USING AN EMBEDDED ACTIVE DATABASE IN A CONTROL SYSTEM ARCHITECTURE

Falkenroth, E.T., Risch, T., Törne A.

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

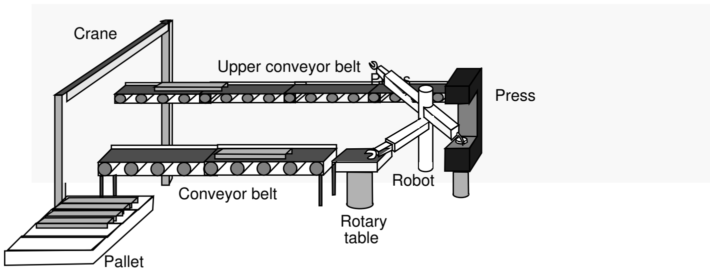
ABSTRACT

Typical data management problems arise in control applications when the controlled environment becomes complex and large volumes of data are involved. This paper addresses this problem by embedding an active object-relational database in a control system architecture. The database stores an abstract model of the controlled environment. A control application language, tightly integrated with the query processor, is used for high-level specification of operations. A set of control algorithms running on a separate real-time kernel performs the actual closed loop control of the external environment. The primitive operation of the control application language is an update of the database that will trigger the control algorithms. In this way our architecture combines cyclic control algorithms and an event driven operation language with data management capabilities. The integrated architecture makes extensive use of the active rule facilities in the database, both in the execution of the control application language and to initiate appropriate control algorithms in the real-time kernel. We discuss practical experiences of building a unified control system with tightly integrated queries and active rules.

INTRODUCTION

A computer based control system must sense the environment and directly influence it through actions. Such systems are subject to time constraints related to the environment in which they operate.
For instance, failures, like dropped objects, may occur unless a conveyor belt is stopped fast enough. Real-time system research has focused on developing mechanisms to support predictable execution of periodic control tasks with minor data dependencies [Bur90]. These traditional real-time instruction formalisms are not well suited for composite control systems that involve large and more complex data sets [Gra92] [Lob93] [Lob94]. This paper presents an implementation of a control system architecture, **`CAMOS`**, which combines traditional control algorithms and high level operations using an embedded object-relational database as middleware. The database stores an abstract model of the controlled environment and information about the current state of the execution. A control application language, **`CAMOS(L)`**, used for describing high level operations, is tightly integrated with the database query language. In this way, **`CAMOS(L)`** provides expressions over combinations of conventional data, process data, execution states, sensor values, information from control algorithms etc. The database presents a highly heterogeneous environment with many data sources as a homogeneous programming environment to the control application programmer.

A set of control algorithms running on a separate real-time kernel performs the actual closed loop control of machinery and equipment in the external environment. Active rules in the database are used to interface the model of the environment, as stored in the database, and the control algorithms. In this way our architecture combines cyclic control algorithms and an event driven operation language with data



1. Metal processing plant

management capabilities. The database provides the architecture with general solutions to data management, complexity and interoperability problems. The database query optimizer improves the performance of the system. A motivating scenario [Lew94] with a production cell consisting of robots, machinery, conveyor belts, and sensory equipment etc. will be used in the paper to illustrate problems, concepts, and the new architecture.

The production cell in Fig. 1 contains a press that processes metal parts. The parts are placed in the press by a two armed robot. Two conveyor belts and a crane transport the parts to and from the production cell. Processed parts are placed on a pallet. The production cell has several sensors providing information about the environment and the machines. Most sensors return a boolean valued result, like "`press is closed`", but some sensors return real values, like "`the rotation angle of the robot base`". A number of actuators control the cell. Typical low level operations are "`start an electric motor`" or "`energize an electromagnet`". Some of these operations are time critical, like the running and positioning of a conveyor belt. Other operations describe the desired behaviour on a higher level, like "`first move the rotary table to pick-up position, and then, when the previous part is pressed, fetch the next part from the rotary table`". These high level operation sequences gives rise to rather complex temporal dependencies between the operations, the physical processes in the environment and the control algorithms.

In addition, the controlling software must restrict the movements of the machines to safe areas and avoid collisions.

We argue that the database centered control system architecture can be used to address problems represented by the production cell scenario. This type of advanced integrated control applications puts new demands on database technology such as: high performance, predictability, active rules, and a foreign function interface. In this paper, we show how the database can be used to store an abstract model of the external environment. We will also present examples of how the control application language based on the query language can be used as a high level operation programming language. We discuss how the active rule facility in AMOS [Skö94][Ris93] is used to monitor the state of the environment as stored in the database and how the appropriate control algorithms are initiated in a separate real-time kernel.

BACKGROUND AND RELATED WORK

Control applications require capabilities to manage long-running activities [Gra92]. In the literature there are several suggestions for new execution models to allow control of long-duration activities, which in themselves may consist of an arbitrary number of transactions (nested, chained, conditional etc.), e.g. SAGAs [Gar87][Gar90], ConTracts [Reu90], and solutions with relaxed transaction models [Chr93] [Geo95].

Workflow management systems (WFMS) [Rus94][Geo95] are used for specification of data and control flow between different activities (transactions). A workflow is typically a collection of operations organized to accomplish some typical business process such as the processing of loan applications or purchase orders. WFMS define the order of task invocation and conditions under which tasks must be invoked, i.e. task synchronization and data flow . A more flexible control structure is achieved using ECA rules to organize long-running activities [Day90]. Our approach uses a specialized control application language that hides the details about rules, rule activations and deactivations. The high level operations in the control application language can be viewed as an efficient workflow specification of activities in control applications. The difference compared to WFMS is that we provide means for dynamic scheduling and synchronization with external environment.

Many papers on hierarchical control system architectures refer to a "global database" very briefly . None of the architectures reviewed by Nat. Inst. of Standards and T echnology [Kra93] use an embedded query language or active rules in the execution model. The original ideas for a database centered control system architecture was presented in 'Active Object Oriented Databases in Control Applications' in the proceedings of the 19th Euromicro conference [Lob93]. The approach has its roots in two different research platforms developed at the Linköping University: the Active Mediator Object System (AMOS) and A Robot and Manufacturing Instruction System (ARAMIS).

A ROBOT AND MANUFACTURING INSTRUCTION SYSTEM

The ARAMIS layered hierarchical control system architecture [Hol92][Lob91] [Lob94] is based on a graphical language, which is a hybrid between a production rule view and traditional imperative programming languages. The goal was to introduce abstraction levels and simplify programming of the different parts of the software - like high-level operation descriptions, control software, and device drivers. The philosophy involves a world model (WM), which is a state model of the objects in the environment. This world model allows programmers to describe high level interactions by using well-defined and simplified world model objects. The highest layer in the architecture coordinates activities in the external environment using the world model as a blackboard [Hay85] to communicate with the control level layer . The control layer is closer to the environment process and acts as a servo mechanism using the world model as a reference value. Finally , a physical layer implements the I/O-interface to actual sensors and actuators. The ARAMIS system uses a data repository to represent the world model. In the work presented here, the simple data repository has been replaced with a main-memory based object-relational database, AMOS, whose query language and active rule facilities are used extensively by the control system.

THE ACTIVE MEDIATOR OBJECT SYSTEM (AMOS)

AMOS [Fah93] is a research platform for experimenting with specialized database systems. It is a fast main-memory object-relational database with a data dictionary , a query language, transactions, database procedures, and active rules[Skö94]. The query and modeling language of AMOS, AMOSQL, is a derivative of OSQL [Fis89] and is based on the DAPLEX functional data model [Shi81]. The system supports high level object-oriented abstractions and declarative queries for extracting and manipulating data. A foreign function interface allows external programs and drivers to be linked to AMOS. A cost based optimizer optimizes the queries [Lit92].

The AMOSQL modeling language has active rules facilities that detect updates to the database or to a data source [Fah93] [Ris93].

Active rules in AMOS has the format:

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

A rule condition can be any AMOSQL query and specifies when the rule should be triggered. The action part can be a procedure call to a foreign function or any database operation. The rules use set-oriented deferred condition-action semantics [Wid90]. Rules are used in control applications, to monitor conditions over combinations of sensor values and conventional data, which cause mode changes or initation of activities in the application.

DATABASE CENTERED CONTROL SYSTEM ARCHITECTURE

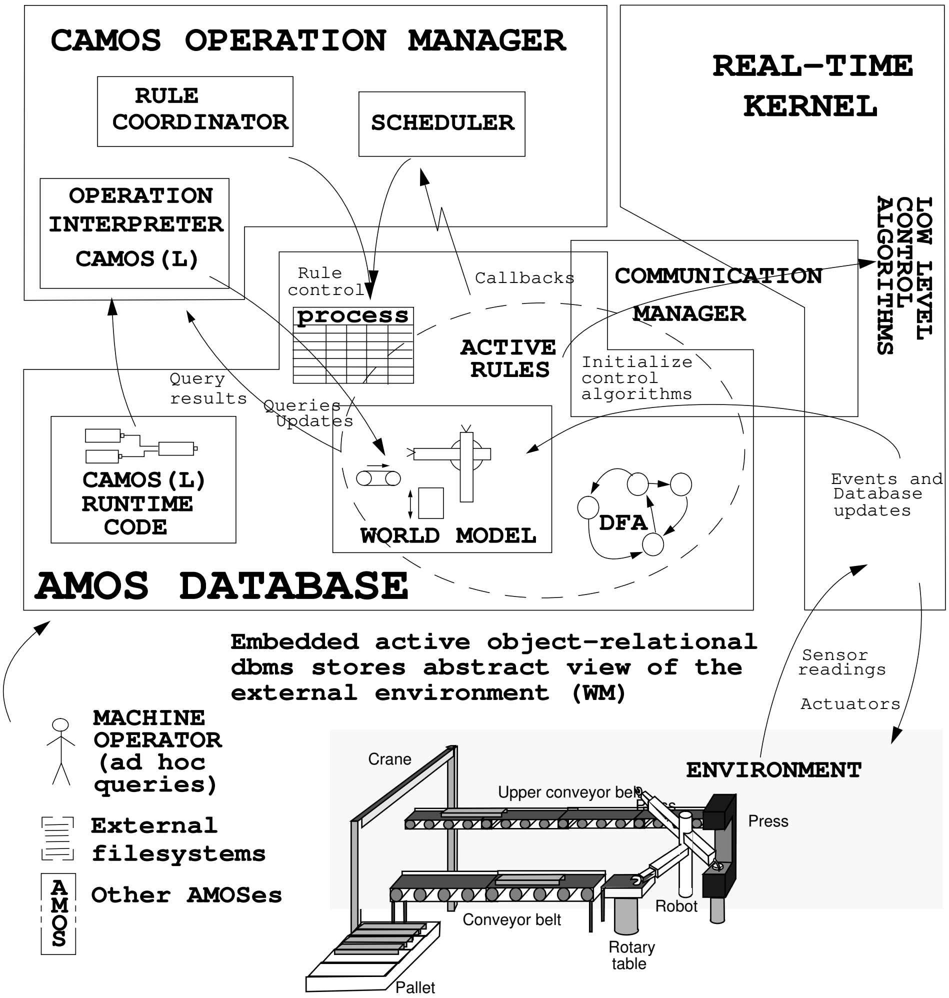
An increasing number of control applications require database-like functionality. The management of data becomes a problem when the control applications grow, become complex, and operate in environments with large volumes of data [Gra92]. Even in the simple scenario in the introduction, there are many types of information. There is information about machines, tools and the actual manufacturing process (i.e. process data). Furthermore, there is information about resources (such as inventory data), capacity, organization, production planning, status, produced amounts, and economical information. Matters are further complicated by the fact that there are many types of users with different needs: Machine operators, management, and the control application itself need to access and manipulate information. The environment in which the control application operates can itself be very complex and heterogeneous with many diverse subsystems that need to exchange information. Typically, the environment may contain sensors, actuators, file systems, communication subsystems, and conventional databases containing e.g. inventory data.

CAMOS (Control Application Mediator Object System) aims at an efficient and general approach to modeling and data management in control applications where the data modeling language of AMOS is used to model and organize information about the environment.

Basically, a control application system and a database system can be combined into a single unified architecture in the following two ways:

- By embedding the database queries and the active rules into the control application. The database will be a tightly integrated system function from the control application point of view. All database facilities are made available in the extended control application language. Query results can be used directly without sending messages or translation of the data format if a unified type system is used and the control application and the query processor can operate in the same address space.
- By coupling database and control application components as parallel, communicating subsystems. In this approach the database is isolated from the control application but the systems can communicate and influence each other. Unfortunately, typical high level operations executing in the control application interact heavily with the world model stored in the database. How efficient this type of integration will be depends on the capacity of the communication channel and on the amount of information that has to be exchanged.

In the **CAMOS** project, it was decided to embed the object-relational database AMOS into the control system architecture (alt. 1). We have built a control application language around the query processor and the active rule facility in AMOS. The database operates in the same address space as the control application. In particular, we aimed to overcome the impedance mismatch problem between traditional programming languages and database management systems by providing a unified type system and tight integration of the query processor. The timeliness of the high level operations involving large volumes of data is normally not a critical issue. All time-critical operations execute in a separate real-time kernel.



2. CAMOS Database centered architecture for control

The controlling software consists of the **CAMOS** operation manager , the embedded AMOS database, and a real-time kernel [Hol92]. The AMOS database is used to store an abstract object model of the external environment, called the world model (WM). In the production cell example it is a simplified representation of the state of materials and of the equipment. The world model gives the programmer opportunity to ignore those aspects of low level control algorithms that are not relevant for the definition of high level operations.

The active behavior of the world model objects is modeled by a set of deterministic finite state automata (DFA) stored in the database. Each node in the DFA corresponds to a state of an external object and each transition corresponds to the running of a control algorithm in the separate real-time kernel.

The control application language **CAMOS(L)** is used as a high level operation programming language and is compiled into runtime code, database queries, active rules, and database update transactions. The runtime code describes the sequences of possibly concurrent operations and interoperation dependencies. The queries are used to retrieve and derive information about the environment as stored in the database.

The execution of high level operations is synchronized with the changes in the state of the world model using the active rule facility of AMOS.

Database update statements are primitive operations in the language. Basically , the control application language coordinates database update transactions in the WM. If any of these updates trigger a transition rule in a world model object DFA, the communication manager sends a message to the real-time kernel that initiates a control algorithm. The control algorithm will influence the environment and the state changes in the environment will be sensed causing further updates to the database. This feedback works like a servo mechanism, keeping the environment consistent with the world model.

We have developed a general description model for operations and processes in control applications within the context of AMOS. The database stores operation descriptions and the state of the currently executing operations (processes). The operation descriptions and the processes are first-class objects in the database. This means that the process objects control the execution and are simultanously available for querying. Therefore, machine operators can state ad hoc queries about the current execution state as well as the world model state.

The runtime components are a scheduler , a rule coordinator , a communication manager , and an interpreter for **`CAMOS(L)`** runtime code. A compiler for the control application language translates expressions and conditions in the operation descriptions into database queries and active rules. The scheduler is a round robin scheduler that handles synchronization with external events and temporal dependencies between operations. The rule coordinator controls activation and deactivation of active rule instances according to executing operations. The communication manager handles an ordered queue of messages. Coordination with the real-time kernel is achieved through a command and status protocol which channels status messages when control algorithms are terminated or aborted. If an algorithm has caused a change of a state variable stored in the database, the communication manager will send an update statement to the database. The interpreter is used to prepare new suboperations processes according to **`CAMOS(L)`** runtime code in the database.

OVERVIEW OF THE CONTROL APPLICATION LANGUAGE

This section contains a brief review of the basic philosophy of the control application language, **`CAMOS(L)`**, being used as high-level operation programming language. **`CAMOS(L)`** is a successor to the programming language used in ARAMIS [Lob93] and addresses the need for data management in control applications by extending its rule based activity language with queries and transactions.

The basic concepts of the language are composite operations, iterative operations, primitive operations, temporal dependencies, and expressions. The composite operations and iterative operations are based on procedural abstraction, treating a group of possibly concurrently executing operations as a single unit. When initiated, iterative operations repeatedly perform sets of operations as long as the iteration condition is satisfied. Composite and iterative operations can be decomposed into preconditioned groups of suboperations that are dynamically scheduled. This provides alternative behavior of operations depending on which group preconditions are satisfied. The suboperations can be partially ordered by temporal dependencies (`dependency` statements), stating that some operations must be executed in sequence. Operations that do not have temporal dependencies will execute in parallel. The temporal dependencies can be said to introduce constraints on possible schedules for a composite operation at runtime. Note that the dependencies are conditional. The scheduling depends on the conditions for the suboperations.

```
CREATE OPERATION getPressedFetchNew(robot r, e_r_table t, press p)
AS WHEN   partsat(p) AND position(p)="open" AND
          palletCapacity()>=COUNT((SELECT b FOR EACH part b
                                    WHERE
                                 position(b)=currentPallet()))
      IF NOTANY(partsat(upperConveyor())) THEN
         1: getpressedpart(r,p)
      IF SOME((SELECT b FOR EACH part b
               WHERE name(position(b))="e_r_table")) THEN
         2a:getnewpart(r,t);
         2b:SET referenceElevation(t)=0;
         DEPENDENCIES:(2a,2b);
      IF SOME(partsat(t)) AND SOME(partsat(p)) THEN
         DEPENDENCIES:(1,2a);
```

3. An example of an operation with query constructs, temporal depencencies, suboperations and a wait condition.

```
CREATE TYPE actuator (name CHARSTRING);

CREATE TYPE rotator (actualRotation REAL, referenceRotation REAL)
   SUBTYPE OF actuator;

CREATE TYPE elevator(actualElevation REAL, referenceElevation REAL)
   SUBTYPE OF actuator;

CREATE TYPE gripper(actualGripperPosition BOOLEAN ,
   referenceGripperPosition)SUBTYPE OF actuator;

CREATE TYPE arm (actualExtension REAL, referenceExtension REAL)
   SUBTYPE OF actuator;

CREATE TYPE robot (gripper gripper,  /* part-of relations */
                   upperarm arm,
                   lowerarm arm,
                   robotBase rotator) SUBTYPE OF actuator;

CREATE gripper INSTANCES :gripper1,:gripper2;
CREATE arm(gripper) INSTANCES:upperarm(:gripper1),
                             :lowerarm(:gripper2);
CREATE rotator INSTANCES :robotBase;
CREATE robot(robotBase,upperArm,lowerArm)
   INSTANCES :robot1(:robotBase,:upperArm,:lowerArm);
ADD TYPE elevator to :robot1;
```

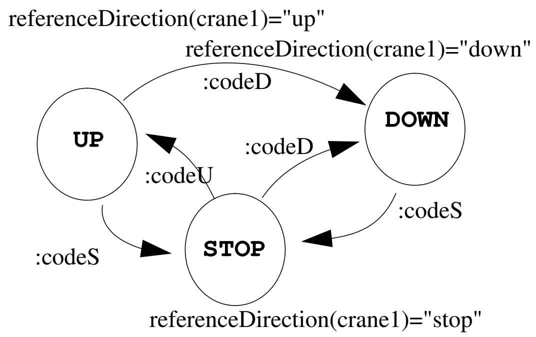
4. Object-oriented modeling of machinery

The execution is synchronized with the external environment using wait conditions. Wait conditions, iteration conditions, and expressions can be parameterized and may contain aggregate expressions. The primitive operations in the language are update transactions in the database, e.g. suboperation `2b` in Fig. 3. The purpose of the composite operation in Fig. 3 is to simultaneously fetch a new part from the elevating rotary table and to move the processed part from the press to the upper conveyor belt. Suboperation 2b is a primitive operation that will alter the world model. The `DEPENDENCIES` declaration introduce a dependency between suboperations 2a and 2b.

THE SYSTEM

The **CAMOS** system relies on several components in the AMOS architecture. This section will describe how the different database facilities (data repository , DDL, DML, transactions, queries, extensibility and active rules) are used to implement the unified control system architecture.

The main function of the database is to store information about the external environment, operation descriptions, and the current execution state. The information about the environment originates from the real-time kernel. The information can be sensed by running control algorithms, or be implicitly known (believed) at the



5. Crane modeled with active rules

time control algorithms terminate in the real-time kernel. The information is transferred to the world model (WM) in the database using a communication manager .

THE OBJECT ORIENTED WORLD MODEL

The WM is defined using the object-relational data definition language of AMOS. An object in the WM represents a meaningful entity in the external environment — like a robot or a metal part. The AMOSQL extensible type system is used to collect entities of similar properties into classes. Usually , the model of the environment includes objects with complex internal structures, which are composed of many subobjects, typically of another type (class) [Ors93]. The example in Fig. 4 illustrates how composite machinery can be modeled with part-of relations or using multiple inheritance. W e model a two-armed robot in the production cell scenario as an object with four subobjects attached. Using multiple inheritance, the object is extended with additional functions (actualElevation and referenceElevation) belonging to the elevator class.

INTERFACING TO THE REAL TIME KERNEL

A set of state variables are associated with the real-time kernel control algorithms for equipment and machinery . A subset of these variables are exported to the database, thus introducing abstraction and separation of shared data in the database and local data in the control algorithms. The " `actualRotation`" in Fig. 4 is an exported state variable. The interface to the real-time kernel involves checking if a transition belongs to a legal sequence of operations on an object, finding the appropriate control algorithm, transferring the data to the real-time kernel with different formats, and receiving results from terminating control algorithms.

The objects with active behavior in the environment are modeled as deterministic finite state automatas (DF A), which introduce restrictions on the usage of the object. Each node in a DF A contains a node constraint and the transitions are associated with control algorithms in the real-time kernel that will take the modeled object from one state to another . When a sequence of database updates satisfies a node constraint, the algorithm associated with the transition to this node will be scheduled for execution. If there is no transition to the node with a satisfied node constraint or there is no node with a satisfied constraint, then an error handler must be invoked. When the algorithms terminate, they report the new values of the exported state variables to the database. In this way , the DF A describes restrictions on legal transitions for active objects, and therefore restricts the possible sequences of algorithms that can be executed.

In the example in Fig. 5, the direction of the crane motor (`up, stop, down`) can be directly reversed if it is going up. If it is going down the motor must be stopped before the direction can be reversed.

A DFA is modeled in the database as a set of nodes, a current node, and a set of active rules activations that are responsible for the state transitions in the DF A. The conditions of the active rules consist of several parts. The condition must test if there exists a satisfied node constraint, if the object is already executing a control algorithm, and that the current node satisfies certain constraints. The action part of the active rule sends a message to the real-time kernel to initiate a previously downloaded algorithm.

```
CREATE RULE transition(object o, node n1, node n2, codeId c) AS
   WHEN  (currentNode(o)=n1 AND
          notany(algorithmRunning(o)) AND
          nodeConstraint(n2))
   DO     startTransition(o,c);

CREATE node(name) INSTANCES :uNode("up"),:sNode("stop"),
                            :dNode("down");
ACTIVATE transition(:crane1,:uNode,:dNode, :codeD);
```

6. Rules controlling the active behavior of a crane

In this way , the objects in the world model are monitored and controlled by the active rules.

The transition rules are parameterized. The same generic rule will be used for all transitions. In AMOSQL, the activation of a parameterized rule correspond to the creation of a rule instance with bound parameters [Ris93]. Fig. 6 shows how the DFA is represented in the database.

In summary , the invocation of control level algorithms is not directly expressed in the control application language. Instead, **CAMOS(L)** interacts indirectly with the environment using the primitive operations (update transactions). This separates the control view of the object and the operations the object is participating in.

QUERIES

The information about the world model and executing processes is retrieved using queries. Query constructs are used as wait conditions on the operations, as iteration conditions, as arguments to sub-operations, and as DF A node constraints.

The processing of queries is complicated by the presence of formal parameters in the operations. The example in Fig. 7 illustrates how a wait condition can be rewritten to a form that use a stored process representation (the instanciated operation) to make parameters available to the query processor . In Fig. 7 a translation of the wait condition in Fig. 3 is shown. It consists of two parts; a database query over the WM which yields a boolean result and an active rule controlling the operation execution.

THE RUNTIME ENVIRONMENT

In this section we present a logical view of how the execution of the control application language is carried out using the active rules. The operation manager repeatedly carries out the following sequence:

1. Data transfer from/to real-time kernel (communication manager)
2. Processing composite operations (interpreter)
3. Dynamic scheduling (scheduler)

```
CREATE FUNCTION cellReady(processId id) AS
   SELECT TRUE WHERE
         partsat(arg(id,3)) AND
         position(arg(id,3))="open" AND
         palletCapacity()>=COUNT((SELECT b FOR EACH part b
                         WHERE
                            name(position(b))=currentPallet()));

CREATE RULE cellReadyCondition(processId id) AS
   WHEN active(id) AND cellReady(id)
      DO BEGIN
            SET status(id)=:running;
            ADD currentProcesses()=id;
         END;
```

7. An example of a translated operation wait condition

4. Rule activation/deactivation (rule coordinator)
5. Rule check (database rule engine)

The interpreter creates a process object for every composite and primitive operation, which is then scheduled by the scheduler . A terminating operation implies deletion of the corresponding process.

Two basic synchronization mechanisms depend on the rule check phase:

- Rules that monitor the transitions in a DFA must be able to detect and react to changes in the world model. Therefore, each primitive operation must be followed by a subsequent rule check operation in the database.
- A rule check operation must also be performed after the control algorithms have terminated and reported the exported state variables. This will enable the operation wait conditions to trigger. Since the rule facility use deferred semantics, the execution model contains explicit rule check operations.

DISCUSSION

USABILITY OF ACTIVE DATABASE TECHNOLOGY FOR CONTROL APPLICATIONS

Applications with interacting rules are hard to develop using a flat active rule structure. Even a very small number of dependencies between rules will make it difficult to maintain the system [Wid94]. There is also the issue of interactions between the controlling software and the active rules.

Another problem is that control applications are highly dynamic. The applications continuously change focus resulting in very frequent activation and deactivation of rule sets. Considering the number of rules needed in a manufacturing control system, it would be difficult to use active rules alone to control it. Generally , the ECA and CA rules are too far from the application to be used as a programming language for control applications.

Therefore, we use a specialized control application language and a combination of active rules and a supervising operation manager to implement the event based part of the control system architecture. The actual condition-action rules are derived from the operation definitions and therefore hidden from the programmer . Hopefully this will result in easier maintenance, better understanding of the control flow and absence of undesired and unanticipated interference between the application programming language and a rule system in the database.

It is important to reduce the workload for the rule engine in the database. Three measures were taken to improve performance of the unified system:

The rules are only responsible for detecting and reacting to complex conditions in the database that otherwise would require extensive polling of the database. Several synchronization points in the execution model can be managed with wrappers, rendezvous techniques or messages instead of active rules. In the current implementation, only the wait conditions and the transitions in the DFA's need to be handled by rules.

The rules instances are active only when they are needed. Our approach benefits from the dynamic nature of control applications and the procedural description of operations. In **CAMOS**, active rules associated with wait conditions are activated when an operation is initiated and immediately deactivated when the rule triggers (fire-once rules). Only rules associated with currently waiting processes will be active. This reduces the workload for the rule engine.

All rule instances are reused. To provide efficient handling of activations and deactivations, rule conditions are augmented with boolean valued flag functions. Using the flag functions the rule coordinator can temporarily activate and deactivate rule instances without using the costly activate/deactivate statements in AMOSQL.

For this to be efficient the flag functions must be evaluated early in the query processing, which is achieved by cost hints defined on the flag functions. The approach will imply a small performance degradation during rule check phase while the flag function of inactive rules is checked.

The individual wait condition rule instances are derived from the same generic rule. The instances have individual parameter bindings and associated operation to which the wait condition belongs. The rule instances can be reused if an intermediary table is used to store parameters and operation identity . A predefined number of inactive rule instances can in this way be created at compile time, which will reduce the number of costly `activate` statements at runtime.

Systems involving active rules must also deal with the cascading rules problem. There are situations where one rule may introduce a change in the database that activates another rule, that may introduce yet another change etc. resulting in unpredictable response times. In the current implementation of CAMOS the action part of active rules are foreign function calls that will reschedule operations in the scheduler . The problem with cascading rules is avoided because no rules are generated containing database updates that directly could lead to triggering of another rule.

Three different categories of active rules are important in this control system.

- Aborting constraint. This type of rule can for instance be used to express safe areas in a declarative manner: `WHEN angle(robotbase(r)) >120 DO ABORT;`
- Synchronization rules: `WHEN actualElevation(t) =1 do synchronize();` This type of rules initiate activities outside the database. In our architecture synchronization rules are used to synchronize the operations with the real-time kernel.
- Rules in a specific application context: `WHEN active(id) AND condition DO.....;`

Typically only a small part of the wait condition rules are active simultaneously . In CAMOS this is related to operation invocations - when an operation is invoked, the scheduler activates a rule instance to monitor the wait condition and the rule is deactivated directly after is is triggered. Built-in system support for rule contexts would make it possible to check only the rules in currently active rule contexts. This mechanism is further discussed in [Skö95].

REQUIREMENTS

The work presented in this paper requires functionality not found in regular databases. The database must support:

- Efficient change monitoring techniques. The main motivation for using active rules in the execution model is that rules can handle operation dependencies involving large volumes of data efficiently. The incremental evaluation technique [Skö94] used by AMOS rules is more efficient than repeated polling or simple indexing of rule conditions.
- Fast rule context switch. The rules in a control system are not static. Since activation and deactivation is a very frequent operation in the control system architecture it must be efficiently implemented.
- Facilities to tightly integrate the database with applications. The result from queries should be directly accessible without translation of data formats and message passing. The **`CAMOS(L)`** language operate in the same address space as the database and share the same data types.

- High performance. The database must be very fast to handle the stream of updates from the real-time kernel. The AMOS database used in this project is a main memory based system that does not need slow and unpredictable disk accesses.
- Foreign function interface. In many situations the database need to access data and programs outside the database (communication manager, operation manager)

It is also important to be able to extend the database system with specialized data structures to model more closely the domain of control applications (processes, operation descriptions etc.) The application specific storage structures should be implemented at the lowest possible level to ensure best performance. The performance is essential if the database is used to store information about processes. Our implementation uses specialized ordered collections to represent schedules for executing processes.

The database must support extensions to query analysis, optimization, execution strategies, data access methods, and storage methods. AMOS is extensible with new data types using a foreign function interface. To provide the additional functionality and data structures efficiently, the new dedicated access methods have to be incorporated into the query optimizer. This is achieved using the open cost model in AMOS[Lit92].

SUMMARY

An increasing number of control applications require database-like functionality. It is the management of data that becomes a problem when the control applications grow, become complex, and operate in environments which involve large volumes of data [Gra92]. The integration of control applications with embedded active object-relational databases and a high level control application language aims at a general approach to modeling, data management, and exchange of information in complex control applications. This paper presented an event driven control application language **`CAMOS(L)`**. To fully benefit from the data management facilities in the AMOS database, queries were embedded in the control application language. This required a tight integration with the query processor and a unified type system to avoid translations of query results. When the control application architecture was extended to include a database with a query language, we also extended the power of expression for control application language. The database provides uniform access to all data. Therefore, conditions and expressions could be made over process data, information about the current execution, timing information, sensors, actuators and even external databases.

In the architecture, we use an object-relational database as middleware, storing an abstract model of the environment. Apart from storing the model, the database plays a central part in the execution model of **`CAMOS(L)`**. Parts of the operation descriptions are compiled to active rules in the database. The active rules efficiently monitor changes in the world model and initiate appropriate control algorithms in the real-time kernel. The integration requires functionality not found in regular databases. The database must contain an efficient active rule facility, support for process synchronization and an efficient context switch mechanism. The database must have high performance to handle the stream of updates from the real-time kernel.

The database centered approach is suitable for control applications that can be layered into an operation level and a traditional feedback level of control. The architecture brings us closer to better data management in control applications.

It is planned that this paper will be followed by an effort to formally define the architecture combining a general activity modeling language and cyclic control algorithms.

The hierarchical system outlined in this paper will be used as a starting point for that effort.

We also develop new tools for specifying and verifying important characteristics of the modeled system using modified timed petri nets [Pen93]. The long term goal is a design environment for modeling of large real-time systems which allows the designer to explore different alternatives and to verify timing properties of the design. We are conducting an extensive case study of a typical industrial production cell to test this approach.

ACKNOWLEDGEMENT

The authors would like to thank NUTEK, The Swedish National Board for Industrial and Technical Development for financial support. The local Linköping University funding from CENIIT - The Center for Industrial Information Technology is also gratefully acknowledged.

REFERENCES

[Bur90] Burns, A., Welling, A: *Real-Time Systems and Their Programming Language*, Addison-Wesley Publishing Company, 1990.

[Chr93] Chrysanthis, P.K., Ramamritham, K., A Formalism for Extended Transaction Models, *Proc. of the 17th VLDB Conf.* 1991

[Day90] Dayal, U., Hsu, M., Ladin, R.: Organizing Long-Running Activities with Triggers and Transactions, *Proc. SIGMOD* , Atlantic City, May 1990.

[Fah93] Fahl, G., Risch, T., Sköld, M.: AMOS - An Architecture for Active Mediators, *NGITS'93*, Haifa, Israel, 1993, pp. 47-53

[Fis89] Fishman D., et al: Overview of the Iris DBMS, *Object-Oriented Concepts, Databases, and Appl.*, ACM press, Addison-Wesley Publ. Comp., 1989.

[Gar90] Garcia-Molina, H. et al: *Coordinating Multi-Transaction Activities*, Technical Report No. CS-TR-247-90, Princeton University, 1990.

[Gar87] Garcia-Molina,H., K.Salem: SAGAS, *Proc. SIGMOD*, May 27-29, 1987, San Fransisco, pp. 249-259.

[Geo95] Georgakopoulos, D.,Hornick,M., Sheth, A.: An Overview of workflow management - From *Process Modeling to Workflow Automation Infrastructure* *Distributed and Parallel Databases*, 3, 1995.

[Gra92] Graham, M.H.: Issues in Real-Time Data Management, J. *Real-Time Systems*, 4, 185-202, 1992.

[Hay85] Hayes-Roth, B.: A blackboard architecture for control, *Artificial Intelligence*, 26, 251-321, 1985.

[Hol92] P.Holmbom,P., Loborg, P., Sköld, M., Törne A.: A Model for the Execution of Task Level Specifications for Intelligent and Flexible Manufacturing Systems, *Proc. of the Intl. Symposium on Artificial Intelligence, ISAI92*, Cancun, Dec 1992.

[Kra93] Kramer, R.,K.,Senehi,M.,K.: *Feasibility study: Reference architecture for machine control system integration* Nat. Inst. of Standards and technology, 1993.

[Lew94] Lewerentz, C., Lindner, T.: *Case Study "Production Cell" - A Comparative Study in Formal Software Development, FZI-publication* 1/94, Forschungszentrum Informatik Haid-und-Neu-Strasse 10-14 Karlsruhe, Germany, 1994.

[Lit92] Litwin, W., Risch, T.: Main Memory Oriented Optimization of OO Queries using Typed Datalog with Foreign Predicates, *IEEE Transactions on Knowledge and Data Engineering*, 4, Dec. 1992

[Lob91] Loborg,P., Törne A.: A Hybrid Language for the Control of Multimachine Environments, *Proc. EIA/AIE-91*, Hawaii, June 1991.

[Lob93] Loborg, P., Risch, T., Sköld, M., Törne, A.: Active OO Databases in Control Applications, in Microprocessing and Microprogramming, Vol. 38, No 1-5, p.255-264. *Proc. 19th Euromicro Conf.*, Barcelona, Sept 1993.

[Lob94] Loborg, P.: *Error Recovey Support in Manufacturing Systems*, Linköping Studies in Science and Technology, Lic Thesis no 440, ISBN 91-7871-376-5, 1994.

[Ors93] Orsborn, K.: *Modeling of Product Data Using an Extensible O-O Query Language*, IDA Technical Report LiTH-IDA-R-93-15, 1993.

[Pen93] Peng, Z., Törne, A.: A Petri Net Based Modelling and Synthesis Technique for Real-Time Systems, *5th Euromicro Workshop on Real-Time Systems*, Oulu, Finland, 1993.

[Reu90] Reuter,A., Wächter,H.: Grundkonzepte und Realisierungsstrategien des Contract-Modells Informatik, *Forschung und Entwicklung* No. 4, 1990.

[Ris93] Risch,T., Sköld, M.: Active Rules based on Object-Oriented Queries, *IEEE Data Engineering* (Quarterly), January 1993.

[Rus94] M.Rusinkiewicz and A.Sheth: Specification and execution of transactional workflows. In W.Kim (ed.): *Modern Database Systems*. Addison-Wesley, 1994.

[Shi81] Shipman, D.W.: The Functional Datamodel and the Data Language DAPLEX, *ACM TODS*, 6(1), March 1981

[Skö94] Sköld, M.: *Active rules based on Object-Relational Queries - Efficient Change Monitoring Techniques,* Licenciate Thesis No. 452, Dept. of Computer Science and Information Science, Linköping University, 1994

[Skö95] Sköld, M., Falkenroth, E., Risch, T.: *Rule Contexts in Active Databases - A Mechanism for Dynamic Rule Grouping* , *Proc. RIDS'95,* Athens, Greece,1995.

[Wid90] Widom, J.: Set-oriented production rules in relational database systems, *Proc. ACM SIGMOD conf.*, Atlantic City, New Jersey, 1990, pp. 259-270

[Wid94] Widom, J: Research Issues in Active Database Systems, *ACM SIGMOD Record,* Vol.23, No.3, September 1994.