A Meta Model and an Infrastructure for the Non-Transparent Replication of Object Databases

Werner Dreyer
Microsoft Corp.[1]
dreyer@acm.org

Klaus R. Dittrich
Database Technology Research Group
Department of Information Technology,
University of Zurich, Switzerland
dittrich@ifi.unizh.ch

ABSTRACT

The distribution of data within or between organizations is crucial for many applications. Such a distribution may be transparent, as it is embodied in distributed database systems to increase their performance, reliability and availability. Other systems either do not require transparent data distribution or even explicitly need their own copies of the data. Examples are data warehouses which collect base data from different transaction processing systems, or financial data distribution systems in banking where data are bought from external providers and distributed to different working groups. Such systems need non-transparent, partial and selective replication of databases. If object databases are used, appropriate replication systems have to cope with the additional complexity of object models. Today's replication systems do not meet these requirements. Consequently, we introduce a meta model that specifies the abstract functionality which is needed for the non-transparent, partial and selective replication of object databases. This meta model allows incorporation of its replication concepts in a variety of object models. In addition, we present the design of a replication infrastructure, based on the meta model, which serves as the foundation for concrete realizations of object replication systems. The development of a replication system for a time series database management system proved the viability of the proposed solution.

Keywords

Object replication, object databases, replication meta model.

1. INTRODUCTION

Numerous distributed systems rely on data replication. Depending on their application domains, the properties of the replication systems vary heavily. For example, data are usually physically distributed and replicated but conceptually centralized in transaction-processing applications. In contrast to these, typical decision-support applications, e.g., financial time series management [5], explicitly distinguish between different copies of data. These copies do not have to be permanently synchronized. Company-wide access to such data is often crucial. Data are obtained from many providers. Financial institutions often store millions of time series. Most of these data are meant to be utilized by a variety of users, who have the following main requirements regarding data access:

Autonomy, local control over data. Most empirical investigations are based on raw data, that is, researchers do not want data that, for example, already underwent seasonal adjustments.

Data management by experts. When a company collects data from a variety of sources, one cannot expect a centralized department to be familiar with all these sources. Therefore, an architecture that allows the experts to manage their own data is preferable.

Access to the whole universe of externally bought financial data. It is more cost-effective for a company to have a global contract with a data provider and to subsequently distribute the data within the company.

Public access to project-generated data. Some of the project-generated data are also of interest to other users, that is, they must be accessible to those too.

Current systems in this and in similar application domains store their data in files only, or in both files and centralized databases, as shown in figure 1. Both approaches do not meet our requirements.

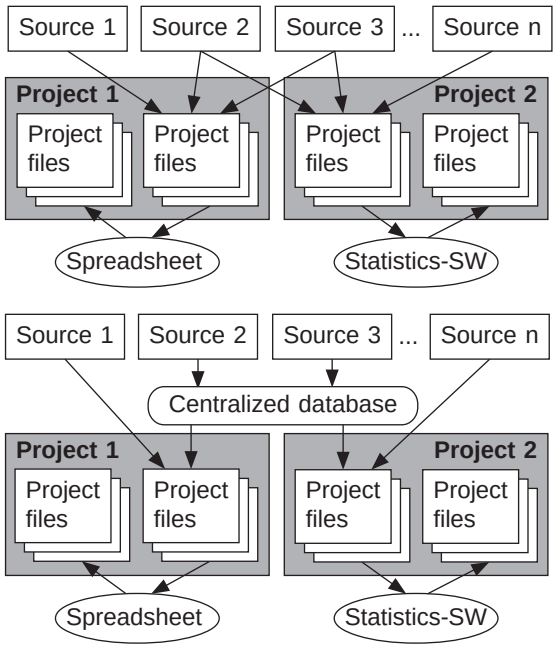

Figure 1: Two current system architectures.

The structure of this paper is as follows: An architecture based on partially replicated databases that meets the requirements regarding company-wide data access is presented in section 2. An object and type replication meta model that is the ground-

CIKM 2000, McLean, VA USA

[1] During the time of writing, the author was working for UBS AG, Switzerland (http://www.ubs.com).

work for the non-transparent, partial and selective replication of object databases is developed in section 3. Section 4 presents a replication infrastructure which allows incorporation of the main properties of the replication meta model into a concrete DBMS. Based on this infrastructure, the implementation of a replication system for a time series management system is illustrated in section 5. Section 6 presents related work. Section 7 contains the conclusions and an outlook.

2. A DATA MANAGEMENT ARCHITECTURE WITH PARTIALLY REPLICATED DATABASES

To meet our requirements, we propose an architecture based on partially replicated databases [10]. Figure 2 presents an overview of this architecture. Black squares represent objects which are stored in a database, dashed lines denote object replication.

The involved databases are autonomous, but all use the same object DBMS. Permitting heterogeneous DBMS would introduce additional requirements like data model transformations, but the basics of the replication functionality which are the core of this paper would remain the same.

Some of the databases are provider-specific, others are project-specific. Provider-specific databases are used to store provider-delivered data. They are administered by specialists with detailed knowledge about the respective data. These experts leverage the semantics of the data. For example, they structure objects in collections, and they add descriptive attributes. Structuring and descriptions are very important, as many providers offer very limited or no search facilities at all, and there are hardly any on-line descriptions of the available objects. Project-specific databases get some objects by replicating them from other databases, or directly from project-specific sources if these objects are not of interest to a wide range of users.

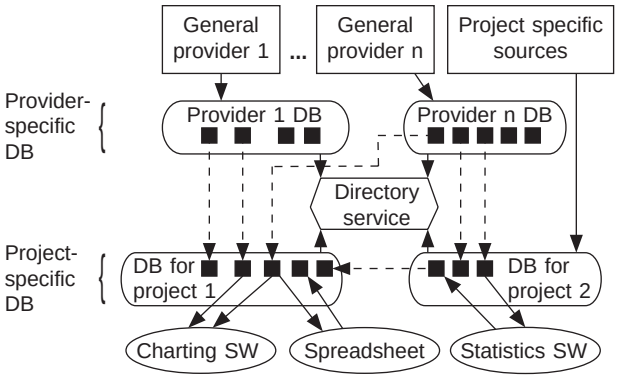

Figure 2: A data management architecture with integrated replication services.

Replication is realized by a publish-and-subscribe mechanism, so publishers do not need to know who will replicate them later on. If a user subscribes to an object, all directly or indirectly referenced objects are replicated too. For example, subscribing to a collection object leads to the replication of its members.

As it is not necessarily known at setup time of a project-specific database which objects will be replicated from other databases, the replication system must also be able to replicate the schema prior to object replication. In addition, subsequent schema changes have to be replicated, too.

The architecture presented in this section shows the following main characteristics: (1) The basic entities to be replicated are objects. (2) Non-transparent replication is used, that is, users explicitly work with a specific replica. (3) Replicas are weakly consistent, update delays can be tolerated. (4) A master-slave configuration is utilized. (5) Individual objects are eligible for replication. (6) Objects that are referenced by a replica are implicitly replicated, too. (7) The schema and its changes are implicitly replicated.

Comparing these characteristics with current replication systems and protocols (see section 6 for related work) shows that no available system has all these characteristics. Consequently, we propose the development of a special-purpose replication system that is tailored to the characteristics of our targeted data management architecture.

3. A META MODEL OF OBJECT AND TYPE REPLICATION

The meta model is the groundwork for non-transparent, partial and selective replication of object databases. It allows incorporation of its replication concepts in a variety of object models. Its main properties are as follows: (1) An object model with generic operations is presumed. (2) Replication takes place between independent databases. (3) Both objects and types are replicated. (4) A master-slave relationship between replicas exists. (5) Replication is non-transparent. (6) Explicit replication of certain objects leads to implicit replication of entire object and type graphs. (7) The semantics of modification operations is enhanced to meet the needs caused by replication. (8) The degree of implicit replication can be controlled. (9) Transactional properties are maintained.

3.1 Object model

As mentioned above, the replication meta model must be applicable for various object models. Therefore, it can only require the presence of a minimal, least common denominator, object model. We introduce a structural object model [6] which supports complex objects and generic operations.

Because user-defined operations can be mapped to a sequence of generic operations, our replication meta model can still be the basis of a fully object-oriented model .

Definition 1: *An **object** is a triple <identifier, state, behavior>. The identifier distinguishes each object from all others. The state of an object encompasses a set of properties and their actual values. The set of operations an object can execute determines its behavior.*

Properties are either attributes or relationships. Attributes are <name, value>-tuples, relationships are named references to other objects. They are either unidirectional or bidirectional.

Definition 2: *A **type** defines a set of objects that share common properties and common behavior. Every type has an identifier. Types can be organized into a type hierarchy.*

Every type comprises a set of attribute declarations, a set of relationship declarations and a set of operation declarations. An attribute declaration consists of a name and an attribute type. Relationship declarations comprise a name, the target type, the cardinality and, in case of bidirectional relationships, the name of the inverse relationship name. Operation declarations consist of the operation signature.

Objects belong to a most specific type (MST) and every other type higher up the hierarchy. For the remainder of this paper, we simply refer to the "type" of an object when we actually mean its MST.

Definition 3: *A **database** is a storage domain for objects. For every object it contains, it also stores the corresponding type. The set of types in a database constitute the database schema. Different databases always have disjunct contents.*

3.2 Inter-database replication

We design our replication meta model for non-transparent replication of objects and types between different databases. Of course, this does not prevent the DBMS from internally and transparently replicate objects for other reasons, for example, to improve performance by multiversion concurrency control [2].

3.3 Replication of single types

Before treating replication of entire object and type graphs, we introduce replication of single objects and types.

Definition 4: *Two types are **replica-equivalent** iff: (1) They define equal properties. The only possible exception are the target types of relationships, which may differ. (2) Non-nil target types of corresponding relationship declarations either have equal properties, or they only differ in their own target types. (3) The inheritance graphs of the two types and of their target types are isomorphic. All corresponding supertypes are again replica-equivalent.*

Nil-typed relationship declarations allow the selective replication of types without being forced to also replicate all related types.

Figure 3 shows an example where types `T2` and `T8` are replica-equivalent. We use OMT [24] notation with the following enhancements: Arrows between types denote unidirectional relationships, and grounding symbols stand for nil-typed relationships.

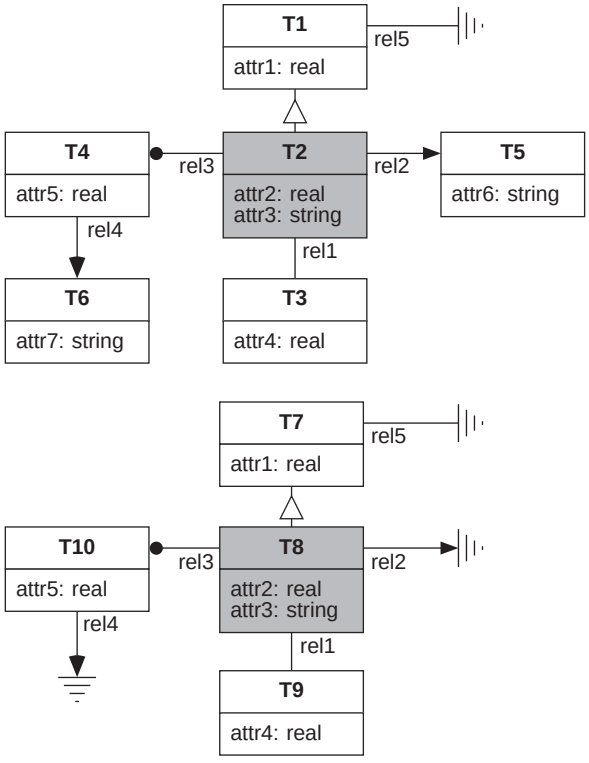

Figure 3: Replica-equivalent types.

Histories and combined histories are a further part of the meta model. The history of a type is the sequence of the type's attribute declarations, relationship declarations, operation declarations, and inheritance hierarchies over time. A combined history of a set of types is the merger of all the individual histories. Merging is based on causal ordering [18]. Thus, more than one combined history may be deduced from a set of individual histories.

Definition 5: *A **replicated type** manages a set of types. It has the following properties: (1) It modifies its types either on external request or to make them replica-equivalent to the current or a former state of a combined history of the whole set of types. (2) It enforces the history of every type within the set to be a replica-equivalent subsequence of the same combined history of the entire set.*

A replicated type is a conceptual notion and thus does not necessarily have to be materialized.

Definition 6: *Types are **type replicas** of each other iff they are managed by the same replicated type.*

Figure 4 presents an example of four replica-equivalent types. Vertical arrows represent replicas over time. Black bullets denote modifications by users or applications. Gray bullets stand for updates caused by the DBMS to restore replica-equivalence. Arrows between bullets visualize the flow of updates. The resulting causal ordering is shown in the lower part of figure 4. The numbers on the arrows indicate from which replica the causality can be deduced. The four possible combined histories are ABDEC, ABDCE, ABCDE, and ACBDE.

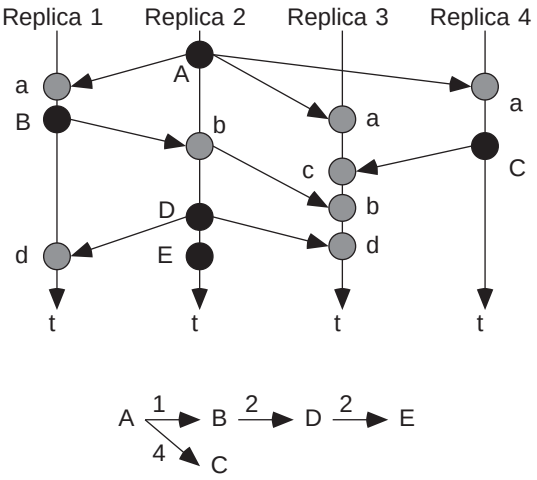

Figure 4: Replica histories and their causal ordering.

3.4 Replication of single objects

The definitions regarding object replication closely resemble those of type replication.

Definition 7: *Two objects are **replica-equivalent** iff: (1) Their types are replica-equivalent. (2) Their attribute values are equal. (3) Each pair of corresponding relationships either has target objects with equal attribute values and replica-equivalent types, or at least one of the target objects is nil.*

As for nil-typed relationship declarations, the reason to permit uninitialized relationships is that this allows more selective object replication.

The history of an object is its sequence of states. A combined object history is defined analogously to a combined type history.

Definition 8: *A **replicated object** manages a set of objects. It has the following properties: (1) It modifies its ob-*

jects either on external request or to make them replica-equivalent to the current or a former state of a combined history of the whole set of objects. (2) It enforces the history of every object within the set to be a replica-equivalent subsequence of the same combined history of the entire set.

Definition 9: *Objects are* ***object replicas*** *of each other if they are managed by the same replicated object.*

Figure 5 shows an example conceptual object diagram. The gray objects are replicas of each other. The grounding symbol stands for an uninitialized relationship. Types and replicated types are represented as first-class objects.

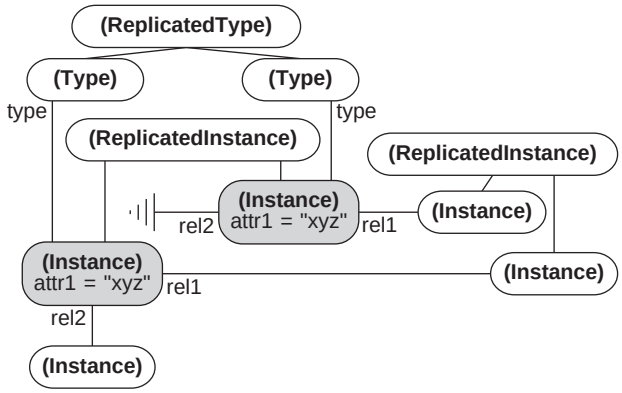

Figure 5: Conceptual diagram with object replicas.

3.5 Master-slave replication

There is a master-slave relationship between replicas. Users designate certain objects to be master replicas, and the replication system creates the corresponding slaves. The system modifies slave replicas according to updates of the respective master replicas, and it prevents users from directly altering slave replicas. This setup does not prohibit cascading replications, where a slave replica can itself be a master replica for another slave. However, every slave has exactly one master, and replication is unidirectional.

3.6 Non-transparent replication

According to the needs of the targeted application domain, non-transparent replication is utilized. Users do not explicitly see a replicated object, they rather directly update the master replica and read specific replicas.

Non-transparent replication in combination with completely autonomous databases implies that databases assign different identities for replicas of the same replicated object or type, respectively.

3.7 Root objects

As we explained in section 2, some objects are explicitly replicated. We call them *root objects*:

Definition 10: *A* ***root object*** *is an object a user deliberately designates to be a source replica for replication into a given destination database.*

For the time being, we define an object o_2 as being reachable from another object o_1 if (o_1,o_2) is in the transitive closure of relationships. A precise definition is given in formula 2 below.

Principle: *Given a root object* o*, all objects which are reachable from* o *are implicitly replicated too.*

In other words, all replicated objects are either root objects themselves, or they are reachable from a root object. In section 3.10, we will present a refined approach that allows a more deliberate selection of objects to be replicated.

To *subscribe* to an object means that it is added to the set of roots of a subscription. To *unsubscribe* from an object signifies that it is removed from the root set.

Definition 11: *A* ***subscription*** *is a triple <source database, destination database, roots>. roots is the set of root objects designated to be replicated from the source database into the destination database.*

3.8 Replication of object and type graphs

The process of implicit replication leads to the construction of a mirrored object graph in the slave database. The object graphs that are spanned by a root object and one of its replicas are isomorphic.

Types are only replicated if they are needed to describe an object replica. Therefore, we establish the following principle for type replication:

Principle: *All types with a non-empty extension of master object replicas are implicitly replicated into the same destination databases as the objects in their extensions.*

The graph of destination type replicas and its corresponding source replica graph are also isomorphic.

We formally model replication graphs as directed, attributed multigraphs. A graph system *GS*, consisting of a master and a slave graph, that conforms to the meta model can thus be defined as follows:

$$GS = \begin{pmatrix} O_M, O_S, Root, R, L_O, r, reach, l_O, rep_O, T_M, \\ T_S, Parent, Rd, L_T, rd, l_T, rep_T, type_M, type_S \end{pmatrix}$$

O_M and O_S, with $O_M \cap O_S = \varnothing$, are the sets of object replicas in the master and the slave graph, respectively. $Root \subseteq O_M$ denotes the set of root objects. R is the set of relationships between objects. L_O stands for the set of relationship attributions. An attribution comprises the relationship name and possible further information, e.g., the cardinality. $r : R \to (O_M \times O_M) \cup (O_S \times O_S)$ is the mapping from relationships to the related objects. The predicate $reach : (O_M \times O_M) \cup (O_S \times O_S) \to \{\top, \bot\}$ is true if the second object is reachable from the first one. $l_O : R \to L_O$ assigns attributions to relationships. $rep_O : O_M \to O_S$ is a bijective function which maps a master to a slave replica object. T_M and T_S, with $T_M \cap T_S = \varnothing$, are the sets of type replicas in the master and the slave graph, respectively. $Parent \subset (T_M \times T_M) \cup (T_S \times T_S)$ is an acyclic relationship. $(t_1, t_2) \in Parent$ denotes that t_1 is a direct supertype of t_2. Rd is the set of relationship declarations. L_T stands for the set of attributions of relationship declarations. $rd : Rd \to (T_M \times T_M) \cup (T_S \times T_S)$ is the mapping from relationship declarations to the related types. $l_T : Rd \to L_T$ assigns attributions to relationship declarations. $rep_T : T_M \to T_S$ is a bijective function that maps a master to a slave replica type. $type_M : O_M \to T_M$ and $type_S : O_S \to T_S$ map objects to their most specific types.

Isomorphism between a master and a slave object graph is specified in formula 1:

$$\forall o_1, o_2 \in O_M : \forall label \in L_O : \begin{pmatrix} \exists rel_M \in R : \\ r(rel_M) = (o_1, o_2) \wedge \\ l_O(rel_M) = label \end{pmatrix} \Leftrightarrow \begin{pmatrix} \exists rel_S \in R : \\ r(rel_S) = (rep_O(o_1), rep_O(o_2)) \wedge \\ l_O(rel_S) = label \end{pmatrix} \quad (1)$$

Formula 2 defines reachability of objects:

$$\forall o_a, o_b \in O_M \cup O_S :$$
$$reach(o_a, o_b) \Leftrightarrow \begin{pmatrix} (\exists rel \in R : r(rel) = (o_a, o_b)) \vee \\ \begin{pmatrix} \exists n \in \mathbb{N} : \exists rel_1, rel_2, \ldots, rel_n, rel_{n+1} \in R : \\ \exists o_1, o_2, \ldots, o_{n-1}, o_n \in O_M \cup O_S : \\ r(rel_1) = (o_a, o_1) \wedge r(rel_2) = (o_1, o_2) \wedge \\ \ldots \wedge \\ r(rel_n) = (o_{n-1}, o_n) \wedge r(rel_{n+1}) = (o_n, o_b) \end{pmatrix} \end{pmatrix} \quad (2)$$

As we stated earlier, master replicas are either roots themselves, or they are reachable by a root object.

$$\forall o \in O_M : o \in Root \vee (\exists o_1 \in Root : reach(o_1, o)) \quad (3)$$

Type graph isomorphism is defined in formulas 4 and 5. First, master replica types and their corresponding slave types contain identical relationship declarations:

$$\forall t_1, t_2 \in T_M : \forall label \in L_T :$$
$$\begin{pmatrix} \exists decl_M \in Rd : \\ rd(decl_M) = (t_1, t_2) \wedge \\ l_T(decl_M) = label \end{pmatrix} \Leftrightarrow \begin{pmatrix} \exists decl_S \in Rd : \\ rd(decl_S) = (rep_T(t_1), rep_T(t_2)) \wedge \\ l_T(decl_S) = label \end{pmatrix} \quad (4)$$

Second, the supertype/subtype graphs between masters and slaves are identical:

$$\forall t_1, t_2 \in T_M : (t_1, t_2) \in Parent \Leftrightarrow (rep_T(t_1), rep_T(t_2)) \in Parent \quad (5)$$

A type is only replicated if its extension contains replicated objects:

$$\forall t \in T_S : \exists o \in O_S :$$
$$type_S(o) = t \vee \exists t_1 \in T_S : type_S(o) = t_1 \wedge (t, t_1) \in <Parent>_t \quad (6)$$

As both the object graphs and the type graphs are isomorphic, the combined graphs are isomorphic too:

$$\forall o_M \in O_M : \forall o_S \in O_S :$$
$$rep_O(o_M) = o_S \Rightarrow rep_T(type_M(o_M)) = type_S(o_S) \quad (7)$$

Figure 6 presents an example. The bold-framed object O11 is a root object, dashed arrows between databases stand for replication. O11 is explicitly replicated as a root object, O12 because it is reachable from O11. O13 is not reachable from a root object and thus remains unreplicated. T11, T12 and T13 have object replicas in their extension and are therefore replicated too.

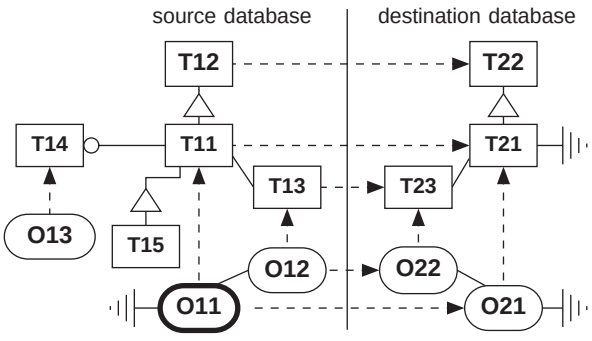

Figure 6: Combined object and type replication.

3.9 The impact of replication on object and type modifications

When a source replica is modified, all or some of the following steps are executed: (1) Determination of the new objects and types that have to be replicated from now on; initial replication of these objects and types. (2) Modification of the slave replica to make it again replica-equivalent to its master. (3) Determination of the types and objects which no longer have to be replicated; termination of replication of these types and objects.

The following modification operations are concerned: (1) Modification of attribute values. (2) Modification of relationships. (3) Specialization and generalization.

The modification of an attribute value in a master replica leads to the same modification of the slaves. The set of objects and types to be replicated remains unchanged.

Relationship modifications have the following consequences: First, the entire object and type graph originating from the new target of the relationship has to be replicated if this is not already the case. Second, the changed relationship in the destination database is set accordingly. Third, it has to be checked for every object and type of the graph that stems from the former target whether it still needs to be replicated; if not, their replication is discontinued.

When an object is generalized, replication for the types below the new type in the inheritance hierarchy only continues if these types still have source replicas in their extensions. Object specialization requires the new type and its supertypes that are more specialized than the former type to be replicated.

Regarding type replication, the following type modification operations are adapted: (1) Insertion and removal of attribute declarations. (2) Insertion and removal of relationship declarations. (3) Creation and deletion of objects.

Whenever a new attribute declaration is added to a source replica or removed from it, an identical declaration is also added to the target replicas or removed from there, respectively.

Insertions and removals of relationship declarations are basically treated in the same way. However, the target type of the relationship declaration is replaced by the corresponding slave type replica.

Object creation is the same as without replication: A new object is never automatically replicated, but only after it either has been determined to be a root object, or has become reachable by a root object.

When a master object is to be deleted, those objects that are reachable from it remain only replicated if they are still reachable by a root object. The object's type and its supertypes only remain replicated if they have further replicated objects in their extensions.

3.10 Restricting the set of objects to be replicated

We defined an object o_2 as being reachable from another object o_1 if (o_1, o_2) is in the transitive closure of relationships. Because this approach potentially leads to a large number of objects to be replicated, we now present an approach to decrease this number.

Formula 2 defined basic object reachability. To restrict replication, we modify this formula by introducing the predicate $traverse : R \rightarrow \{\top, \bot\}$ which tests relationships for the necessity of traversal:

$$\forall o_a, o_b \in O_M \cup O_S : reach(o_a, o_b) \Leftrightarrow$$
$$\begin{pmatrix} (\exists rel \in R : r(rel) = (o_a, o_b) \wedge traverse(rel)) \vee \\ \begin{pmatrix} \exists n \in \mathbb{N} : \exists rel_1, \ldots, rel_{n+1} \in R : \exists o_1, \ldots, o_n \in O_M \cup O_S : \\ r(rel_1) = (o_a, o_1) \wedge traverse(rel_1) \wedge \\ r(rel_2) = (o_1, o_2) \wedge traverse(rel_2) \wedge \ldots \wedge \\ r(rel_n) = (o_{n-1}, o_n) \wedge traverse(rel_n) \wedge \\ r(rel_{n+1}) = (o_n, o_b) \wedge traverse(rel_{n+1}) \end{pmatrix} \end{pmatrix} \quad (8)$$

As the predicate's input is the set of relationships R, a large variety of factors can influence its evaluation, namely, the originating object, the target object, the relationship attributions, or combinations of these factors. Examples of object characteristics that influence replication are the type of an object, the object identifier or the value of an attribute. The predicate could be further enhanced by differentiating between different destination databases or distinct replication tasks.

3.11 Transactional properties

We examined the impact of replication on individual operations. In a concrete system, updates are executed under transactional control. Destination transactions do not necessarily contain an update operation for every corresponding source operation: First, only a subset of the modified objects may be replicated. Second, as we illustrated in definitions 5 and 8, only a subsequence of source changes might be applied at the destination.

4. A REPLICATION INFRASTRUCTURE

Based on the meta model, this section presents a replication infrastructure that becomes an integral part of a DBMS. The infrastructure's basic concepts are described, abstracting from a concrete DBMS. Section 5 will complete this description by presenting the implementation of a real replication system.

We describe the most essential part of the meta model, that is, replication of individual objects and of object graphs.

4.1 Architectural overview

The overall architecture of the replication infrastructure is depicted in figure 7.

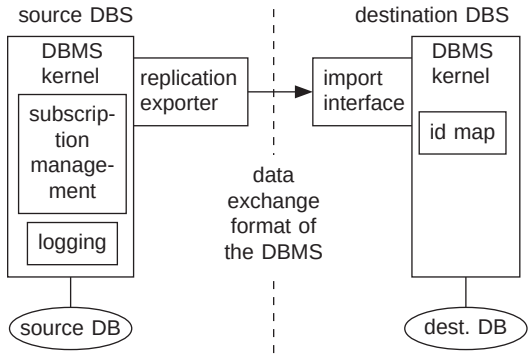

Figure 7: Replication infrastructure, overall architecture.

A DBMS that manages a source database is enhanced by a log facility and a subscription management component. The replication exporter uses these two components to determine all changes of a subscription since the most recent instant of replication of this subscription, and to export these changes in the standard data exchange format of the DBMS. On the destination side, the importer applies these changes to the destination database after replacing source database-specific references by their corresponding target database counterparts.

4.2 Source DBMS components

Figure 8 shows the replication-relevant components of a source DBMS.

A Database manages instances and types, and it controls a subscription manager. Instance stands for objects according to definition 1. Every instance can be updated and queried, and its reachable instances can be determined by recursively getting referenced instances. Type denotes types as introduced in definition 2. They are represented as first-class objects in the sense of type-representatives [4]. As section 5 will show, this allows to treat type replication very similarly to object replication. DbObject serves to factor out logging and other common characteristics of objects and types.

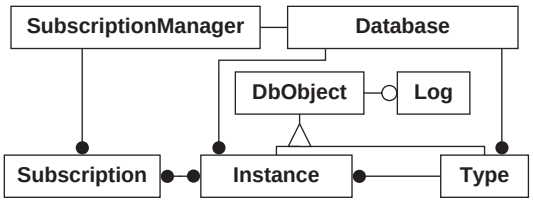

Figure 8: Replication components of a source DBMS.

A source database has one instance of Subscription for every destination database which subscribes to it. A subscription stores the current root objects, and also all objects that belonged to the subscription at the most recent instant of replication. Comparing the latter set (*previous*) to the current instances (*current*) reachable from the root objects allows the subscription to determine the instances which are to be newly replicated ($current \setminus previous$), persist to be replicated ($previous \cap current$), and those whose replication has become obsolete ($previous \setminus current$). Furthermore, a subscription determines for every instance those log entries that have been made after the last instant of replication. Subscriptions are managed by the SubscriptionManager.

A Log is created for every instance which belongs to at least one subscription. Whenever such an instance is modified, a corresponding entry is added to its log. A log entry comprises an identification of the changed property (e.g., the attribute name) and the kind of operation. The new value itself is not stored in the log.

When a subscription is to be replicated, the replication exporter lets the subscription determine the new, persisting and obsolete replicas. For new objects, a create-directive for a target replica and the entire state are exported. For persisting replicas, the log is used to determine changed properties, and those values are then exported. If the log does not reach back far enough, the object's entire state is taken. For obsolete replicas, a delete-directive is generated.

4.3 Destination DBMS components

The relevant components of a destination DBMS are illustrated in figure 9. An instance of IdMap manages the mappings between the source and the corresponding destination replica identifiers.

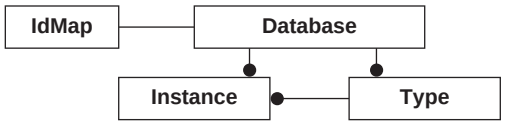

Figure 9: Replication components of a destination DBMS.

First, the creation of new destination replicas is handled. The importer, given the identifier of a source replica, queries the id mapping table and detects that there is no corresponding destination replica. Consequently, it creates a new instance and registers the identifier mapping. This is done for all new replicas, but their states are not set yet.

Second, the states of both the newly created and the persisting destination replicas are set or updated, respectively. Destination replicas are retrieved by using the mapping from

source to destination identifiers. Attribute values of destination replicas are set to the corresponding source values, and relationship targets are mapped to their local counterparts according to the mapping table.

Third, obsolete replicas are deleted, and their mapping table entries are removed.

5. A REPLICATION SYSTEM FOR THE CALANDA TIME SERIES MANAGEMENT SYSTEM

Based on the replication infrastructure described in section 5, we have realized a replication system as a component of Calanda [7, 8], which is a special-purpose DBMS for the management of time series [26]. Calanda is developed and used within UBS, the largest Swiss bank. Its main audiences are economists and financial analysts who want to be able to design and use their time series databases without the help of computing experts. Calanda has been developed on top of ObjectStore [21]. Its object model provides types like time series, group and calendar. A time series consists of a *header*, that is, data characterizing the time series as a whole, and a chronologically ordered sequence of *observations*. Groups allow to partition the set of time series into different, possibly nested, categories. A group comprises a *header* and a set of *members*. As time series and groups are the basic abstractions, instances of these two types can be replicated. The type system presented in figure 8 is thus enhanced as follows: Timeseries and Group inherit from Instance, TimeseriesType and GroupType from Type.

Calanda's replication system is a straightforward realization of the infrastructure which has been presented in the previous section. Therefore, we only describe two specific topics, namely, log processing and an optimized determination of a subscription's member set.

5.1 Log processing

The replication infrastructure is part of a concrete DBMS and thus also operates on a well-defined data model. The knowledge of the frequency and the kind of update operations allows to optimize logging by choosing an appropriate granularity of a log entry. For example, the log does not distinguish between individual header or observation attributes, but simply records that the header or a specific observation has been changed. The reason is that typical applications rarely change the header, and usually update or add entire observations rather than they alter individual observation attributes.

Furthermore, log processing is optimized by knowing the semantics of the update operations. As an example, modifying an observation followed by clearing all observations in a source time series has the net effect of clearing all observations. Therefore, only the clearing operation is relevant for the destination database, the other log entry can be ignored.

5.2 Determination of subscription members

If there are many objects with a large number of relationships between them, the determination of a subscription's member set becomes a costly operation, which should be avoided as often as possible. As an optimization, we only redetermine the members of a subscription if relationships are modified. Experience has shown that only a minority of all operations do this.

5.3 Type replication

The current working system does not implement type replication. However, this functionality is straightforward to add: The sets of new, persisting and obsolete type replicas can be calculated like their object counterparts. Schema evolution corresponds to modifications of type-representative objects. These operations can be logged and processed like instance-modifying operations.

5.4 Experiences

The first productive use of Calanda's replication system within UBS is for a data feed infrastructure that gathers financial data from different source and prepares them for import into data marts. Provider data is converted into Calanda's data exchange format, possibly followed by some transformation steps (for example, provider-specific numeric country codes in a header are replaced by real country names). The resulting data are stored in a Calanda time series base that serves as a staging database. All time series that originate from one provider are placed in an appropriate group structure. Destination databases subscribe to this staging database and add the groups they are interested in to their subscriptions. At the time of writing, this system was operative for half a year and entirely met the users' requirements.

6. RELATED WORK

A variety of different replication protocols can be found in the literature and in working systems. See, for example, [ÖV98], [13], [12], and [17].

Examples for distributed systems that either use replication or provide infrastructure for building replication systems are described in, among others, [3], [19], [23], [15], [28], [14], and [27].

The majority of contemporary DBMS support some kind of replication. Among object DBMS, these are, for example, GemEnterprise [11], Versant [25], Objectivity/DB [20] and ObjectStore [21].

An approach for composite object replication is presented in [1]. Three different replication schemes are specified, namely, type-specific, instance-specific and instance-set-specific replication.

A related area of research is object view maintenance and materialization [16].

A detailed evaluation of various replication protocols, replication systems, and replication functionality in commercially available DBMS can be found in the thesis this paper is based on [9] (http://www.ifi.unizh.ch/ifiadmin/staff/rofrei/Dissertationen/Jahr_1999/thesis_dreyer.pdf). As this evaluation shows, none of the existing protocols and system meets all the requirements presented in section 2.

7. CONCLUSIONS AND OUTLOOK

We have presented an object replication meta model and the design of a replication infrastructure that is well suited for company-wide distribution of financial data aimed at decision-support. The meta model is general enough that it can be applied to a variety of object models. The replication infrastructure only requires minor DBMS enhancements: Source DBMS need a logging facility, a subscription management and a method to determine reachable objects, destination DBMS require an identifier mapping mechanism. Some of these enhancements can also be utilized for other purposes or are even already available: A logging mechanism, for example, can also serve statistical or accounting purposes, and the identifier mapping facility is useful for coping with external data providers who use proprietary object identifiers. (Calanda also uses the mapping tables for this purpose.)

Further work that would enhance the applicability of the developed solution are, for example, the incorporation of schema transformations, the consideration of inter-database object relationships, and the support for peer-to-peer replication with a mechanism that supports different consistency strategies.

The system as presented here can also be used for other purposes than object replication between databases. An example is a notification mechanism: A user of a large database that contains hundreds of thousands of objects is usually only interested in a rather small subset of these data, but these may still be several dozens or hundreds of time series. A notification system that informs this user about changes of objects of interest closely resembles the replication system: Instead of importing the source data changes into a destination database, they can be parsed to generate an appropriate notification which is then mailed to the user. We are currently developing such functionality for another financial database system within UBS.

8. ACKNOWLEDGMENTS

We thank UBS for the partial funding of the Ph.D. studies that led to the results presented here.

9. REFERENCES

[1] B. Bhargava, S. Browne, J. Srinivasan: Composite Object Replication in Distributed Database Systems. Proc. of the 3rd Intl. Conf. on Information Systems and Management of Data, 1992.

[2] P. Bernstein, V. Hadzilacos, N. Goodman: Concurrency Control and Recovery in Database Systems. Addison-Wesley, 1987.

[3] K. Birman, R. van Renesse: Reliable Distributed Computing with the Isis Toolkit. IEEE Computer Society Press, 1994.

[4] R. Cattell: Object Data Management. Addison-Wesley, 1994.

[5] C. Chatfield: The Analysis of Time Series: An Introduction. Chapman & Hall, London, 5th edition, 1996.

[6] K. Dittrich: Object-Oriented Data Model Concepts. Proc. of the NATO Advanced Study Institute on Object-Oriented Database Systems. Springer, 1994.

[7] W. Dreyer, A. Kotz Dittrich, D. Schmidt: Research Perspectives for Time Series Management Systems. SIGMOD Record, 23(1), 1994.

[8] W. Dreyer, A. Kotz Dittrich, D. Schmidt: An Object-Oriented Data Model for a Time Series Management System. Proc. of SSDBM'94, 1994.

[9] W. Dreyer: A Meta Model and an Infrastructure for the Non-Transparent Replication of Object Databases. Ph.D. thesis, University of Zurich, 1999. http://www.ifi.unizh.ch/ifiadmin/staff/rofrei/Dissertationen/Jahr_1999/thesis_dreyer.pdf

[10] W. Dreyer, D. Schmidt, A. Kotz Dittrich, M. Bleichenbacher: Requirements and Design for Replication Services for a Time Series Management System. Proc. of SSDBM'96, 1996.

[11] GemStone Systems, Inc.: GemEnterprise. Product description, 1998.

[12] R. Golding: Weak-Consistency Group Communication and Membership. Ph.D. Thesis, University of California, Santa Cruz, USA, 1992.

[13] A. Helal, A. Heddaya, B. Bhargava: Replication Techniques in Distributed Systems. Kluwer Academic Publishers, 1996.

[14] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, M. West: Scale and Performance in a Distributed File System. ACM TOCS, 6(1), 1988.

[15] IONA Technologies Ltd.: OrbixTalk. White Paper, 1996.

[16] Incremental Maintenance of Materialized Object-Oriented Views in MultiView: Strategies and Performance Evaluation. IEEE Transactions on Knowledge and Data Engineering, 10(5), 1998.

[17] R. Ladin, B. Liskov, L. Shrira, S. Ghemawat: Providing High Availability Using Lazy Replication. ACM TOCS, 10(4), 1992.

[18] L. Lamport: Time, Clocks, and the Ordering of Events in a Distributed System. Communication of the ACM, Vol. 21, No. 7, 1978.

[19] S. Maffeis: Run-Time Support for Object-Oriented Distributed Programming. Ph.D. Thesis, University of Zurich, Switzerland, 1995.

[20] Objectivity, Inc.: Data Replication in Objectivity/DB. White Paper, 1996.

[21] Object Design, Inc.: ObjectStore Management. ObjectStore Release 5.0 Documentation, 1997.

[22] A. Prakash, H. Shim: DistView: Support for Building Efficient Collaborative Applications using Replicated Objects. Proc. of CSCW'94, 1994.

[23] G. Parrington, S. Shrivastava, S. Wheater, M. Little: The Design and Implementation of Arjuna. USENIX Computing Systems Journal, 8(3), 1995.

[24] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen: Object-Oriented Modeling and Design. Prentice-Hall, 1991.

[25] Y.-M. Shyy, H. Au-Yeung, C. Chou: Versant Replication: Supporting Fault-Tolerant Object Databases. Proc. of SIGMOD'95, 1995.

[26] D. Schmidt, A. Kotz Dittrich, W. Dreyer, R. Marti: Time Series, a Neglected Issue in Temporal Database Research?. Proc. of the Intl. Workshop on Temporal Databases, 1995.

[27] M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasaki, E. Siegel, D. Steere: Coda: A Highly Available File System for a Distributed Workstation Environment. IEEE Transactions on Computers, 39(4), 1990.

[28] Tibco, Inc.: TIB/Rendezvous. White Paper, 1998.

[29] D. Terry, M. Theimer, K. Petersen, A. Demers, M. Spreitzer, C. Hauser: Managing Update Conflicts in Bayou, a Weakly Consistent Replicated Storage System. Proc. of the 15th ACM Symposium on Operating System Principles, 1995.