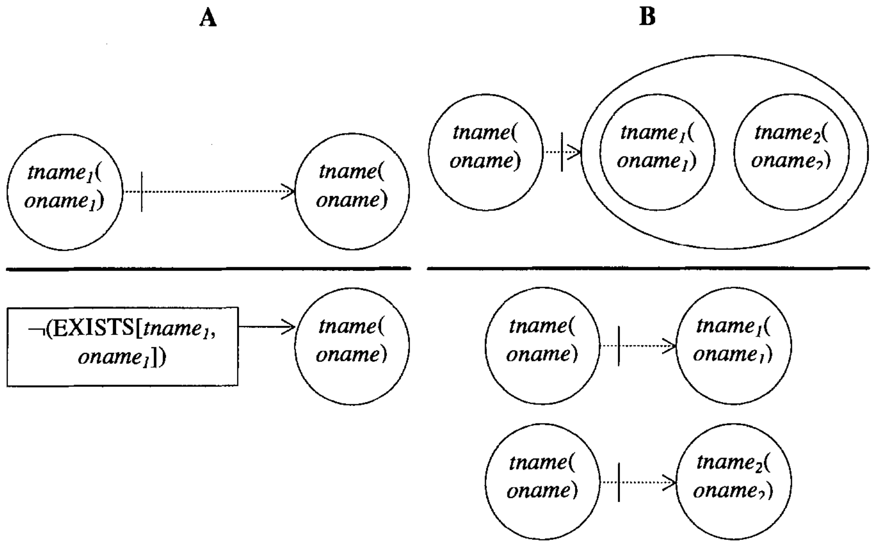

Figure 7 Disabling, non-immediate predecessors.

Figure 6 defines the syntax and semantics of graphical symbols for the specification of enabling, non-immediate predecessors. The dotted arrows indicate non-immediate predecessors. Figure 6 (A) states that an *oname* object, *o*, can participate in a *tname* transaction if and only if *o* has participated in a *tname*$_1$ transaction as an *oname*$_1$ object in the past.

Figure 7 defines the syntax and semantics of graphical symbols for the specification of disabling, non-immediate predecessors. Figure 7 (A) states that an *oname* object, *o*, can participate in a *tname* transaction if and only if *o* has not participated in a *tname*$_1$ transaction as an oname$_1$ object in the past. Figure 8 defines a graphical symbol that can be used to ensure that objects can participate in a transaction of a certain type at most one time.

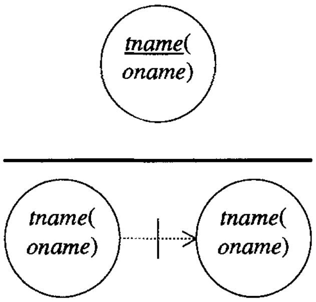

Figure 8 Single-occurrence events.

5. Specification Examples

In this section we use specification examples from a library to illustrate the use of our specification language. Reserve is an independent transaction that is not returned by the function LAST.

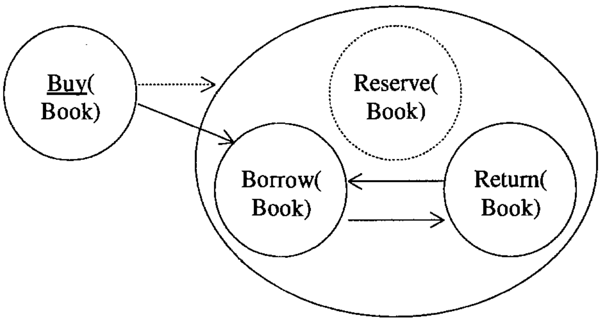

Figure 9 Library book.

The specification in Figure 9 implies that the first transaction in which a Book object participates must be a Buy transaction. A particular Book object can participate in at most one Buy transaction. Then, the Book object must participate in an alternating sequence of Borrow and Return transactions starting with a Borrow transaction. In-between the Borrow and Return Transactions the Book object can participate in Reserve transactions.

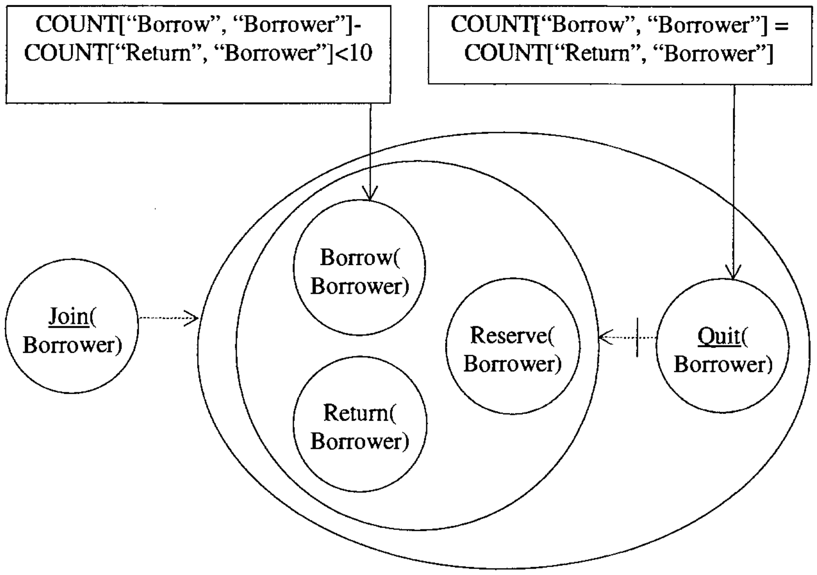
COUNT["Borrow", "Borrower"]-COUNT["Return", "Borrower"]<10
COUNT["Borrow", "Borrower"] = COUNT["Return", "Borrower"]
Join(Borrower)
Borrow(Borrower)
Return(Borrower)
Reserve(Borrower)
Quit(Borrower)

Figure 10 Library borrower.

The specification in Figure 10 implies that the first transaction in which a Borrower object participates must be a Join transaction. Then, the Borrower object can participate in any sequence of Borrow, Return, and Reserve transactions. The number of currently borrowed books is not allowed to exceed 10. Finally, the Borrower object can participate in a Quit transaction if all borrowed books have been returned.

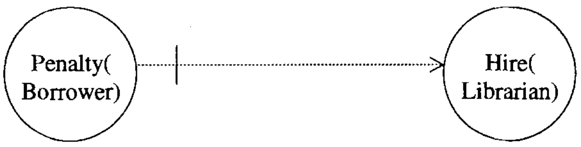

Figure 11 Role modeling in library.

Figure 11 is interpreted as follows. A Librarian object can participate in a Hire transaction if and only if the object has not participated in a penalty transaction in the past. This illustrates that our specification language can be used to define constraints that relate the various roles objects may play during their time of existence.

6. Conclusion

We have presented a two-layer language that supports the specification of database evolution in terms of transaction-based, dynamic integrity constraints. The basic layer of our language is based on first-order logic. The underlying idea is to specify dynamic database constraints in terms of the transaction history. We have used the basic layer to define the semantics of a graphical language for transaction-based specification of database evolution.

Our two-layer language is orthogonal to existing state-based specification languages. Consequently, it can be used as a supplement to these languages. Our language can be used as a supplement to any data model that supports immutable object identifiers or immutable surrogates. Our language is characterized by a very loose coupling between constraint specifications and other schema elements like transaction specifications and object type definitions. Potential application areas include value chain specification, product flow specification, quality control systems, work flow specification, and process-based role modeling.

Future work includes further experiments with application areas, development of analysis tools that can support formal analysis of the consistency and correctness of specifications, and development of efficient implementation methods. Currently, we are extending our language with an event triggering mechanism.

References

Brodie, M. L. (1980). "The Application of Data Types to Database Semantic Integrity." Information Systems 5: 287-296.

Brodie, M. L. and D. Ridjanovic (1984). "On the Design and Specification of Database Transactions". Topics in Information Systems. On Conceptual Modeling. M. L. Brodie, J. Mylopoulos and J. W. Schmidt, Springer-Verlag: 277-312.

Bækgaard, L. (1993). "Specification and Efficient Monitoring of Transaction Dependencies. Ph.D. Thesis". Department of Mathematics and Computer Science, Aalborg University.

Castilho, J. M. V. D., M. A. Casanova, et al. (1982). "A Temporal Framework for Database Specifications". 8th International Conference on Very Large Databases, Mexico City, Mexico.

Cervasato, I. and C. F. Eick (1992). "Specification and Enforcement of Dynamic Consistency Constraints". CIKM'92. ISSM International Conference on Information and Knowledge Management, Baltimore, Maryland, USA.

Chomicki, J. (1992). "History-Less Checking of Dynamic Integrity Constraints". 8th International Conference on Data Engineering, Tempe, Arizona, USA, Computer Society Press.

Codd, E. F. (1970). "A Relational Model of Data for Large Shared Data Banks." Communications of the ACM 13(6): 377-387.

Formica, A. and M. Missikoff (1992). "Adding Integrity Constraints to Object-Oriented Models". CIKM'92. ISSM International Conference on Information and Knowledge Management, Baltimore, Maryland, USA.

Gray, J. (1981). "The Transaction Concept. Virtues and Limitations". VLDB'81. 7th International Conference on Very Large Databases, Cannes, France.

Hall, G. and R. Gupta (1991). "Modeling Transition". 7th International Conference on Data Engineering, Kobe, Japan, Computer Society Press.

Hammer, H. and D. McLeod (1981). "Database Description with SDM. A Semantic Database Model." ACM Transactions of Database Systems 6(3): 351-386.

Jackson, M. (1983). *System Development*, Prentice-Hall.

Kappel, G. and M. Schrefl (1991). "Object/Behavior Diagrams". 7th International Conference on Data Engineering, Kobe, Japan, Computer Society Press.

Khoshafian, S. and G. P. Copeland (1986). "Object Identity". Object-Oriented Programming Systems, Languages, and Applications-86, ACM.

Kim, W., Ed. (1995). *Modern Database Systems. The Object Model, Interoperability, and Beyond*, Addison-Wesley.

Kim, W. and F. H. Lochocsky, Eds. (1989). *Object-Oriented Concept, Databases, and Applications*, ACM Press, Addison-Wesley Publishing Company.

Kung, C. H. (1985). "On Verification of Temporal Database Constraints". SIGMOD'85. International Conference on Management of Data, Austin, Texas, USA.

Leonard, M. and B. Y. Luong (1981). "Information Systems Design Approach Integrating Data and Transactions". VLDB'81. 7th International Conference on Very Large Databases, Cannes, France.

Lipeck, U. W. (1986). "Stepwise Specification of Dynamic Database Behaviour". SIGMOD'86. International Conference on Management of Data, Washington, D.C.

Lipeck, U. W. and G. Saake (1987). "Monitoring Dynamic Integrity Constraints Based on Temporal Logic." Information Systems 12(3): 255-269.

Martin, H., M. Abida, et al. (1992). "Consistency Checking in Object-Oriented Databases. A Behavioral Approach". CIKM'92. ISSM International Conference on Information and Knowledge Management, Baltimore, Maryland, USA.

Motro, A. (1989). "Integrity = Validity + Completeness." ACM Transaction on Database Systems 14(4): 480-502.

Mück, T. and G. Vinek (1989). "Modeling Dynamic Constraints Using Augmented Place Transition Nets." Information Systems 14(4): 327-340.

Mylopoulos, J. (1992). "Conceptual Modeling and Telos". Conceptual Modeling, Databases, and CASE. An Integrated View of Information Systems. P. Loucopoulos and R. Zicari, John Wiley & Sons.

Ngu, A. H. H. (1984). "Transaction Modeling". 5th International Conference on Data Engineering, Los Angeles, USA, Computer Society Press.

Ngu, A. H. H. (1990). "Specification and Verification of Temporal Relationships in Transaction Modeling." Information Systems 15(2): 257-267.

Qian, X. (1988). "A Transaction Logic for Database Specification". SIGMOD'88. International Conference on Management of Data, Chicago, Illinois, USA.

Rosenquist, C. J. (1982). "Entity Life Cycle Models and their Applicability to Information Systems Development Life Cycles." Computer Journal 25(3).

Shipman, D. W. (1981). "The Functional Data Model and the Data Language DAPLEX." ACM Transactions on Database Systems 6(1): 140-173.

Simon, E. and P. Valduriez (1984). "Design and Implementation of an Extendible Integrity Subsystem". SIGMOD'84. International Conference on Management of Data, Boston, USA.

Stonebraker, M. (1975). "Implementation of Integrity Constraints and Views by Query Modification". SIGMOD'75. Workshop on Management of Data, San Jose, California, USA.

Vianu, V. (1988). "Database Survivability Under Dynamic Constraints." Acta Informatica 25: 55-84.

Wieringa, R., J.-J. Meyer, et al. (1989). "Specifying Dynamic and Deontic Integrity Constraints." Data & Knowledge Engineering 4: 157-189.