Appeared in: Harmancı, A. E. and Gelenbe, E. and Örencik, B., editors, Proc. of the 10th Int. Symposium on Computer and Information Sciences (ISCIS X) — Volume I, Kuşadası, Turkey, pages 217–224. Istanbul Technical University, October/November 1995.

Prototyping Object Specifications Using Active Database Systems*

Stefan Conrad Can Türker

Otto-von-Guericke-Universität Magdeburg
Institut für Technische Informationssysteme
Postfach 4120, D–39016 Magdeburg, Germany
E-mail: {conrad|tuerker}@iti.cs.uni-magdeburg.de

Abstract

For prototyping and implementing specified systems, an operational semantics has to be provided for the used specification language. For the specification of information systems ECA-rules being an operational mechanism seem to be a promising basis on which prototyping and implementation can be carried out. Therefore, we present two approaches to giving an operational semantics to an object specification language by generating ECA-rules from object specifications. We discuss several possible choices and their effect on the resulting operational semantics.

Keywords: object specification, operational semantics, active database systems, ECA-rules, prototyping, implementation.

1 Introduction

In the area of object specification the work has been mainly focussed on developing semantic foundations for the specification of objects and their dynamic behavior, see for instance [EDS93]. In connection with this, a number of object specification languages has evolved, e.g. Albert [DDP93], LCM [FW93], TROLL [HSJ+94, JSHS95]. In addition, proving of object properties based on their specification has been investigated and several logics and calculi have been proposed for this purpose (e.g. [CE94, FM91]). In order to achieve a high acceptance for applying object specification techniques in practice, it is necessary to provide a way for implementing systems by using their specification. At least, support for prototyping specifications is needed.

Recently, operational semantics for object specification languages has been considered, e.g. in [Har94] where an operational semantics is presented for a subset of a language which includes a number of highly declarative concepts.

Here, we present another approach to operational semantics which is based on generating ECA-rules from object specifications. The concept of ECA-rules [CBB+89] known from active databases seems to be a promising way for prototyping and implementing object specifications. ECA-rules allow to define state-dependent reactions to events which

*This work is partially supported by the CEC: Esprit Basic Research Working Group No. 8319 ModelAge (A Common Formal **Model** of Cooperating Intelligent **Age**nts).

could lead to critical situations. Especially, they support integrity checking in a flexible and time-dependent way. Examples of activities in the area of active databases are Postgres [SHP88] and Starburst [WF90]. Recently, these active mechanisms have been transferred to and integrated into object-oriented database systems, see for instance the prototype system REACH [BZBW95].

From our point of view it is obvious that mechanisms of active database systems can be used to realize or support prototyping or implementation of object specifications in a rather straightforward way. Above that, an additional motivation for employing ECA-rules is based on the observation that it is often not possible to fully determine the behavior of objects in advance. During the runtime of an information system the behavior of some objects could change due to additional or modified requirements. In such a case the part of the behavior specification realized by ECA-rules could easily be exchanged without disturbing the on-going work of the system. Up to now, object specification languages are not yet able to deal with changing behavior. However, first efforts towards extensions of object specification have been started [SSS95, SCT95].

In this paper we present our ideas for generating ECA-rules from object specifications. We discuss different ways and point out a number of problems which have to be solved in future work.

2 Object Specification: An Example

For discussing the generation of ECA-rules from object specification we introduce an example specification. It is a small universe of discourse consisting of one or more bank objects and account objects belonging to these banks. As specification language we employ the language Troll [HSJ+94, JSHS95] which offers a wide spectrum of modeling concepts for describing structure of objects as well as object dynamics (see also [Saa93]). Here, we focus on a number of basic concepts in order not to overload our presentation.

In Figure 1 Troll specifications for a class of bank objects and a class of account objects are given. For the purposes of this paper we have only specified a small number of properties a bank could have. Here, we assume a bank to have a name, a (unique) bank number, and a set of Accounts which belongs to and are managed by that bank. Typical events which can happen in a bank are the creation of a new account and the transfer of money from one account to another. An account object as instance of the specified Account class has an account number, a bank by which it is managed, a holder, a balance, and a limit for overdrawing. Here, we are interested in events like opening or creating an account, withdrawing or paying money from/into that account.

3 A Naive Translation

In this section, we present a first, naive approach of generating ECA-rules from Troll object specifications. The basic motivation of this approach is an easy and quick prototyping of object specifications. For this purpose it seems quite reasonable to translate specifications as much as possible into ECA-rules. In case the object structure (i.e., classes and attributes and relationships between objects) is stable, we can experiment with different behavior of objects by simply generating ECA-rules which implement the object behavior.

```
object class Bank
  identification  ByBankID: (Name, No) .
  attributes
    Name:      string .
    No:        nat constant
                   restricted No>=10000000 and No<=99999999 .
  components
    Acct:      Account set .
  events
    Open(BankName:string, BankNo:nat)
      birth
      changing Name   := BankName,
               No     := BankNo .
    OpenNewAccount(HID:|Customer|, AN:nat)
      calling   Account(Self, AccountNo).Open(Self, AN, HID)
      changing Acct := insert(Acct, Account(Self, AN).Self) .
    Transfer(AN1:nat, AN2:nat, M:money)
      enabled  in(Account(Self, AN1), Acct) and
               in(Account(Self, AN2), Acct) and
               M > 0.00
      calling   Account(Self, AN1).Withdrawal(M) ,
               Account(Self, AN2).Deposit(M) .
end object class Bank

object class Account
  identification  ByAccountID: (Bank, No) .
  attributes
    No:        nat constant
                   restricted No>=10000000 and No<=99999999 .
    Bank:      |Bank| .
    Holder:    |Customer| .
    Balance:   money initialized 0.00 .
    Limit:     money initialized 0.00
                     restricted Limit<=0.00 .
  events
    Open(BID:|Bank|, AN:nat, HID:|Customer|)
      birth
      changing Bank   := BID,
               No     := AN,
               Holder := HID .
    Withdrawal(W:money)
      enabled  W >= 0.00 and Balance - W >= Limit
      changing Balance := Balance - W .
    Deposit(D:money)
      enabled  D >= 0.00
      changing Balance := Balance + D .
end object class Account
```

Figure 1: TROLL specification of a `Bank` class and an `Account` class.

In order not to stick to the syntax of certain active database systems we use the following abstract notation for ECA-rules:

E: *the triggering event*
C: *the additional condition*
A: *the required (sequence of) actions*

We introduce generic functions (methods) for updating attributes (*set*) and for causing the execution of events in other objects (*call*). Furthermore, a predicate *enabled* is needed for checking whether an event to be called is enabled to be executed at all.

The general idea we present in this section is based on the observation that ECA-rules could be used to realize the dynamic behavior of objects. Assuming that classes can directly be translated into the concepts of the underlying (object-oriented) databases without changing structure or introducing artificial classes, for each event given in the specification of a class, a method is generated which is only used for invoking ECA-rules in case the event shall happen.

The effect which an event is to have on the attributes of an object can be realized by a corresponding ECA-rule. This rule simply reacts on the invocation of this event (i.e., the invocation of the generated method which is named in the event part of the ECA-rule). It checks the enabling condition which is given in the condition part of the ECA-rule. In the action part the required attribute changes are realized. The action part is only executed if the enabling condition is fulfilled.

For an event which can cause other events to be executed by event calling the corresponding methods for those events have to be called in the action section of the ECA-rule.

The mapping of TROLL concepts (or parts of the object specification) to the three parts of an ECA-rule is depicted in the following table (restricted to the concepts occurring in the example):

TROLL concept	mapped to ECA-rule		
	E	**C**	**A**
attributes	—	—	—
— *constant*	—	—	*no attribute change*
— *restricted*	—	*checking event parameters*	*controlled changes*
— *initialized*	—	—	*set*(attribute, ...)
events	triggering event	—	—
— *enabled*	—	enabling condition	—
— *calling*	—	*enabled*(event)	*call*(event/method)
— *changing*	—	—	*set*(attribute, ...)

Considering the specification of objects given in the previous section, we first have to generate ECA-rules changing attributes when an event is occurring. Here, the events `Deposit` and `Withdrawal` for objects of class `Account` are a simple case. The corresponding ECA-rules generated from the specification are the following ones:

E: `Deposit(D)`
C: `D >= 0.00`
A: *set*`(Balance, Balance + D)`

E: `Withdrawal(W)`
C: `W >= 0.00` *and* `Balance - W >= Limit`
A: *set*`(Balance, Balance - W)`

Now, let us have a look at the `Open` event given for objects of class `Account`. Here, the condition that this occurrence of the `Open` event is the first one in the life of the considered object has to be checked. This condition is a constraint inherent to the language TROLL:

E: `Open(BID, AN, HID)`
C: *first occurrence of* `Open` *for this object*
A: *set*`(Balance, 0.00);` *set*`(Limit, 0.00);` *set*`(Bank, BID);` *set*`(No, AN);`
set`(Holder, HID)`

Consider the `OpenNewAccount` event for objects of class `Bank`, such an event involves an event calling:

E: `OpenNewAccount(HID, AN)`
C: *enabled*`(Account(Self, AN).Open(Self, AN, HID))`
A: *call*`(Account(Self, AN).Open(Self, AN, HID));`
set`(Acct, insert(Acct, Account(Self), AN).Self)`

The `Transfer` event of class `Bank` requires an even more complex transformation:

E: `Transfer(AN1, AN2, M)`
C: `in(Account(Self, AN1), Acct)` *and* `in(Account(Self, AN2), Acct)` *and*
`M > 0.00` *and* *enabled*`( Account(Self, AN1).Withdrawal(M))` *and*
enabled`(Account(Self, AN2).Deposit(M))`
A: *call*`(Account(Self, AN1).Withdrawal(M));`
call`(Account(Self, AN2).Deposit(M))`

Of course, the presented generation of ECA-rules from object specifications is a quite naive approach because we abuse ECA-rules for realizing any kind of object manipulation. For a first prototype this might be acceptable. However, such a prototype would become very inefficient.

4 A More Sophisticated Approach

In this section we present a more sophisticated way of generating ECA-rules from object specifications. In contrast to the naive approach presented in Section 3, we now investigate another approach with the goal to derive a more efficient system implementation from object specifications.

In the first approach we have mapped all events of an object specification directly to ECA-rules. From the viewpoint of an efficient system implementation, this approach is unacceptable because of the overhead caused by the rule management. Every time an event happens an appropriate rule manager has to be informed to fire all attached ECA-rules and to control their execution. In other words, after each event at least an additional database access to invoke the relevant rule manager object is required.

A pragmatic way to lower this overhead can be done by reducing the set of possible ECA-rules. To do this, we have to find out which events of an object specification should/must be represented as ECA-rules and for which one a simple coding in methods is sufficient?

In comparison to "hardcoded" methods, ECA-rules seem to be more suitable for modeling dynamic behavior of objects. In fact, ECA-rules have the advantage that additional or modified behavior of objects can be easily integrated into the system at runtime (see active databases like REACH [BZBW95]). Hence, the reaction on a specific event may be changed dynamically by replacing/removing an existing ECA-rule or by integrating a new one. Therefore, we are of the opinion that behavior of objects which cannot be completely specified at compilation/specification time should be expressed by ECA-rules. Especially integrity constraints which may change throughout the lifetime of an system can be realized by ECA-rules in a more adequate way as by usual methods. In consequence, we can state that all "stable" events of an object specification should be integrated into methods, whereas the "dynamic" ones should be mapped onto ECA-rules. Here, we also want to point out that the presented approach is one among many possible reasonable approaches.

According to the discussion above, we now do not formulate ECA-rules for the events `Deposit`, `Withdrawal`, `OpenNewAccount` and `Transfer`. These events are realized by simple methods, e.g. `Deposit(D)` or `Withdrawal(W)`:

```
Deposit(D) {
  if (D > 0.00)
    set(Balance, Balance + D)
}
```

```
Withdrawal(W) {
  if (W > 0.00 and Balance - W >= Limit)
    set(Balance, Balance - W)
}
```

For the event `Open` of class `Account`, however, we generate a method and an additional ECA-rule. In the method all attributes of an account object are initialized. The ECA-rule serves for integrity checking. In our example the corresponding transaction is aborted if the constraint that an account number must lie in a certain interval is not fulfilled. But, there may also be other/further reactions (e.g. printing an error message) possible.

```
Open(BID, AN, HID) {
  set(Balance, 0.00);
  set(Limit, 0.00);
  set(Bank, BID);
  set(No, AN);
  set(Holder, HID)
}
```

```
E: Open(BID, AN, HID)
C: (AN < 1000000) or (AN > 9999999)
A: abort_transaction
```

Please note the advantage of realizing a given integrity constraint in an ECA-rule. If for example in the future all account numbers are assigned to clients, only the corresponding ECA-rule has to be replaced by a new one which allows a greater domain of valid account numbers.

5 Discussion and Conclusions

In the sequel we discuss the presented approaches with regard to a number of choices which could be made in transforming object specifications to a prototype or implementation based on an active database system.

How far should we go in generating ECA-rules? The answer to this question depends on the intention we pursue. For instance, in case we are interested in a prototyping system for object specification for supporting the specification phase, it seems to be acceptable to generate as much as possible ECA-rules from the object specification. This would enable us to validate very quickly the adequacy of a specification by slightly modifying the specification and comparing the resulting prototypes of the specified system.

As already explained for a real implementation it is obviously better not to generate to many ECA-rules. In this case we would try to realize as much as possible directly within methods. In consequence, we have to make a design or implementation decision with regard to the degree of generating ECA-rules from object specifications.

How can we guarantee to preserve the semantics of the specification? Especially for event calling, TROLL assumes that the calling event is only executed if the called event is executed. This means that in case the called event is rejected to be executed the calling event must be rejected as well. In order to capture this semantics both the calling event and the called event must be embedded into one transaction.

By embedding each method into an own (sub-) transaction and assuming ECA-rules always to be executed as a sub-transaction of the transaction in which the triggering event (i.e. method) is executed, we would obtain a system preserving the original TROLL semantics. The same idea of relating methods to (sub-) transactions has been independently employed in [GD95] where it is used for realizing the principle of programming-by-contract. In fact, our work could be understood as a similar approach starting on a more abstract level by using an object specification language instead of a programming language enriched by adding invariants and pre- and post-conditions to methods (procedures).

Alternative Semantics? In order to preserve the semantics of a specification (in TROLL), all ECA-rules have to be executed *immediately* (after the event detection) in *vital* subtransactions [BÖH+92]. If a vital subtransaction is aborted, the parent (sub-) transaction must also be aborted. This may result in cascading aborts which are not desired. But, in some cases an abort of an subtransaction should not prevent a parent transaction to commit. Especially in open heterogeneous environments these further/alternative semantics are often required. Here, ECA-rules provide a flexible and powerful framework by supporting various coupling modes (for more details see [BZBW95]).

The work presented here is based on the assumption that non-dynamic parts of a specification can be directly transferred to the database system in mind. This requires that the structural concepts of the specification language exactly match the corresponding concepts of the database systems. Considering the object specification languages and the object-oriented database systems available today, we have to realize that there is a certain discrepancy with respect to structural concepts (for instance, TROLL offers the concept of roles which has to be simulated by the concepts the database system offers). Moreover, we often need to introduce for instance artificial classes in order to map the object and class structure given in an object specification language to the object and class structure possible in a database system. This is due to the fact that a number of object-oriented database systems can only realize complex values (e.g. sets, lists) by introducing object classes implementing these complex values and their operations. In consequence, we have to take this into account for future work.

Another very appealing aspect for future work is the change of behavior specifications for objects. Often it is not possible to fully determine the behavior of an object at specification time. Especially, in case the requirements are changing during the runtime of the specified information system, it is obviously reasonable to allow changeable behavior for objects. Today's object specification languages cannot treat this aspect. In [SSS95] a first approach to extend the language TROLL is proposed. A more general analysis of this problem can be found in [SCT95].

References

[BÖH+92] A. Buchmann, M. T. Özsu, M. Hornick, D. Georgakopoulos, and F. Manola. A Transaction Model for Active Distributed Object Systems. In A. K. Elmagarmid, editor, *Database*

Transaction Models for Advanced Database Applications, pages 123–151. Morgan Kaufman Publishers, 1992.

[BZBW95] A. P. Buchmann, J. Zimmermann, J. A. Blakeley, and D. L. Wells. Building an Integrated Active OODBMS: Requirements, Architecture, and Design Decisions. In P. S. Yu and A. L. P. Chen, editors, *Proc. of the 11th Int. Conf. on Data Engineering (ICDE'95), Taipei, Taiwan*, pages 117–128. IEEE Computer Society, March 1995.

[CBB+89] S. Chakravathy, B. Blaustein, A. Buchmann, M. Carey, U. Dayal, D. Goldhirsch, M. Hsu, R. Jauhari, R. Ladin, M. Livny, D. McCarthy, R. McKee, and A. Rosenthal. HiPAC: A Research Project in Active, Time-Contrained Database Management. Final Technical Report, Xerox Advanced Information Technology, 1989.

[CE94] S. Conrad and H.-D. Ehrich. An Elementary Logic for Object Specification and Verification. In U. Lipeck and G. Vossen, editors, *Workshop Formale Grundlagen für den Entwurf von Informationssystemen, Tutzing*, pages 197–206. TR-03/94 Univ. Hannover, 1994.

[DDP93] E. Dubois, P. Du Bois, and M. Petit. O-O Requirements Analysis: An Agent Perspective. In O. Nierstrasz, editor, *ECOOP'93 – Object-Oriented Programming (Proc. 7th European Conf., Kaiserslautern)*, pages 458–481. Springer-Verlag, LNCS 707, 1993.

[EDS93] H.-D. Ehrich, G. Denker, and A. Sernadas. Constructing Systems as Object Communities. In M.-C. Gaudel and J.-P. Jouannaud, editors, *Proc. Theory and Practice of Software Development (TAPSOFT'93)*, pages 453–467. Springer, Berlin, LNCS 668, 1993.

[FM91] J. Fiadeiro and T. Maibaum. Towards Object Calculi. In G. Saake and A. Sernadas, editors, *Information Systems – Correctness and Reusability, Workshop IS-CORE '91, ESPRIT BRA WG 3023, London*, pages 129–178. Informatik-Bericht 91–03, Technische Universität Braunschweig, 1991.

[FW93] R. B. Feenstra and R. J. Wieringa. LCM 3.0: A Language for describing Conceptuel Models. Technical Report, Faculty of Mathematics and Computer Science, Vrije Universiteit Amsterdam, 1993.

[GD95] A. Geppert and K. R. Dittrich. Specification and Implementation of Consistency Constraints in Object-Oriented Database Systems: Applying Programming-by-Contract. In *Proc. GI-Fachtagung Datenbanksysteme in Büro, Technik und Wissenschaft (BTW'95), Dresden, Germany*, pages 322–337. Springer-Verlag, March 1995.

[Har94] T. Hartmann. Specification and Execution of Object Interactions in Troll. In U. Lipeck and G. Vossen, editors, *Workshop Formale Grundlagen für den Entwurf von Informationssystemen, Tutzing*, pages 89–95. TR-03/94 Univ. Hannover, 1994. (In German).

[HSJ+94] T. Hartmann, G. Saake, R. Jungclaus, P. Hartel, and J. Kusch. Revised Version of the Modelling Language Troll (Version 2.0). Informatik-Bericht 94–03, Technische Universität Braunschweig, 1994.

[JSHS95] R. Jungclaus, G. Saake, T. Hartmann, and C. Sernadas. Troll – A Language for Object-Oriented Specification of Information Systems. *ACM Transactions on Information Systems*, 1995. *To appear.*

[Saa93] G. Saake. *Object-oriented Specification of Information Systems.* Teubner, Stuttgart/Leipzig, 1993. (In German).

[SCT95] G. Saake, S. Conrad, and C. Türker. From Object Specification towards Agent Design. In *Object-Oriented & Entity-Relationship Modelling — O-O&ER'94 (Proc. of the 14th Int. Conf., Gold Coast, Queensland, Australia)*, December 1995. *To appear.*

[SHP88] M. Stonebraker, E. Hanson, and S. Potamianos. The Postgres Rule Manager. *IEEE Transaction on Software Engineering*, 14(7):897–907, July 1988.

[SSS95] G. Saake, A. Sernadas, and C. Sernadas. Evolving Object Specifications. In R. Wieringa and R. Feenstra, editors, *Information Systems - Correctness and Reusability. Selected Papers.* World Scientific Publishing, 1995.

[WF90] J. Widom and S.J. Finkelstein. Set-Oriented Production Rules in Relational Database Systems. In *Proc. of the Int. Conf. on Management on Data*, pages 259–270, ACM-Press, 1990.