An Execution Model for Distributed Database Transactions and Its Implementation in VPL*

Eva Kühn and Franz Puntigam
University of Technology Vienna
Institute of Computer Languages
Argentinierstr. 8, A-1040 Vienna
Austria, Europe
{eva,franz}@mips.complang.tuwien.ac.at

Ahmed K. Elmagarmid
Indiana Center for Database Systems
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907, USA
ake@cs.purdue.edu

Abstract

We present an execution model for distributed transactions that can be employed for multidatabase systems. We use the Flex Transaction model that has been proposed as a highly general and flexible tool for the specification of distributed transactions and extend it by allowing nested and possibly recursive transaction specifications. We show how a given transaction specification and its execution model can be mapped into a representation in a new concurrent Prolog language, the VPL (Vienna Parallel Logic) language. The representation in VPL can be considered as an executable specification. We show some optimizations concerning this mapping and define a significant subclass of Flex Transactions with a declarative representation in VPL that can be modeled by AND/OR structures. We argue that it is more advantageous to use VPL directly for the transaction specification because it provides more flexibility and more control aspects than the Flex Transaction model.

1 Introduction

A transaction model, called *Flex Transactions* [2, 8], has been proposed for the specification of the control of multidatabase transactions. In particular, this model extends commonly used transaction models by allowing the transaction programmer to release the atomicity and isolation requirements [1]. Function replication is provided in that the goal of a local subtransaction can also be achieved by other subtransactions. This model distinguishes between subtransactions that can be compensated and subtransactions that cannot be compensated after they have committed. Compensatable transactions, also called *sagas* [3], may commit before the global transaction completes and may therefore reduce the lock time in local databases. As compensatable and non-compensatable

*The work is supported by the Austrian FWF (Fonds zur Förderung der Wissenschaftlichen Forschung), project "Interoperability of Autonomous Databases", contract number P7773-PHY.

transactions are supported by the Flex Transaction model, we speak of *mixed transactions*. Flex Transactions specify the control plan, i.e., parallel or sequential execution of transactions due to given dependencies. However, Flex Transactions are not runnable transactions. For the transaction processing in a heterogeneous environment, we need
(a) a specification model for the transaction control (we use Flex Transactions), and
(b) an execution model for transactions.

In Section 2, we extend the Flex Transaction model by allowing nested and recursively defined Flex Transactions. A subtransaction may either be a transaction on a local database or again a Flex Transaction. We present an execution model for this extended definition and prove its correctness in Section 3. This model allows a maximum of parallelism: as soon as all pre-conditions for a subtransaction are fulfilled, this subtransaction may be scheduled for execution. The locking time in local databases is kept as short as possible by immediately committing compensatable transactions after they have succeeded. This feature contributes to the autonomy of local database systems [7]. We assume that the corresponding local database systems provide the two phase commit protocol for non-compensatable transactions.

In [5, 6] we have proposed a new concurrent logic programming language, the VPL (Vienna Parallel Logic) language. This language has been influenced by the family of concurrent Prolog languages [9] and provides explicit language constructs to control the parallelism. VPL differs from existing concurrent Prolog languages in many aspects: VPL is transaction oriented, supports the compensation of committed code, and allows full backtracking. Synchronization and communication in VPL is done by *communication variables*. Section 4 gives a brief survey of VPL.

In Section 5 we show how a Flex Transaction specification can be mapped into a VPL program. This program can be considered as a runnable specification. We prove that the execution of this program is equivalent to the previously defined execution model concerning the produced output and the execution time. Next we prove that VPL's control mechanisms are at least as powerful as the Flex Transaction model. In particular, VPL provides even more mechanisms for the control of distributed transactions. We argue to use VPL directly as the specification language because there exists a significant subclass of Flex Transactions with a highly declarative representation in VPL: they can be modeled by AND/OR structures (Section 6). Thus, VPL can serve for the specification and execution of flexible multidatabase transactions.

2 Flex Transactions and Definitions

We give a definition of nested Flex Transactions and some definitions that we use to prove the correctness of the execution model in Section 3. Our definition of Flex Transactions differs from that in [2] and [8] in that also nested Flex Transactions are considered and only relevant internal dependencies are tested.

Definition 2.1 (*Flex Transaction*). A *Flex Transaction* T is a 5-tuple $(D, \prec_S, \prec_F, \Pi, C)$ where
(a) $D = \{t_1, \ldots, t_n\}$ is a finite set of *typed subtransactions* called the *domain* of T;
(b) the ordering relation $\prec_S$ specifies the *success order* of T (a partial order on D);