Active Object Oriented Databases in Control Applications

Peter Loborg, Tore Risch, Martin Sköld, Anders Törne

Dept. of Computer and Information Science, Linköping University
S-581 83 Linköping, Sweden
E-mail: petlo, torri, marsk, andto@ida.liu.se

This paper describes a unified architecture for control applications using an extended object-oriented database system with queries and rules. We specify the requirements that control applications demand on the database, and how they are met by our database system architecture. The database system, AMOS, is a main memory database that provides information sharing, powerful data access via an object oriented query language (AMOSQL), data independence, and reactive behavior by active rules. The application considered is a robot and manufacturing instruction system, ARAMIS, which is a task level programming system. The presentation uses a specific scenario related to manufacturing control.

1. INTRODUCTION

Control applications are a significant and important part of the industrial use of information technology . Control may be described as the means to control or restrict the behaviour of external world processes, so that a "correct" or "intended" behaviour is achieved. This might be done by software, specially designed hardware, human intervention or mechanical devices. We will here discuss design and architectural issues with regard to control by software.

Typical for the software design problem in control applications are the requirements originating from timing constraints, some derived from pure control considerations and others from the external processes themselves. These requirements put constraints on the software execution times [7][19]. A lot of research effort has been put into methods and formalisms for timing analysis to guarantee the timing requirements of the system - so called hard real time systems.

This paper will not focus on this aspect, but rather on the *data management* problems which arise when the controlled and controlling systems become large, composite, and complex, and when the controlling subsystem involves human operators.

Traditionally the understanding of control by software focuses on fully automatic control at algorithmic level or at a level close to the external process [23]. At this level the data management problems are small. Usually there is no problem with sharing data between different applications or use of data, since data is local and normally not used outside the local algorithm or control loop. However, as soon as the controlled system becomes larger and more complex, typical data management and information processing problems arise.

The contemporary most popular approach to handling the design of complex software systems is object orientation. A common approach when applying this to control by software is to use object-oriented (OO) modelling

This work is partially supported by the Swedish National Board for Industrial and Technical Development (NUTEK) and by CENIIT, Linköping University.

and programming to structure the software [1][13]. This structure usually reflects the physical structure of the controlled system. Although this aspect of the design is important, the approach ignores the problems with loosely coupled control processes possibly involving several human operators, e.g., nuclear power plants or large manufacturing facilities, which have a high degree of autonomy and parallelism. This extension of the problem domain makes the management problem of the shared data obvious. The present paper focuses on this problem.

One of the important aspects is the separation of shared data and local data for control. All the different control processes share the same data model. However, data independence will be achieved by allowing each process to have their own views of shared data. The shared data model is called a *world model* (WM). These properties are achieved by representing the WM in an OO database system having query and logical data view capabilities.

The use of databases in real-time control systems has recently attained increasing interest. Some of the relevant issues are discussed in [7][9][14][16][17].

We are developing a robotics and manufacturing instruction and runtime system, ARAMIS, for manufacturing applications, and a next generation database architecture, AMOS [4], as a general framework for engineering databases. AMOS has an OO Query Language AMOSQL, which is an extension of OSQL [5]. AMOSQL has transactions, active rules, and extensibility by foreign functions. This paper will discuss a unified approach for control applications, where the world model is represented in an OO database with query and rule capabilities, like AMOS, with ARAMIS as the control application environment.

The next section will give a scenario to be used in the later sections. Section 3 and 4 will present the ARAMIS world model and the AMOS architecture. Section 5 will discuss the unified database architecture for control applications and then we conclude with a general discussion.

2. AN EXAMPLE SCENARIO

In following sections a scenario from a manufacturing application will be used to exemplify the use of active OO databases in control applications.

The task in the scenario is to assemble a subassembly of, for example, an airplane. The resources available are a manipulator and a fixture to perform the assembly itself. Some of the parts needed in the subassembly arrive via a conveyor belt in the order necessary to perform the assembly. These parts are originating from part feeders. Finished subassemblies are placed on a pallet.

The functional requirements on the application is that the assembly should only start if all needed parts are available to the manipulator during the assembly process. If feeder storage is low it should be filled, manually or by automatic guided vehicles, from central storage. Some suboperations of the different processes need preconditions to be fulfilled before starting, e.g., the PICK-UP operation of the manipulator has as a precondition that the part is available in the pick-up position and the conveyor belt is secured (locked).

The task can naturally be decomposed into processes - the assembly process itself involving the manipulator and the fixture, the part transport process for feeding parts onto the conveyor belt and positioning them for the pick-up operation and the central storage fetching process which serves this assembly cell together with many others.

3. THE ARAMIS WORLD MODEL

The ARAMIS system is an instruction and runtime system for control of manufacturing environments. The system has been designed with a layered architecture based on different levels of abstraction [21].

3.1 Description

ARAMIS consists of three different programming levels, *the task level*, *the control* *level* and *the physical level*. The task level specifies what operations should be performed in the physical environment and under what conditions. It uses a graphical rule based hybrid language with primitives for creating parallel execution threads dynamically and synchronization of different parallel threads [12]. The concept corresponding to a task is called a *worker*, which can contain parallel threads by using the parallelisation primitives. The task level program (a set of workers) executes by setting reference values for the objects in a world model (WM). The WM is shared between the different tasks (workers) executing in the physical environment. Basically this corresponds to a blackboard model for communication [8], although we emphasize the database aspects. The model data may be functions or sensor values or they might be implicitly known by the known postconditions resulting from executed actions. One of the benefits of the model is that no difference is made between these forms of knowledge about world state.

The control level is responsible for keeping the real world in a state represented by the world model. This level therefore mimics a *servo mechanism* for the whole system. The programming at this level is typically done by control engineers in a traditional language.

The physical level is the actual connection to the real world, where explicit I/O is performed with sensors and actuators.

One of the important aspects of the world model is that every variable has two values, the *set value* and the *get value*. Set values have the semantics of reference values for the servo mechanism and the get values have the semantics of “known” state parameters - actual values.

3.2 The execution model

Each object in the world model (also called an *active component*) is represented as a deterministic finite automaton (DFA), augmented with control algorithms to be executed for each state transition and state [11]. Each DFA may be in one of two states - transiting between states or being held in a state by the control level. Concurrent transitions between objects are orchestrated by the task level.

The execution model implies that state transitions have a time duration. During the state transition the value of the changing model values are considered as unknown, unless explicit reference to a sensing model is made.

The graphical language for describing tasks is, as was indicated above, able to represent conditional sequencing and possible parallelism of the subactions in each task (worker).

The tasks (workers) model reactive behaviour.

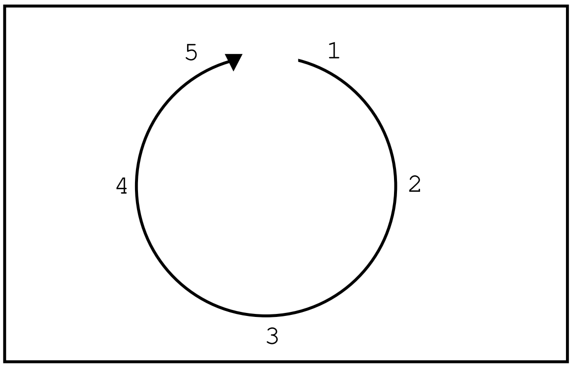

Figure 1. The ARAMIS execution cycle

```
1. external event
2. possible satisfaction of conditions
   in the world model
3. triggering the execution of one or
   several tasks
4. execution of possibly parallel
   subactions
5. requests for possibly concurrent
   state transitions in the WM objects
```

3.3 The scenario data model

In this paper a data model of the scenario in section 2 will be given as an example. It is possible to create the complete corresponding data base model and worker specifications from this description (if completed).

The manipulator, the conveyor belt, the fixture, the feeders and the pallets are represented as active components. Furthermore the parts are active components because they have state and restrictions on their state transitions. However, they only have a passive role for the control process as information carriers in the world model, and are therefore not needed for the following presentation.

Objecttype	**Attribute/ Function**	**Valuetype**
`Workcell`	`assembles`	`Subassembly`
	`transport_of`	`Transport`
	`manipulator_of`	`manipulator`
	`feeder_of`	`Feeder`
	`out_pallet_of`	`Pallet`
`Manipulator`	`state_of`	`{OK, idle, busy}`
`Transport`	`at_pickup_location`	`Part`
	`state_of`	`{locked, moving}`
`Feeder`	`state_of`	`{OK, empty, needs-refill}`
	`feeds_part`	`Part`
`Pallet`	`state_of`	`{OK, empty, full}`

The objects will be instantiated for different configurations of workcells.

3.4 Requirements on the database realization

The ARAMIS model presented briefly above have certain requirements on a database realization:

- Performance is obviously important, since control actions and reactivity should be as timely as possible. Database functionality always imposes overhead performance costs, but they should be minimized. Any control action or reactive action with timing requirements faster than is realizable in the database must be modelled at the control level, thereby losing the benefits of database management.
- The response time of the database must be predictable, i.e., it is not acceptable that the same database operation performs significantly different from time to time. This is not the case if, for example, data access is dependent on whether sought data are available in a buffer or not, as in the case of disk-based DBMS.
- Object oriented modelling and access via a query and data definition language is desirable. This levels the database approach for control applications with OO programming approaches. The query language gives declarative data access functionality.
- Extensibility is required to execute actions and to perform sensing.
- Heterogeneous database access is important when control actions are dependent upon data of traditional character, like in-stock figures or exchange rates.
- The reactive behaviour requirements in control applications demands active database behaviour.
- Transactions and other error recovery mechanisms must be supported.

4. THE AMOS ARCHITECTURE

The AMOS (Active Mediators Object System) architecture [4] uses the *mediator* approach [22], which introduces an intermediate level of software between the database and its use in applications and by users. We call our class of intermediate modules *active mediators*, since they support 'active' database facilities.

The AMOS architecture is built around a *main memory* based platform for intercommunicating object-oriented databases. Each AMOS server has DBMS facilities, such as a local database, a data dictionary, a query processor, transaction processing, remote access to data sources, etc. Main-memory database processing is necessary for control applications in order to have fast and predictable response times. In AMOS the disk is used for background back-up purposes only .

4.1 Object-Oriented Queries

A central component of AMOS is an object-oriented (OO) query language AMOSQL that generalizes OSQL [5]. AMOSQL supports OO abstractions and declarative queries, which makes it possible to declaratively specify different object views for different applications.

The system is extensible through *foreign functions* written in an external programming language (usually Lisp or C), e.g., to access sensor data [16], or to start actuator action.

To support the initial work on AMOS, a main-memory OO DBMS engine is used [10]. A query optimizer translates AMOSQL queries and methods into optimized execution plans in an internal logical language, ObjectLog.

4.2 Active rules based on OO queries

AMOS supports 'active rules' as an extension of OSQL [15]. In an active rule a procedure is executed when the database reaches a specified state. The rules are of the type:

```
when query(parameters)
do exec procedure(parameters)
```

The query in the rule condition can be any AMOSQL query and specifies when the rule should be triggered. The action part can be any AMOSQL database operation.

This type of rules are more powerful than ordinary database triggers or 'ECA ' rules [2], since the entire condition for the triggering of a rule is specified through a declarative query . Rules can be parameterized and overloaded on different types in the database. The execution of rules is made efficient by using incremental computation of rule conditions and by using efficient optimization techniques of the involved queries.

4.3 Transactions

AMOS supports atomic transactions so that database updates are rolled back in case an error occurs. The 'rollback routines' can be programmed to customized clean-up, e.g., to restore the external world to the new state after a rollback of the world model database.

4.4 Heterogeneous database access

A distributed AMOS architecture is being developed where several AMOS servers communicate, and where queries in a multi-database language are allowed to refer to other autonomous AMOS servers, relational databases, sensors, and other data sources [4]. A relational database system, SYBASE, is currently being integrated with AMOS. A particular problem with such an integration is to get OO access to non-OO data sources. The method allows OO queries to be stated with transparent access to non-OO data sources. It will be possible to state queries that combine sensor data with, e.g., conventional databases.

5. A UNIFIED ARCHTECTURE FOR CONTROL APPLICA TIONS

5.1 Declarative modelling/access via OSQL

The representation of the WM of ARAMIS in a database provides powerful data access through the query language. The object oriented data modelling language of AMOS, AMOSQL, has the benefits of a traditional OO language by providing a type system with an inheritance mechanism over subtypes. Furthermore, by accessing the database through object views defined by a query language, data independence between the database and the rest of the system can be achieved, since a data access query can be made looking the same, even after the database structure has changed.

Access to sensors can be implemented as external side-effect free function calls from AMOSQL [16]. This would allow the application programs and/or the operator to state arbitrary complex queries over the current state of the world model - superior to ad hoc navigational database access.

5.2 Active rules in control applications

Active rules in the database can be used for two basic functions in the ARAMIS architecture.

The starting conditions for ARAMIS processes (workers) can be compiled to AMOSQL rules. Starting conditions in AMOS are conditions over the WM, and are thus monitored more efficiently , directly in the database. A starting condition compiled as an AMOSQL rule, awakes the associated process when the starting condition becomes true.

The servo mechanism in the ARAMIS architecture can be implemented as active rules ranging over properties of active components in the WM. The rules can be defined for specific component instances or for whole component classes. The servo mechanism can be implemented by an interplay between ARAMIS actions, AMOSQL rules and control algorithms.

The rules have conditions that are sensitive to state changes of particular active components and actions that calls algorithms in the control system or awakes the ARAMIS inference machine.

The servo mechanism will consist of three phases:

- An ARAMIS process changes the state of the WM and is suspended.
- An AMOSQL rule detects the change and calls a control algorithm.
- The control algorithm executes and changes the physical state of the controlled system to match the state in the WM. Upon completion the rule calls the ARAMIS inference machine to awaken the suspended process.

5.3 Heterogeneous database access in control applications

The distributed and heterogeneous database access capabilities provided by AMOS have the following benefits for ARAMIS:

- Uniform access to heterogeneous data makes it easy to extend the WM with access to conventional databases. It will be possible to state AMOSQL queries combining control and conventional data. For example, error messages can refer to manufacturing data for robot parts, which are accessible from a relational database. Similarly , activity reports can be printed that combine control and relational data.
- Data distribution will make it possible to have geographically distributed ARAMIS systems, each having their own WM views, but also sharing parts of the WM with other ARAMIS processes.
- By generalizing active rules to be distributed over several databases, one may coordinate the behavior of several ARAMIS processes so that WM updates of one process remotely triggers actions in other processes.

5.4 Error detection and recovery

As the ARAMIS system is divided into a task level and a control level, so is the error handling. Furthermore, the error handling (at each level) can be viewed as twofold; handling anticipated errors and unanticipated ones. Different techniques may be used to handle each case.

At the control level each request for a state change (a state transition) is viewed as a transaction. However , using pure transactions as a base for error recovery at this level is not always possible, since there are irreversible state transitions and transitions where the inverse transition is composed of several transitions through some intermediate states. Classifying the transitions as ‘continuable’, ‘undoable’, etc., and augmenting them with extra information can be a possible approach to handle ‘anticipated’ errors [20]. Therefore, the state transition should be modelled as a collections of coupled transactions, e.g., SAGAs [6] or activity models [3]. Some of the transitions do not guarantee that the resulting state is consistent, i.e., there might be transitions that may abort and report an error , as well as those who fails (but leaves everything in a consistent state) and reports the failure. The reasons leading to the abor-

```
create function assembles(Workcell) -> Subassembly;

create function feeder_of(Workcell) -> Feeder;

create function out_pallet_of(Workcell) -> Pallet;

create function parts_of_subassembly(Subassembly) -> bag of Part;

create function state_of(Feeder) -> Charstring;

create function state_of(Pallet) -> Charstring;

create function feeds_part(Feeder) -> Part;

create function ready_parts(Workcell c) -> Part p as
      select p for each Feeder f
      where p = feeds_part(f) and
            f = feeder_of(c) and
            state_of(f) != "empty";

create rule ready_to_go(Workcell c, Worker w) as
      when in(parts_of_subassembly(assembles(c)), ready_parts(c)) and
            state_of(out_pallet_of(c)) != "full"
      do activate_worker(w); /* a procedure that calls ARAMIS */
```

Figure 2. A worker initiation condition modelled by an AMOSQL rule

tion or failure of a transition may be internal (programming errors) as well as external - in the latter case, either detected by the algorithm itself, by operator intervention or by active rules monitoring the state transition (e.g., prevail condition checks).

At the task level active rules can be used as exception handlers, i.e., to handle more or less anticipated errors. For unanticipated errors a combination of manual intervention and planning is required.

5.5 The Scenario

The scenario in section 2 can be implemented as a number of AMOSQL functions and rules. In figure 2 the condition for activation of an assembly worker is monitored by the rule `ready_to_go`. The function `parts_of_subassembly` returns the parts of a particular subassembly . The function `ready_parts` returns all the parts that are ready to be feed onto to the transporter . The condition in the rules checks that all the parts needed for the subassembly are present in the feeders. In figure 3 the condition for the PICK-UP operation is monitored by the rule `ready_to_pickup`. The condition in the rule checks that an object is in the pick-up location on the transporter , that the transporter is locked and that the manipulator is not busy .

```
create function manipulator_of(Workcell) -> Manipulator;

create function transport_of(Workcell) -> Transport;

create function at_pickup_location(Transport) -> Part;

create function state_of(Transport) -> Charstring;

create function state_of(Manipulator) -> Charstring;

create rule ready_to_pickup(Manipulator m) as
      when for each Workcell c, Transport t, Part p where
           state_of(m) != "busy" and
           m = manipulator_of(c) /* find c given m */ and
           t = transport_of(c) and
           p = at_pickup_location(t) and
           state_of(t) = "locked"
      do pickup(m, p); /* activates a control algorithm */
```

Figure 3. An operation invocation condition modelled by an AMOSQL rule

6. DISCUSSION

Our approach has the following advantages vis-a-vis conventional approaches.

The world model may be designed at a very high level using OO abstractions and declarative queries.

The world model is easy to access using OO queries. Sensor data can easily be made accessible from within AMOSQL queries.

Incremental modification of the world model is supported by , e.g., adding new functions, rules, data sources, actuators, etc. A lot of flexibility is gained.

The transaction management of AMOS can be utilized to guarantee atomic updates of the world model, even when much data is updated simultaneously and concurrently . This guarantees world model data consistency after more or less complex updates.

The main-memory representation of the database, guarantees predictable and fast response times.

REFERENCES

[1] T.E. Bihari, P . Gopinath: Object-Oriented Real-T ime Systems: Concepts and Examples, *IEEE Computer* , **25** , 12, 25-32, (1992).

[2] U.Dayal, D.McCarthy: The architecture of an Active Database Management System, *ACM SIGMOD conf.* , 1989, pp. 215-224

[3] U.Dayal, M.Hsu, R.Ladin: Organizing Long Running Activities with T rigger and T ransactions, *Proc. SIGMOD* , May 23-25, Atlanta City , 1990, pp. 204-214.

[4] G. Fahl, T. Risch, M. Sköld: AMOS - An Architecture for Active Mediators, *NGITS'93*, Haifa, Israel, 1993 (to be published)

[5] D.Fishman, et. al: Overview of the Iris DBMS, *Object-Oriented Concepts, Databases, and Applications*, ACM press, Addison-Wesley Publ. Comp., 1989

[6] H.Garcia-Molina, K.Salem: Sagas, *Proc. SIGMOD*, May 27-29, 1987, San Fransisco, pp. 249-259.

[7] M.H. Graham: Issues in Real-Time Data Management, *J. Real-Time Systems*, **4**, 185-202 (1992)

[8] B. Hayes-Roth: A blackboard architecture for control, *Artificial Intelligence*, 26, 251-321 (1985).

[9] J.Huang: Extending Interoperability into the Real-Time Domain, *Research Issues in Data Engineering: Interoperability in Multidatabase Systems, RIDE-IMS'93*, Vienna, Austria, IEEE Computer Society Press, April 1993.

[10] W.Litwin, T.Risch.: Main Memory Oriented Optimization of OO Queries using Typed Datalog with Foreign Predicates, *IEEE Transactions on Knowledge and Data Engineering*, **4**, 6, December 1992

[11] P.Loborg, M.Sköld, A.Törne, P.Holmbom: A Model for the Execution of Task Level Specifications for Intelligent and Flexible Manufacturing Systems, in *Proceedings of the Vth Int. Symposium on Artificial Intelligence, ISAI92*, Cancun, Mexico, dec 1992.

[12] P.Loborg, A.Törne: A Hybrid Language for the Control of Multimachine Environments, in *Proceedings of EIA/AIE-91*, Hawaii, June 1991.

[13] O.Z. Maimon, E.L. Fisher: An Object-Based representation Method for a Manufacturing Cell Controller, *Artificial Intelligence in Engineering*, 1988, **3**, 1, 2-11.

[14] K.Ramamritham: Real-Time Databases, *Distributed and Parallel Databases*, **1**, 2, April 1993

[15] T.Risch, M.Sköld: Active Rules based on Object-Oriented Queries, *IEEE Data Engineering* (Quarterly), January 1993.

[16] R.Snodgrass: A Relational Approach to Monitoring Complex Systems, *ACM Transactions on Computer Systems,* **6**,2, May 1988, pp. 157-196.

[17] S.H.Son: Real Time Database Systems: A New Challenge, *IEEE Data Engineering*, **13**, 4, 51-57 (1990).

[18] J.A. Stankovic, and K. Ramamritham: *Hard Real-Time System*s, Tutorial, IEEE, 1988.

[19] J.A. Stankovic: Misconceptions about Real-time Computing, *Computer*, **21**, 10, 10-19 (1988).

[20] U. Schmidt: A Framework for Automated Error Recovery in FMS, *2:nd Int. Conf. on Automation, Robotics and Computer Vision*, Singapore, 1992.

[21] A.Törne: The Instruction and Control of Multi-Machine Environments, in *Applications of Artificial Intelligence in Engineering V*, vol2, proc. of the 5th Int. Conf., Boston, July 90, Springer-Verlag.

[22] G.Wiederhold: Mediators in the Architecture of Future Information Systems, *IEEE Computer*, March 1992.

[23] K.J.Åström, B.Wittenmark: *Computer Controlled Systems*, Prentice Hall, N.J., 1984.

Active Object Oriented Databases in Control Applications

Peter Loborg, T ore Risch, Martin Sköld, Anders Törne

Dept. of Computer and Information Science, Linköping University
S-581 83 Linköping, Sweden
E-mail: petlo, torri, marsk, andto@ida.liu.se

This paper describes a unified architecture for control applications using an extended object-oriented database system with queries and rules. We specify the requirements that control applications demand on the database, and how they are met by our database system architecture. The database system, AMOS, is a main memory database that provides information sharing, powerful data access via an object oriented query language (AMOSQL), data independence, and reactive behavior by active rules. The application considered is a robot and manufacturing instruction system, ARAMIS, which is a task level programming system. The presentation uses a specific scenario related to manufacturing control.

1. INTRODUCTION

Control applications are a significant and important part of the industrial use of information technology . Control may be described as the means to control or restrict the behaviour of external world processes, so that a "correct" or "intended" behaviour is achieved. This might be done by software, specially designed hardware, human intervention or mechanical devices. W e will here discuss design and architectural issues with regard to control by software.

Typical for the software design problem in control applications are the requirements originating from timing constraints, some derived from pure control considerations and others from the external processes themselves. These requirements put constraints on the software execution times [7][19]. A lot of research effort has been put into methods and formalisms for timing analysis to guarantee the timing requirements of the system - so called hard real time systems.

This paper will not focus on this aspect, but rather on the *data management* problems which arise when the controlled and controlling systems become large, composite, and complex, and when the controlling subsystem involves human operators.

Traditionally the understanding of control by software focuses on fully automatic control at algorithmic level or at a level close to the external process [23]. At this level the data management problems are small. Usually there is no problem with sharing data between different applications or use of data, since data is local and normally not used outside the local algorithm or control loop. However, as soon as the controlled system becomes larger and more complex, typical data management and information processing problems arise.

The contemporary most popular approach to handling the design of complex software systems is object orientation. A common approach when applying this to control by software is to use object-oriented (OO) modelling

This work is partially supported by the Swedish National Board for Industrial and Technical Development (NUTEK) and by CENIIT, Linköping University.

and programming to structure the software [1][13]. This structure usually reflects the physical structure of the controlled system. Although this aspect of the design is important, the approach ignores the problems with loosely coupled control processes possibly involving several human operators, e.g., nuclear power plants or large manufacturing facilities, which have a high degree of autonomy and parallelism. This extension of the problem domain makes the management problem of the shared data obvious. The present paper focuses on this problem.

One of the important aspects is the separation of shared data and local data for control. All the different control processes share the same data model. However, data independence will be achieved by allowing each process to have their own views of shared data. The shared data model is called a *world model* (WM). These properties are achieved by representing the WM in an OO database system having query and logical data view capabilities.

The use of databases in real-time control systems has recently attained increasing interest. Some of the relevant issues are discussed in [7][9][14][16][17].

We are developing a robotics and manufacturing instruction and runtime system, ARAMIS, for manufacturing applications, and a next generation database architecture, AMOS [4], as a general framework for engineering databases. AMOS has an OO Query Language AMOSQL, which is an extension of OSQL [5]. AMOSQL has transactions, active rules, and extensibility by foreign functions. This paper will discuss a unified approach for control applications, where the world model is represented in an OO database with query and rule capabilities, like AMOS, with ARAMIS as the control application environment.

The next section will give a scenario to be used in the later sections. Section 3 and 4 will present the ARAMIS world model and the AMOS architecture. Section 5 will discuss the unified database architecture for control applications and then we conclude with a general discussion.

2. AN EXAMPLE SCENARIO

In following sections a scenario from a manufacturing application will be used to exemplify the use of active OO databases in control applications.

The task in the scenario is to assemble a subassembly of, for example, an airplane. The resources available are a manipulator and a fixture to perform the assembly itself. Some of the parts needed in the subassembly arrive via a conveyor belt in the order necessary to perform the assembly. These parts are originating from part feeders. Finished subassemblies are placed on a pallet.

The functional requirements on the application is that the assembly should only start if all needed parts are available to the manipulator during the assembly process. If feeder storage is low it should be filled, manually or by automatic guided vehicles, from central storage. Some suboperations of the different processes need preconditions to be fulfilled before starting, e.g., the PICK-UP operation of the manipulator has as a precondition that the part is available in the pick-up position and the conveyor belt is secured (locked).

The task can naturally be decomposed into processes - the assembly process itself involving the manipulator and the fixture, the part transport process for feeding parts onto the conveyor belt and positioning them for the pick-up operation and the central storage fetching process which serves this assembly cell together with many others.

3. THE ARAMIS WORLD MODEL

The ARAMIS system is an instruction and runtime system for control of manufacturing environments. The system has been designed with a layered architecture based on different levels of abstraction [21].

3.1 Description

ARAMIS consists of three different programming levels, *the task level*, *the control* *level* and *the physical level*. The task level specifies what operations should be performed in the physical environment and under what conditions. It uses a graphical rule based hybrid language with primitives for creating parallel execution threads dynamically and synchronization of different parallel threads [12]. The concept corresponding to a task is called a *worker*, which can contain parallel threads by using the parallelisation primitives. The task level program (a set of workers) executes by setting reference values for the objects in a world model (WM). The WM is shared between the different tasks (workers) executing in the physical environment. Basically this corresponds to a blackboard model for communication [8], although we emphasize the database aspects. The model data may be functions or sensor values or they might be implicitly known by the known post-conditions resulting from executed actions. One of the benefits of the model is that no difference is made between these forms of knowledge about world state.

The control level is responsible for keeping the real world in a state represented by the world model. This level therefore mimics a *servo mechanism* for the whole system. The programming at this level is typically done by control engineers in a traditional language.

The physical level is the actual connection to the real world, where explicit I/O is performed with sensors and actuators.

One of the important aspects of the world model is that every variable has two values, the *set value* and the *get value*. Set values have the semantics of reference values for the servo mechanism and the get values have the semantics of "known" state parameters - actual values.

3.2 The execution model

Each object in the world model (also called an *active component*) is represented as a deterministic finite automaton (DFA), augmented with control algorithms to be executed for each state transition and state [11]. Each DFA may be in one of two states - transiting between states or being held in a state by the control level. Concurrent transitions between objects are orchestrated by the task level.

The execution model implies that state transitions have a time duration. During the state transition the value of the changing model values are considered as unknown, unless explicit reference to a sensing model is made.

The graphical language for describing tasks is, as was indicated above, able to represent conditional sequencing and possible parallelism of the subactions in each task (worker).

The tasks (workers) model reactive behaviour.

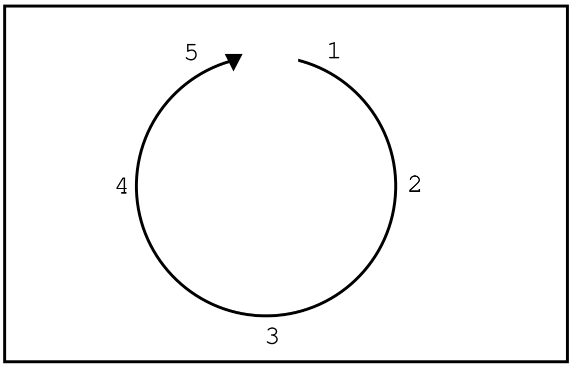

Figure 1. The ARAMIS execution cycle

```
1. external event
2. possible satisfaction of conditions
   in the world model
3. triggering the execution of one or
   several tasks
4. execution of possibly parallel
   subactions
5. requests for possibly concurrent
   state transitions in the WM objects
```

3.3 The scenario data model

In this paper a data model of the scenario in section 2 will be given as an example. It is possible to create the complete corresponding data base model and worker specifications from this description (if completed).

The manipulator, the conveyor belt, the fixture, the feeders and the pallets are represented as active components. Furthermore the parts are active components because they have state and restrictions on their state transitions. However, they only have a passive role for the control process as information carriers in the world model, and are therefore not needed for the following presentation.

Objecttype	Attribute/ Function	Valuetype
Workcell	assembles	Subassembly
	transport_of	Transport
	manipulator_of	manipulator
	feeder_of	Feeder
	out_pallet_of	Pallet
Manipulator	state_of	{OK, idle, busy}
Transport	at_pickup_location	Part
	state_of	{locked, moving}
Feeder	state_of	{OK, empty, needs-refill}
	feeds_part	Part
Pallet	state_of	{OK, empty, full}

The objects will be instantiated for different configurations of workcells.

3.4 Requirements on the database realization

The ARAMIS model presented briefly above have certain requirements on a database realization:

- Performance is obviously important, since control actions and reactivity should be as timely as possible. Database functionality always imposes overhead performance costs, but they should be minimized. Any control action or reactive action with timing requirements faster than is realizable in the database must be modelled at the control level, thereby losing the benefits of database management.
- The response time of the database must be predictable, i.e., it is not acceptable that the same database operation performs significantly different from time to time. This is not the case if, for example, data access is dependent on whether sought data are available in a buffer or not, as in the case of disk-based DBMS.
- Object oriented modelling and access via a query and data definition language is desirable. This levels the database approach for control applications with OO programming approaches. The query language gives declarative data access functionality.
- Extensibility is required to execute actions and to perform sensing.
- Heterogeneous database access is important when control actions are dependent upon data of traditional character, like in-stock figures or exchange rates.
- The reactive behaviour requirements in control applications demands active database behaviour.
- Transactions and other error recovery mechanisms must be supported.

4. THE AMOS ARCHITECTURE

The AMOS (Active Mediators Object System) architecture [4] uses the *mediator* approach [22], which introduces an intermediate level of software between the database and its use in applications and by users. We call our class of intermediate modules *active mediators*, since they support 'active' database facilities.

The AMOS architecture is built around a *main memory* based platform for intercommunicating object-oriented databases. Each AMOS server has DBMS facilities, such as a local database, a data dictionary, a query processor, transaction processing, remote access to data sources, etc. Main-memory database processing is necessary for control applications in order to have fast and predictable response times. In AMOS the disk is used for background back-up purposes only .

4.1 Object-Oriented Queries

A central component of AMOS is an object-oriented (OO) query language AMOSQL that generalizes OSQL [5]. AMOSQL supports OO abstractions and declarative queries, which makes it possible to declaratively specify different object views for different applications.

The system is extensible through *foreign functions* written in an external programming language (usually Lisp or C), e.g., to access sensor data [16], or to start actuator action.

To support the initial work on AMOS, a main-memory OO DBMS engine is used [10]. A query optimizer translates AMOSQL queries and methods into optimized execution plans in an internal logical language, ObjectLog.

4.2 Active rules based on OO queries

AMOS supports 'active rules' as an extension of OSQL [15]. In an active rule a procedure is executed when the database reaches a specified state. The rules are of the type:

```
 when query(parameters)
do exec procedure(parameters)
```

The query in the rule condition can be any AMOSQL query and specifies when the rule should be triggered. The action part can be any AMOSQL database operation.

This type of rules are more powerful than ordinary database triggers or 'ECA ' rules [2], since the entire condition for the triggering of a rule is specified through a declarative query . Rules can be parameterized and overloaded on different types in the database. The execution of rules is made efficient by using incremental computation of rule conditions and by using efficient optimization techniques of the involved queries.

4.3 Transactions

AMOS supports atomic transactions so that database updates are rolled back in case an error occurs. The 'rollback routines' can be programmed to customized clean-up, e.g., to restore the external world to the new state after a rollback of the world model database.

4.4 Heterogeneous database access

A distributed AMOS architecture is being developed where several AMOS servers communicate, and where queries in a multi-database language are allowed to refer to other autonomous AMOS servers, relational databases, sensors, and other data sources [4]. A relational database system, SYBASE, is currently being integrated with AMOS. A particular problem with such an integration is to get OO access to non-OO data sources. The method allows OO queries to be stated with transparent access to non-OO data sources. It will be possible to state queries that combine sensor data with, e.g., conventional databases.

5. A UNIFIED ARCHTECTURE FOR CONTROL APPLICA TIONS

5.1 Declarative modelling/access via OSQL

The representation of the WM of ARAMIS in a database provides powerful data access through the query language. The object oriented data modelling language of AMOS, AMOSQL, has the benefits of a traditional OO language by providing a type system with an inheritance mechanism over subtypes. Furthermore, by accessing the database through object views defined by a query language, data independence between the database and the rest of the system can be achieved, since a data access query can be made looking the same, even after the database structure has changed.

Access to sensors can be implemented as external side-effect free function calls from AMOSQL [16]. This would allow the application programs and/or the operator to state arbitrary complex queries over the current state of the world model - superior to ad hoc navigational database access.

5.2 Active rules in control applications

Active rules in the database can be used for two basic functions in the ARAMIS architecture.

The starting conditions for ARAMIS processes (workers) can be compiled to AMOSQL rules. Starting conditions in AMOS are conditions over the WM, and are thus monitored more efficiently , directly in the database. A starting condition compiled as an AMOSQL rule, awakes the associated process when the starting condition becomes true.

The servo mechanism in the ARAMIS architecture can be implemented as active rules ranging over properties of active components in the WM. The rules can be defined for specific component instances or for whole component classes. The servo mechanism can be implemented by an interplay between ARAMIS actions, AMOSQL rules and control algorithms.

The rules have conditions that are sensitive to state changes of particular active components and actions that calls algorithms in the control system or awakes the ARAMIS inference machine.

The servo mechanism will consist of three phases:

- An ARAMIS process changes the state of the WM and is suspended.
- An AMOSQL rule detects the change and calls a control algorithm.
- The control algorithm executes and changes the physical state of the controlled system to match the state in the WM. Upon completion the rule calls the ARAMIS inference machine to awaken the suspended process.

5.3 Heterogeneous database access in control applications

The distributed and heterogeneous database access capabilities provided by AMOS have the following benefits for ARAMIS:

- Uniform access to heterogeneous data makes it easy to extend the WM with access to conventional databases. It will be possible to state AMOSQL queries combining control and conventional data. For example, error messages can refer to manufacturing data for robot parts, which are accessible from a relational database. Similarly , activity reports can be printed that combine control and relational data.
- Data distribution will make it possible to have geographically distributed ARAMIS systems, each having their own WM views, but also sharing parts of the WM with other ARAMIS processes.
- By generalizing active rules to be distributed over several databases, one may coordinate the behavior of several ARAMIS processes so that WM updates of one process remotely triggers actions in other processes.

5.4 Error detection and recovery

As the ARAMIS system is divided into a task level and a control level, so is the error handling. Furthermore, the error handling (at each level) can be viewed as twofold; handling anticipated errors and unanticipated ones. Different techniques may be used to handle each case.

At the control level each request for a state change (a state transition) is viewed as a transaction. However , using pure transactions as a base for error recovery at this level is not always possible, since there are irreversible state transitions and transitions where the inverse transition is composed of several transitions through some intermediate states. Classifying the transitions as ‘continuable’, ‘undoable’, etc., and augmenting them with extra information can be a possible approach to handle ‘anticipated’ errors [20]. Therefore, the state transition should be modelled as a collections of coupled transactions, e.g., SAGAs [6] or activity models [3]. Some of the transitions do not guarantee that the resulting state is consistent, i.e., there might be transitions that may abort and report an error , as well as those who fails (but leaves everything in a consistent state) and reports the failure. The reasons leading to the abor-

```
create function assembles(Workcell) -> Subassembly;

create function feeder_of(Workcell) -> Feeder;

create function out_pallet_of(Workcell) -> Pallet;

create function parts_of_subassembly(Subassembly) -> bag of Part;

create function state_of(Feeder) -> Charstring;

create function state_of(Pallet) -> Charstring;

create function feeds_part(Feeder) -> Part;

create function ready_parts(Workcell c) -> Part p as
      select p for each Feeder f
      where p = feeds_part(f) and
            f = feeder_of(c) and
            state_of(f) != "empty";

create rule ready_to_go(Workcell c, Worker w) as
      when in(parts_of_subassembly(assembles(c)), ready_parts(c)) and
            state_of(out_pallet_of(c)) != "full"
      do activate_worker(w); /* a procedure that calls ARAMIS */
```

Figure 2. A worker initiation condition modelled by an AMOSQL rule

tion or failure of a transition may be internal (programming errors) as well as external - in the latter case, either detected by the algorithm itself, by operator intervention or by active rules monitoring the state transition (e.g., prevail condition checks).

At the task level active rules can be used as exception handlers, i.e., to handle more or less anticipated errors. For unanticipated errors a combination of manual intervention and planning is required.

5.5 The Scenario

The scenario in section 2 can be implemented as a number of AMOSQL functions and rules. In figure 2 the condition for activation of an assembly worker is monitored by the rule `ready_to_go`. The function `parts_of_subassembly` returns the parts of a particular subassembly . The function `ready_parts` returns all the parts that are ready to be feed onto to the transporter . The condition in the rules checks that all the parts needed for the subassembly are present in the feeders. In figure 3 the condition for the PICK-UP operation is monitored by the rule `ready_to_pickup`. The condition in the rule checks that an object is in the pick-up location on the transporter , that the transporter is locked and that the manipulator is not busy .

```
create function manipulator_of(Workcell) -> Manipulator;

create function transport_of(Workcell) -> Transport;

create function at_pickup_location(Transport) -> Part;

create function state_of(Transport) -> Charstring;

create function state_of(Manipulator) -> Charstring;

create rule ready_to_pickup(Manipulator m) as
      when for each Workcell c, Transport t, Part p where
           state_of(m) != "busy" and
           m = manipulator_of(c) /* find c given m */ and
           t = transport_of(c) and
           p = at_pickup_location(t) and
           state_of(t) = "locked"
      do pickup(m, p); /* activates a control algorithm */
```

Figure 3. An operation invocation condition modelled by an AMOSQL rule

6. DISCUSSION

Our approach has the following advantages vis-a-vis conventional approaches.

The world model may be designed at a very high level using OO abstractions and declarative queries.

The world model is easy to access using OO queries. Sensor data can easily be made accessible from within AMOSQL queries.

Incremental modification of the world model is supported by , e.g., adding new functions, rules, data sources, actuators, etc. A lot of flexibility is gained.

The transaction management of AMOS can be utilized to guarantee atomic updates of the world model, even when much data is updated simultaneously and concurrently . This guarantees world model data consistency after more or less complex updates.

The main-memory representation of the database, guarantees predictable and fast response times.

REFERENCES

[1] T.E. Bihari, P . Gopinath: Object-Oriented Real-T ime Systems: Concepts and Examples, *IEEE Computer* , **25**, 12, 25-32, (1992).

[2] U.Dayal, D.McCarthy: The architecture of an Active Database Management System, *ACM SIGMOD conf.* , 1989, pp. 215-224

[3] U.Dayal, M.Hsu, R.Ladin: Organizing Long Running Activities with T rigger and T ransactions, *Proc. SIGMOD* , May 23-25, Atlanta City , 1990, pp. 204-214.

[4] G. Fahl, T. Risch, M. Sköld: AMOS - An Architecture for Active Mediators, *NGITS'93*, Haifa, Israel, 1993 (to be published)

[5] D.Fishman, et. al: Overview of the Iris DBMS, *Object-Oriented Concepts, Databases, and Applications*, ACM press, Addison-Wesley Publ. Comp., 1989

[6] H.Garcia-Molina, K.Salem: Sagas, *Proc. SIGMOD*, May 27-29, 1987, San Fransisco, pp. 249-259.

[7] M.H. Graham: Issues in Real-Time Data Management, *J. Real-Time Systems*, **4**, 185-202 (1992)

[8] B. Hayes-Roth: A blackboard architecture for control, *Artificial Intelligence*, 26, 251-321 (1985).

[9] J.Huang: Extending Interoperability into the Real-Time Domain, *Research Issues in Data Engineering: Interoperability in Multidatabase Systems, RIDE-IMS'93*, Vienna, Austria, IEEE Computer Society Press, April 1993.

[10] W.Litwin, T.Risch.: Main Memory Oriented Optimization of OO Queries using Typed Datalog with Foreign Predicates, *IEEE Transactions on Knowledge and Data Engineering*, **4**, 6, December 1992

[11] P.Loborg, M.Sköld, A.Törne, P.Holmbom: A Model for the Execution of Task Level Specifications for Intelligent and Flexible Manufacturing Systems, in *Proceedings of the Vth Int. Symposium on Artificial Intelligence, ISAI92*, Cancun, Mexico, dec 1992.

[12] P.Loborg, A.Törne: A Hybrid Language for the Control of Multimachine Environments, in *Proceedings of EIA/AIE-91*, Hawaii, June 1991.

[13] O.Z. Maimon, E.L. Fisher: An Object-Based representation Method for a Manufacturing Cell Controller, *Artificial Intelligence in Engineering*, 1988, **3**, 1, 2-11.

[14] K.Ramamritham: Real-Time Databases, *Distributed and Parallel Databases*, **1**, 2, April 1993

[15] T.Risch, M.Sköld: Active Rules based on Object-Oriented Queries, *IEEE Data Engineering* (Quarterly), January 1993.

[16] R.Snodgrass: A Relational Approach to Monitoring Complex Systems, *ACM Transactions on Computer Systems,* **6**,2, May 1988, pp. 157-196.

[17] S.H.Son: Real Time Database Systems: A New Challenge, *IEEE Data Engineering*, **13**, 4, 51-57 (1990).

[18] J.A. Stankovic, and K. Ramamritham: *Hard Real-Time System*s, Tutorial, IEEE, 1988.

[19] J.A. Stankovic: Misconceptions about Real-time Computing, *Computer*, **21**, 10, 10-19 (1988).

[20] U. Schmidt: A Framework for Automated Error Recovery in FMS, *2:nd Int. Conf. on Automation, Robotics and Computer Vision*, Singapore, 1992.

[21] A.Törne: The Instruction and Control of Multi-Machine Environments, in *Applications of Artificial Intelligence in Engineering V*, vol2, proc. of the 5th Int. Conf., Boston, July 90, Springer-Verlag.

[22] G.Wiederhold: Mediators in the Architecture of Future Information Systems, *IEEE Computer*, March 1992.

[23] K.J.Åström, B.Wittenmark: *Computer Controlled Systems*, Prentice Hall, N.J., 1984.