InterBase: A Multidatabase Prototype System*

Omran A. Bukhres, Jiansan Chen, Ahmed K. Elmagarmid,
Xiangning Liu, and James G. Mullen
Department of Computer Sciences, Purdue University
West Lafayette, IN 47907
{bukhres, jchen, ake, xl, jgm}@cs.purdue.edu

1 Introduction

The computing environments of most organizations currently consist of distributed, heterogeneous, and autonomous hardware and software systems. Previously, these systems ran in isolation, supporting their individual applications. However, decreasing network costs made the connection of these distributed systems feasible, and it soon became evident that more complex applications, involving multiple systems, could be supported if the systems could cooperate with each other. The main obstacle to cooperation is local (system) autonomy. That is, it is generally not possible to modify pre-existing systems, and without modification, one generally can have only limited control over the systems.

Several systems that integrate pre-existing systems, especially database systems, have been developed, see for example [1]. However, the focus of most of this work has been on schema integration, and very little work has been done on transaction management. In the InterBase Lab at Purdue University, we have developed a system called InterBase that differs from most multidatabase systems in that it provides facilities for transaction management and support for a flexible transaction model [4] that allows global transactions to execute over heterogeneous database and non-database systems. This paper provides an overview of the InterBase system.

*The InterBase project and system at Purdue University are unrelated to the InterBase product of Borland.

SIGMOD /5/93/Washington, DC,USA
© 1993 ACM 0-89791-592-5/93/0005/0534...$1.50

2 The InterBase System Architecture

The InterBase system architecture is shown in Figure 1, and consists of two main components:

- **Distributed Flex Transaction Manager (DFTM).** The DFTM is responsible for managing global transaction executions in InterBase. Global transactions are specified using the InterBase Parallel Language (IPL) [3].
- **Remote System Interface (RSI).** RSIs are agents of the InterBase system that act as an interface between DFTMs and local systems integrated into InterBase.

Currently, InterBase runs on an interconnected network with a variety of hosts, such as UNIX workstations and IBM mainframes, and supports global applications accessing many local systems including SAS, Sybase, Ingres, DBS, and UNIX utilities. The InterBase System represented in Figure 1 shows example local systems. In general, multiple local systems may run on a given site, and there may be multiple local systems of the same type (e.g. Ingres), although each local system requires its own RSI.

The Distributed Flex Transaction Manager (DFTM) is at the center of InterBase. The DFTM functions are to:

- ensure the reliable execution of global transactions [1];
- manage the data flow within global transactions;
- provide global concurrency control; and
- recover InterBase from errors.

The DFTM consists of a set of DFTM images, with one DFTM image per global transaction.

The general strength of the InterBase architecture is its decentralized nature, which can be seen in Fig-

[1] A global transaction accesses multiple local systems, it consists of several subtransactions, each of which is executed as a local transaction at a single local system. A local transaction accesses only a single local system.

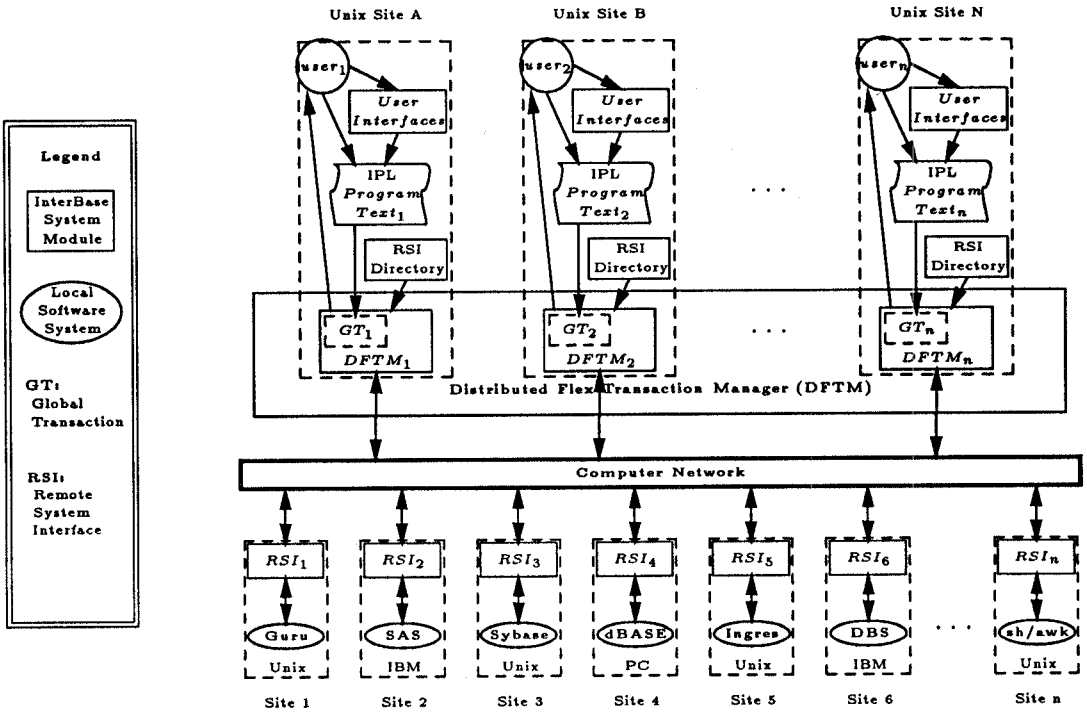

Figure 1: The logical architecture of InterBase

ure 1. InterBase is designed to avoid direct communication among DFTM images, allowing DFTM images to be executed independently, and therefore simplifying the implementation of the DFTM.

The DFTM is distributed over all the machines from which IPL programs are executed; that is, each global transaction G_i is associated with a single DFTM image D_i. G_i usually consists of subtransactions, each of which must be executed on a local system through its associated RSI. In order to provide the correct concurrent execution of global transaction, D_i must first communicate with relevant RSIs to arrange the relative execution order of G_i's subtransactions on corresponding RSIs. The corresponding RSIs then execute G_i's subtransactions in the specified order.

Remote System Interfaces (RSIs) provide a uniform system-level interface between the DFTM and local systems and deal with the heterogeneity of the local systems, including command and data format transformation, thus relieving the DFTM from dealing with each local system directly. The RSI Directory stores information such as location and communication protocols and allowable data transfer methods for different RSIs, thus supporting location and distribution transparency for the system [7].

An RSI consists of an RSI server and RSI services. The RSI server is designed to accept the execution requests of concurrent DFTM images for their associated global transactions and negotiate with the DFTM images to arrange for the execution order of the subtransactions of these global transactions on its associated local systems. It then creates RSI services for these subtransactions according to the specified order. In this way, InterBase allows several DFTM images to be executed concurrently as long as their execution is serializable, thus increasing the throughput of InterBase. An RSI service is responsible for the consistent and reliable execution of the subtransaction and is coincident with its life cycle. The RSI server needs also to trace the status of running and completed RSI services, so as to decide when to schedule the execution of upcoming and queued subtransactions.

While RSI *servers* are local-system-independent since they do not interact directly with local systems, RSI *services* are local-system-specific, since they communicate directly with local systems, and therefore must have knowledge of the language and data formats used by the local system. This allows the RSI server code to be shared by different RSI servers, whereas, the code for RSI services needs to be modified for new local systems. This versatility is an advantage of di-

viding the RSI function between RSI servers and RSI services. A second advantage of this division is that concurrent execution of subtransactions in InterBase is allowed. Another advantage is that all RSI servers can always run on the same platforms as DFTM images no matter where their corresponding RSI services run. This type of RSI structure not only makes communication among DFTM images and RSI servers easy to handle, but also makes it easy for InterBase component crash detection and recovery.

3 The InterBase Parallel Language

The InterBase Parallel Language (IPL) is the transaction specification language of InterBase. IPL allows users to specify all actions associated with a global transaction, such as the control flow and data flow among subtransactions. And, InterBase will automatically execute subtransactions in parallel when it can do so without violating the specified control flow or data flow constraints. As in any traditional transaction processing system, the user is responsible for ensuring the individual correctness of transaction programs. InterBase is responsible for ensuring that transactions execute atomically.

The features of IPL are most effectively illustrated through an example. Suppose a user wishes to transfer money from either his/her checking or savings account to his/her Visa account. Prior to initiating the transfer, the user has not yet determined from which account to withdraw; this will be decided during the transfer. We assume that the bank system is a distributed multidatabase system which maintains all checking accounts in a `Sybase` database system on the machine `sonata`, all savings accounts in an `Ingres` database system on the machine `ector`, and all Visa accounts in a `guru` database system on the machine `interbase`. The IPL program for this application is shown in Figure 2, and we will refer to it throughout our explanation of IPL.

An IPL program contains three parts that respectively are used to specify: definition of data types, definition of subtransactions, and dependency relations among subtransactions.

Definition of Data Types: In IPL programs, a subtransaction is associated with a data type which specifies the data structure of the result if the subtransaction is successfully executed (i.e., reaching its ready-to-commit state). The predefined data types provided are **int**, **real**, **boolean**, and **charString**, and users can define complex types through aggregation. In the example, the subtransactions *checking*, *savings*, and *visa* all produce outputs of the complex type `account`.

Definition of Subtransactions: This part provides mechanisms for specifying subtransactions within an IPL program. In the example, `checking` takes data of the type `inparams` from the input task `in` as its input parameter. The local software system involved is the `sybase` system, which runs on the machine `sonata`. Its execution step, optional confirm step, and optional undo step[2] are defined between IPL keyword pairs `beginexec` and `endexec`, `beginconfirm` and `endconfirm`, and `beginundo` and `endundo`, respectively. These steps, consisting of uninterpreted blocks of statements, are passed through the IPL interpreter to the appropriate RSI. The RSI will translate these statements to statements compatible with its associated local systems, if necessary, before their execution. These statements can be in SQL, the native query language of the local system, or a mixture of the two. If these statements are in SQL, a syntax check will be performed before the execution of the IPL program.

IPL variables are quoted by `$$`. Before the execution of a subtransaction, the IPL variables must be replaced by corresponding values. In the example, both `$$in.amount$$` and `$$in.accNum$$` will be replaced by the corresponding data obtained from the input task `in`. The **select** statement in *checking* is used to produce the needed output. Since IPL keyword **output** is defined for *checking*, its output then becomes a part of the output of the IPL program.

Dependency Description: The dependency description provides users with a mechanism for specifying the execution order among the subtransactions of a global transaction. In the example, the execution order among subtransactions *checking*, *savings*, and *visa* can be described as:

- *checking* and *savings* can be executed without any precondition;
- *visa* can be executed only after either *checking* or *savings* succeeds;
- if *visa* succeeds, the global transaction succeeds.

The dependency description supports *functionally equivalent* subtransaction alternatives for the goals of

[2]The execution step of a subtransaction is executed first. Its confirm step or undo step, if defined, will be executed when the execution of the IPL program is committed or aborted from its prepare-to-commit state, respectively.

```
program
  record inparams of  /* the inputs to the program from the customer */
    accNum : charString;          /* account number */
    amount : integer;             /* the amount of money to be transferred */
  endrecord;

  record account of   /* the schema of bank accounts such as savings */
    name,    accNum,       suffix : charString;
    balance, preBalanace,  amount : integer;
  endrecord;

  /* input task "in" to obtain the inputs of type inparams from the customer */
  input in : inparams endinput;

  subtrans checking (in) : account use sybase at sonata output
    beginexec  /* execution step, in SQL format */
      begin tran sybasebank;
        update bank set amount = -$$in.amount$$ where accNum = $$in.accNum$$;
        update bank set preBalance = balance where accNum = $$in.accNum$$;
        update bank set balance = balance - $$in.amount$$ where accNum = $$in.accNum$$;
        select name, accNum, suffix, balance, preBalanace, amount
          from bank where accNum = $$in.accNum$$;
    endexec
    beginconfirm  /* confirm step, in SQL format, two-phase commit */
      commit tran sybasebank;
    endconfirm
    beginundo  /* undo step, in SQL format, two-phase commit */
      rollback tran sybasebank;
    endundo
  endsubtrans;

  subtrans savings (in) : account use ingres at interbase8 output
    beginexec  /* execution step, in SQL format */
      update bank set preBalance = balance where accNum = $$in.accNum$$;
      update bank set amount = -$$in.amount$$ where accNum = $$in.accNum$$;
      update bank set balance = balance - $$in.amount$$ where accNum = $$in.accNum$$;
      select name, accNum, suffix, balance, preBalanace, amount
        from bank where accNum = $$in.accNum$$;
    endexec
    beginundo  /* undo step, in SQL format, compensation */
      update bank set balance = balance + $$in.amount$$ where accNum = $$in.accNum$$;
    endundo
  endsubtrans;

  subtrans visa (in) : account use guru at interbase output
    beginexec  /* execution step, in SQL format */
      update bank set preBalance = balance where accNum = $$in.accNum$$;
      update bank set amount = -$$in.amount$$ where accNum = $$in.accNum$$;
      update bank set balance = balance - $$in.amount$$ where accNum = $$in.accNum$$;
      select name, accNum, suffix, balance, preBalanace, amount
        from bank where accNum = $$in.accNum$$;
    endexec
    beginundo  /* undo step, in SQL format, compensation */
      update bank set balance = balance + $$in.amount$$ where accNum = $$in.accNum$$;
    endundo
  endsubtrans;

  dependency
    checking or savings : visa;
    visa : accept;
  enddep;

endprogram
```

Figure 2: Example IPL Program

transactions. This flexibility allows an increased tolerance of individual subtransaction failures. In the example, since two subtransaction alternatives (*checking, visa*) and (*savings, visa*) can be deduced, the global transaction can therefore tolerate the failure of either *checking* or *savings*.

A mixed commit approach is used by IPL to facilitate atomicity control of IPL programs. In this example, two-phase commitment is used for *checking*, while compensation is used for both *savings* and *visa*.

The first step of the execution is to check the syntax and semantics of the IPL program. If, for example, there is a syntax error, or a cyclic or recursive dependency[3], or an undefined input to a subtransaction, the execution of the IPL program will be denied.

After the syntax and semantic checks of G_i, a simple execution graph (its nodes represent the subtransactions of G_i and its edges reflects the dependency relations among the subtransactions) is built for G_i. $DFTM_i$ first interacts with the relevant RSIs to arrange the relative execution order for its subtransactions using an algorithm combining the feature of two-phase locking and linear ordering of resource locks[4]. $DFTM_i$ then sends subtransactions whose execution conditions are satisfied to the relevant RSIs for execution. In this example, input task *in* is executed first, then *checking* and *savings*, then *visa*. Relevant RSIs execute the subtransactions in the pre-specified order in their associated sites. The subtransactions are executed until their ready-to-commit states are reached or their executions have failed. The $DFTM_i$ then modifies the execution graph. This process continues until G_i reaches its prepare-to-commit state or fails. In this example, the prepare-to-commit state is signaled by one of the subtransaction alternatives (*checking, visa*) or (*savings, visa*) containing only ready-to-commit subtransactions. The failure is signaled by both subtransaction alternatives (*checking, visa*) and (*savings, visa*) containing a failed subtransaction.

At the prepare-to-commit state, the user is prompted to choose a subtransaction alternative to commit, from a list of alternatives consisting of ready-to-commit subtransactions. $DFTM_i$ then commits those subtransactions selected by the user and undoes those that the user does not want. In the example, if all the three subtransactions are ready-to-commit, both subtransaction alternatives (*checking, visa*) and (*savings, visa*) will be listed for the user to choose. If the user chooses (*checking, visa*), *checking* and *visa* will be committed, while *savings* will be compensated (semantically rolled back). Upon failure, $DFTM_i$ aborts subtransactions at their ready-to-commit state. Throughout, $DFTM_i$ consults the RSI Directory to determine the interface and data transfer protocols for the individual RSIs.

The major advantage of using IPL is its semantic power and suitability. Through dependency description, programmers in IPL can specify control flow among subtransactions, which gives IPL the flexibility to support parallel execution and synchronization among subtransactions. As shown in the example, IPL provides a method for specifying data flow within a global transaction. IPL permits the construction of mixed global transactions by allowing the extent of compensatability to be specified in the declarations of subtransactions. Commit and abort operations of subtransactions are deferred until their global transactions commit or abort, if they are defined, and thus support atomic transactions. IPL also allows the specification of transactions that may include subtransactions that access nondatabase systems, and database systems with complex data models, because statements in the native language can be incorporated into IPL programs.

4 The InterBaseView Graphical User Interface

InterBaseView [5] is our graphical user interface for InterBase and was designed to provide the following features:

- loose coupling to InterBase;
- system environment customization;
- application oriented window generation;
- graphical display of subtransaction dependencies;
- monitoring of transaction execution status;
- intelligent editing facilities;
- database schema display; and
- guidance for new users.

InterBaseView consists of two parts, an *execution interface* and an *IPL program editor*. The execution interface can load an IPL program and invoke it as

[3] Cyclic or recursive dependencies are not allowed, in order to prevent deadlock.

[4] This method begins by numbering all RSIs in an order O with each RSI server maintaining a site-lock. Prior to executing G_i, $DFTM_i$ must first request all necessary site-locks from the relevant RSI servers in an order consistent with O. The relative execution order (REO) of G_i is determined at all relevant sites only when $DFTM_i$ has acquired the necessary site-locks. After the REO of G_i is determined, $DFTM_i$ releases all held site-locks. This method is dead-lock free and totally distributed, while ensuring correct synchronization of concurrent site-lock requests.

a global transaction. The execution history of the transaction is displayed to track its execution status. The execution interface supports Flex transactions by graphically displaying the dependence relations among subtransactions and providing intermediate results, allowing users to select among acceptable alternative subtransactions in committing the transaction. If the user decides to commit the transaction, the final result will be displayed.

The IPL program editor allows a user to conveniently construct IPL programs. After a keyword button is clicked, the keyword is automatically inserted into the edited IPL program at the appropriate position, and help information is simultaneously displayed in the help area leading the user through the process with pop-up menus and dialog windows. Users can always easily undo their last action. The RSI directory is referenced to provide users with the available software system names and the corresponding machine names. All user defined record types and subtransaction names are kept for use in the program. A window displays a sample IPL program, providing an example as an aid to learning. To assist in the writing of applications that presuppose a structural knowledge of the component database systems, InterBaseView provides facilities for users to access these database schemas. After editing an IPL program, a user can check its syntax in the working area before execution.

InterBaseView was implemented using OSF/MOTIF widgets on top of the X window system [6]. The original implementation was on Sun SPARC workstations. Currently, InterBaseView and InterBase are also running on HP 9000/300 workstations at Bell Northern Research Inc., allowing users to develop applications involving the analysis and manipulation of data collected from various databases.

5 Conclusions and Future Work

This paper has described the InterBase multidatabase system which supports global applications in an environment consisting of distributed, heterogeneous, and autonomous software systems. We have presented an overview of the architecture of InterBase, its transaction specification language IPL, and its graphical user interface InterBaseView. A more complete description of the InterBase system will appear in [2].

Although InterBase is far from perfect, we believe that it has the potential to offer an innovative and effective solution to the problems of heterogeneous application program integration. This has been demonstrated through a trial implementation at Bell Northern Research Inc.

We plan to extend InterBase to incorporate more machines and operating system platforms, thus serving a larger community of users. The performance evaluation of InterBase is also being undertaken.

We plan to complete an improved language interface (called InterSQL) for what will be the next-generation InterBase system. The InterSQL interface will provide support for multiple global schema views, an object-oriented common query language, and high level-support for a unified commitment protocol that supports additional multidatabase atomic commitment approaches. We also plan to extend InterBaseView to support the InterSQL interface.

References

[1] M. W. Bright, A. R. Hurson, and S. H. Pakzad. A taxonomy and current issues in multidatabase systems. *Computer*, 25(3):50–60, March 1992.

[2] O. Bukhres, J. Chen, W. Du, A. Elmagarmid, and R. Pezzoli. InterBase : An Execution Environment for Global Applications over Distributed, Heterogeneous, and Autonomous Software Systems. *IEEE Computer*, 1993. (to appear).

[3] J. Chen, O. A. Bukhres, and A. K. Elmagarmid. IPL: A Multidatabase Transaction Specification Language. In *Proc. of the 13th International Conference on Distributed Computing Systems*, 1993. (to appear).

[4] A. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz. A Multidatabase Transaction Model for InterBase. In *Proceedings of the 16th International Conference on Very Large Data Bases*, pages 507–581, Brisbane, Australia, Aug. 1990.

[5] X. Liu, J. Chen, and R. Pezolli. The InterBaseView Graphical User Interface. Technical Report SERC-TR-126-P, Department of Computer Sciences, Purdue University, Nov. 1992.

[6] J. G. R. W. Scheifler. The X Window System. *ACM Transactions on Graphics*, 5(2), 1986.

[7] M. Rusinkiewicz, S. Ostermann, A. Elmagarmid, and K. Loa. The Distributed Operation Language for Specifying Multi-System Applications. In *Proceedings of the First International Conference on System Integration*, pages 337–345, Apr. 1990.