A Theory of Advanced Transaction Models in the Situation Calculus

Iluju Kiringa
Department of Computer Science
University of Toronto, Toronto, Canada
kiringai@cs.toronto.edu

June 14, 2001

Abstract

We propose a theory for describing, reasoning about, and implementing transaction models that relax the ACID properties of classical transactions. Such extensions, called *advanced transaction models* (ATMs), are proposed for dealing with applications involving long-lived, endless, and cooperative activities. We specify a given ATM as a non-Markovian theory of the situation calculus, and formulate its properties are formulas of the same calculus. We also consider an implementation of these specifications.

1 Introduction

Various transaction models have been proposed to extend the classical flat transactions by relaxing their ACID (Atomicity-Consistency-Isolation-Durability) properties ([3]). Such extensions, generally called *advanced transaction models* (ATMs), are proposed for applications involving long-lived, endless, and cooperative activities. The ATMs improve the functionality and the performance of these applications.

The ATMs, however, have been proposed in an *ad hoc* fashion, thus lacking in generality in a way that it is not obvious to compare the different ATMs, and to formulate their properties in a way that one clearly sees which new functionality has been added, or which one has been subtracted. To address these questions, there is a need for a general and common framework within which to specify ATMs, specify their properties, and reason about these properties. Thus far, ACTA ([1]) seems to our knowledge the only known framework addressing these questions at a high level of generality. In ACTA, a first order logic-like language is used to capture the semantics of any ATM.

In our on-going work, done jointly with Ray Reiter and based on his development of the situation calculus ([6]), we address the problem of specifying database transactions at the logical level by appealing to non-Markovian theories of the situation calculus. This is a classical second order logic for specifying dynamic domains. Non-Markovian theories allow one to refer to past states other than the previous one.

2 Logical Foundations

We use a *basic relational language*, which is a fragment of the situation calculus ([6]) that is suitable for modeling relational database transactions. The language is a many-sorted second order language with sorts for *actions*, *situations*, and *objects*. *Actions* are first order terms consisting of an action function symbol and its arguments, *situations* are first order terms denoting finite sequences of actions, and *objects* represent domain specific individuals other than actions and situations. In formalizing databases, actions correspond to the elementary database operations of inserting, deleting and updating relational tuples, and situations represent the database *log*. Relations and functions whose truth values vary from situation to situation are called *fluents*, and are denoted by predicate symbols and function symbols with last argument a situation term.

A dynamic domain is axiomatized in the situation calculus with Markovian axioms which describe *how* and under what *conditions* the domain is changing or not changing as a result of performing actions. Such axioms are called basic action theory in [6]. They comprise the following: domain independent foundational axioms for situations; action precondition axioms, one for each action term, stating the conditions of change; successor state axioms, one for each fluent, stating how change occurs; unique names axioms for action terms; and axioms describing the initial situation.

3 Specifying ATMs

In [1], five building blocks for ATMs are identified: *history*, inter-transaction *dependencies*, *visibility* of operations on database objects, *conflict* between operations, and *delegation* of responsibility of objects visible to a transaction. We use these building blocks to represent ATMs in the situation calculus.

In the situation calculus, the history of [1] corresponds to the database log. We extend the basic action theories of [6] to include a specification of relational database transactions, by giving action precondition axioms for transactional actions such as *Begin*, *End*, *Commit*, *Rollback*, *Split*, *Chain*, *Spawn*, etc. *Commit*, and *Rollback* are coercive actions that must occur whenever they are possible. We also give successor state axioms that state how change occurs in databases in the presence of both database updates and the transactional actions. All these axioms are typically non-Markovian; they form *basic relational theories*, which we subsequently use for describing, simulating, and reasoning about ATMs.

Basic relational theories provide the *first dimension* of the situation calculus framework for axiomatizing transactions, namely the axiomatization of the effects of transactions on fluents. Relational theories also comprise axioms indicating which transactions are conflicting with each other, what sublogs of the current log are visible, and which visible sublogs are delegated to the transactions. The *second dimension* of the situation calculus framework is made of dependencies between transactions. All the dependencies expressed in ACTA can also be expressed in our framework. As an example, we have the strong commit dependency of t on t' meaning that if t' commits in a log, then t must also commit in that log. All these dependencies are properties of database logs that are in some way "legal".

To control dependencies that may develop among running transactions, we use a set of predicates denoting these dependencies. For example, we use $sc\_dep(t, t', s)$ to denote the strong commit dependency of t on t' in the log s. These are fluents whose truth value is changed by the relevant transaction models by taking into account dependencies generated by the execution of its external actions and those generated by the execution of its internal actions.

Using this framework, we first formulate the ATMs as relational theories and their properties as formulas of the situation calculus. We then define a legal database log as one whose actions are all possible and in which all the coercive actions must occur whenever they are possible. Finally, we show that the properties of the ATMs, including the relaxed ACID constraints, are properties of legal logs and logical consequences of basic relational theories corresponding to those ATMs.

4 Simulating ATMs

GOLOG ([5]) is a situation calculus-based programming language for defining complex actions in terms of a set of primitive actions axiomatized in the situation calculus. It has the usual control structures of Algol-like languages. Its semantics, however, is defined by macro-expansion, using a ternary relation Do; $Do(prog, S, s')$ is an *abbreviation* for a situation calculus formula whose intuitive meaning is that s' is one of the situations reached by evaluating the GOLOG program $prog$, beginning in a given situation S.

We use the GOLOG language as a transaction language for specifying and simulating ATMs at the logical level by explicitly adding non-Markovian processing abilities to it. To simulate a specific ATM, we first pick the appropriate basic relational theory $\mathcal{D}$ corresponding to that ATM. Then, we write a GOLOG program T expressing the desired transactional behavior. finally, we establish whether $\mathcal{D} \models (\exists s')\ Do(T, S_0, s')$, where S_0 is the initial, empty log.

5 Conclusion

It is important to distinguish between our approach which is a purely logical, abstract specification in which all system properties are formulated relative to the database log, and an implementation which normally materializes the database using progression ([6]). This is the distinguishing feature of our approach. The database log is a first class citizen of the logic, and the semantics of all transaction operations – *Commit*, *Rollback*, etc. – are defined with respect to this log. Our approach goes far beyond ACTA as it is an implementable specification. To that end, the main implementation theorems needed are formulated in [6].

On-going work extending our framework includes: accounting for some of the most recent ATMs, introducing on-line, that is actual execution of transactions, as opposed to off-line or hypothetical execution. We will also consider modeling active rules and different active rule processing mechanisms within the framework of this paper in the near future ([4]).

References

[1] P.K. Chrysanthis. *ACTA, A Framework for Modeling and Reasoning about Extended Transactions.* PhD thesis, Dept. of Comp. and Information Science, Univ. of Massachusetts, Amherst, 1991.

[2] A. Gabaldon. Non-markovian control in the situation calculus. In G. Lakemeyer, editor, *Proc. of the 2nd Int. Cognitive Robotics Workshop*, p. 28–33, Berlin, 2000.

[3] I. Kiringa. *A Formal Account of Relational Active Databases in the Situation Calculus.* PhD thesis, Comp. Science, Univ. of Toronto, Toronto, forthcoming.

[4] H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R.B. Scherl. Golog: A logic programming language for dynamic domains. *J. of Logic Prog.*, 31(1-3):59–83, 1997.

[5] R. Reiter. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems.* MIT Press, Cambridge, 2001.