Cooperative Systems Supported by Active Database Technology

Mike van de Graaf *Martin Kersten*

CWI
Kruislaan 413
1098 SJ Amsterdam
The Netherlands
e mail:{mgraaf,mk}@cwi.nl

Abstract

Rapid changing business processes need an IT infrastructure that supports the cooperation of applications that are located in separated business units. This paper explores the opportunity of combining process formalisation and active database technology to support this IT infrastructural need. Both parts have to be incorporated in today's IT infrastructure to support tomorrow's business demands.

1 Introduction

During the past decades, the importance of Information Technology (IT) in supporting the operations of organizations has increased. Meanwhile, organizations face more turbulent markets forcing them to rethink their strategies and processes frequently to survive [1, 2]. Three emerging trends to cope with market turbulence all have a significant impact on the IT supporting the day to day operations. The first trend is decentralisation; to get a better hold on market demand. Another option is to reengineer processes to significantly improve quality and performance. A third strategy is to search for new opportunities by 'reinventing the industry'. Decentralisation and 'reinventing industry' are somewhat complementary; smaller businesses can form multiple coalitions, which can change the shape of industry fast and unpredictable.

Interoperability of ISs is defined in [3] as the construction of applications where the implementation is constrained either by organizational boundaries or by legacy systems. They propose to support interoperability by modifying databases, operating systems, and communication protocols to permit new interoperability between applications to be more easily built. In this paper ways are explored to support interoperability of systems that are constrained by organizational boundaries.

1.1 Future Requirements

IT infrastructural demands differ for the type of business processes these have to support. Business processes are divided into two types [4]; **ill-structured** or ad hoc business processes are human oriented. The focus of the under-

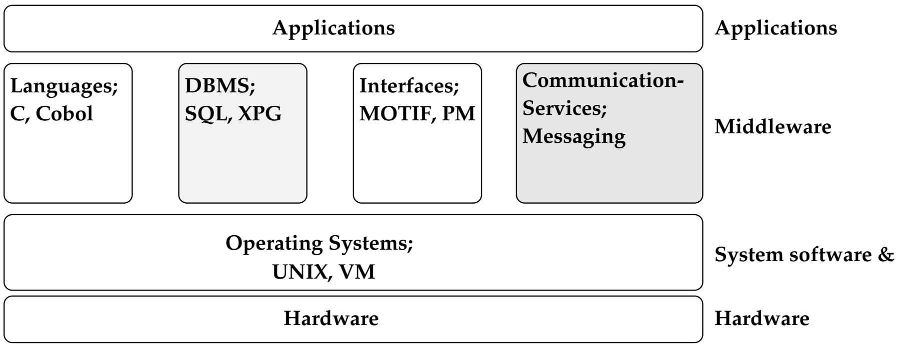

Figure 1: *ISO architecture.*

lying technology is to provide a means to communicate. There's little *a priori* known about the state and sequence of the work progress. Possible solutions to support ill-structured processes are studied in the field of *Computer Supported Cooperative Work* (CSCW).

The second business process type is called **structured**. It is supported by transaction processing systems that schedule repetitive tasks that are aligned according a predictable sequence. Frequent changes in the business process show a weakness in many of today's transaction oriented systems.The first challenge therefore is to specify the dynamics of a process. This would define **cooperation** among applications that execute the process' tasks. The dynamic model highlights the behaviour of the process and this can be changed without the modification of local applications. Through that, ISs can cope with the rapid changing demands while at the same time reaping of the benefits of repeatability and predictability.

Tomorrow's transaction oriented processes will be supported by a wide variety of applications, presumably allocated over many different sites. This introduces a heterogeneity problem in two ways; the semantics of distributed data and data access. This paper focuses on the latter, as it will introduce one interaction protocol that supports **heterogeneous query languages**.

1.2 Possible Solutions

Many papers and proposed standards have appeared that described system infrastructures. Among them, the ISO architecture divides systems into three layers; hardware and system software, middleware, and applications. Middleware enables applications to be constructed independent from the bottom layer. Figure 1 shows these three layers and some relevant Application Program Interfaces (APIs) in the middleware.

One of the possible solutions to incorporated flexibility in today's Information Systems (ISs) is emerging in the field of *End User Computing* [5]. IS development is 'down sized' as an answer to its backlog, enabling users to define their own data formats and interactions, via mechanisms such as *Dynamic Data Exchange* and *Object Linking and Embedding*. These mechanisms are supported in the latest office environments, such as Lotus Notes and Microsoft Office.

Unfortunately, these approaches are not well suited for transaction oriented processes, since they are merely focussed on **communication services**. They are often referred to as 'CSCW applications'. Transaction oriented systems are at the present incorporated in large **DBMS**s, based on some Client/Server mode of interaction. The clients link their execution to the DBMS server by embedding SQL type languages in their interfaces. An example of such an approach is outlined in [6], showing embedded SQL-statements for numerous languages. This way every application defines its own interaction to the database, which is costly to maintain and not easy adjusted to new business demands.

In search for a sound approach in supporting transaction oriented systems for both flexible cooperation and persistent data application interaction, we propose a solution that enhances the DBMS function of middleware in two ways: 1) by adding dynamic process descriptions which are necessary to specify process behaviour, and 2) providing implementation support for these specifications by applying activity management. These enhancements emerge from two areas that are elaborated upon in the remaining of this paper.

Workflow: In recent years, the development of workflow management and tools is receiving much attention. The management part is viewed from a business perspective as a vehicle to migrate to process focussed organizations. The tool part focuses on technological issues, and can be divided into the three main areas [4] of construction and analysis of business processes, data management, and interfaces.

Many methodologies and models appeared to support the analysis of processes. An sound approach is the use of high level Petri Nets [7], since they are powerful in specifying business transactions at a highly formalised level. As will be shown, this is a good basis for an implementation strategy.

Active Databases: To provide a stronger link to 'database actions', while at the same time providing for a persistent 'interface' with applications, active DataBase ManagementSystems (aDBMSs) are proposed. An aDBMS incorporates active behaviour by triggers and alerters. Although the need to capture active behaviour in a DBMS emerged merely from an internal focus on data management, such as dynamic view maintenance and constraint checking, an early study of the HiPAC project showed that aDBMSs could be used to schedule applications [8]. More recently, this approach is elaborated upon in [9], showing the following results:

1. The applications can communicate indirectly via rule firing. If the data changes, a rule is enabled which invokes the correct application and passes its data.
2. Operational decisions in a business process can be encoded in the control logic of aDBMS rules.
3. The behaviour of a business process can be changed by modifying the rules rather then the software.

It is our hypothesis that a synergy emerges if both approaches are combined and added to the middleware.

First, the coordination between different applications is formalised, thereby incorporating the cooperation function. Second, the aDBMS rules are derived from the formalisation and invoke applications with the correct data. Thus, heterogeneous data access of applications is defined by only one interaction protocol that is controlled in the activity manager of the aDBMS.

1.3 Paper Outline

We take a 'top down' approach in this paper, by capturing the fundamentals of a structured business process first conceptually in Section 2. It will be shown that both coordination and the interaction protocol can be modelled by a high level Petri Net. From this conceptual model, the database rules are derived in Section 3. It shows a minimum architecture that provides the middleware with active components. Section 4 shows our approach in a small sample process, translating process descriptions to active database implementations. Concluding remarks are presented in Section 5.

2 Dynamic Process Model

Structured business processes can be described in many different ways, ranging from natural language to complex modelling techniques. A representation of a business process for this purpose must highlight the dynamic behaviour of the process. It must show the interdependencies between activities and 'when' they need to operate on 'what'.

Petri Nets are used both in Workflow and aDBMSs to analyse dynamic behaviour. Although the approaches in Workflow use high level Petri Nets (HLPNs), SAMOS [10] showed that simple Petri Nets could be used to define the sequence of rule firing. In this section an approach is introduced to describe a business process with a HLPN. It provides a means of translation to aDBMS rules, discussed in the next section.

2.1 Petri Nets in Workflow

A structured business process consists of recognizable states during operation. These states can be modelled using a Petri Net, as first described by Carl Adam Petri in 1962. Such a Petri Net contains: places, transitions, and tokens. Input places are connected with transitions via input arcs, and the transition's output arcs are connected with output places. When all input places of a transition contain a token, it *fires*, consuming the tokens from the input places and generating new tokens in the output places.

Many extensions have been defined to enhance the Petri Net for the modelling of complex systems, such as business processes, termed HLPNs. It is used here to specify the dynamics of a structured process to derive formal event-action implementation at a subsequent stage. The main components of a business process are mapped on those of a HLPN, as listed below and based on [7].

Coordination of activities in a business process are translated to *transitions* in a HLPN. They carry a logical formula, called an *inscription*, enabling the firing of the transition on the availability and *state* of the input. It is elaborated upon in Section 2.2, showing five interdependencies of activities modelled out in transitions.

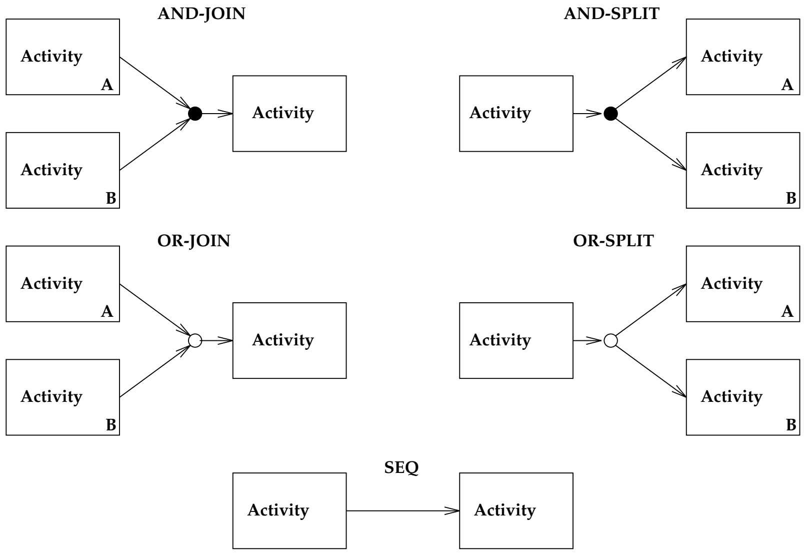

Figure 2: *Five coordination mechanisms.*

Applications specify the activities carried out in a business processes. They are modelled as *places*, which can transform a token. The transformation is guided by aDBMS rules in a subsequent stage and introduces a need to define one interaction protocol to schedule applications. This is formalised in Section 2.3.

Products are created and transformed by the activities and are defined as high level *tokens*. These tokens can be any object 'flowing' through the process, but it is limited to information objects here. A common view on information objects is that of a traditional form, containing attributes or fields. Section 2.4 describes the modelling of products.

2.2 Coordination

The need to coordinate activities to achieve a common goal was recognized a long time ago, but only recently approaches appeared to formalise it in the context of structured business processes. Focusing on the formalisation of coordination of activities, a set of standard coordination mechanisms can be derived as shown in Figure 2, obtained from the Workflow Management Coalition[1]. These coordination mechanisms can be applied in any transaction oriented process. Their implementation can be restricted to a set of rules at a subsequent stage. The coordination mechanisms are listed below in general terms of activities, but this applies to applications just the same.

- **And-join** interdependency occurs when two activities have to finish their operations before a subsequent activity can start. An example is order entry; only if complete product and customer data is available, an order can enter the system for further processing. **And-split** takes place when one activity is followed by a number of *parallel* activities. For example, an order can be used

[1] WWW address: http://www.aiai.ed.ac.uk-/WfMC/

Table 1: Input/Output of Coordination Transitions.

Type	Inscription	Action
AND-JOIN	$(A \wedge B)$	O:→post-condition
AND-SPLIT	pre-condition	O:→$(A \wedge B)$
OR-JOIN	$(A \vee B)$	O:→post-condition
OR-SPLIT	pre-condition.A	O:→A
	pre-condition.B	O:→B
SEQ	pre-condition	O:→post condition

simultaneously by manufacturing, finance, and logistics for different reasons. Concurrency control on parallel activities is handled by low level transaction management.

- **Or-join** interdependency occurs when two activities can precede one activity without any ordering. For example; every outgoing product is logged. **Or-split** is perhaps the most encountered coordination mechanism, where one activity is followed by one activity out of a set. This implies *rule based* scheduling. An example is a finance department receiving orders. Every order-amount exceeding a certain threshold is sent to a higher leveled administrator.

- **Seq** interdependency occurs when one activity is followed sequentially by the next, e.g. if new mail enters a department, an acknowledge is sent to the sender.

The coordination mechanisms of Figure 2 are formalised for HLPNs in Table 1, where they are shown with two input or output places. This can be extended to a more general formalisation of n places without loss of functionality.

2.3 Applications

When a product is assigned to an application, a persistent interaction needs to guarantee correct execution of its transformations. An application is signaled by sending an 'envelop' with the data, and the reply to that envelop is monitored for during the execution. The elementary parts of this interaction protocol are stated below, and are somewhat similar to those identified in [11].

1. **Start/Notify/Nothing:** When work is assigned to a specific application, one response is to startup the application and pass its data. This can impose constraints on an organization that are too strong, since some applications need human intervention to complete. Therefore, the start-function is loosened by the introduction of other options, including notify and nothing. The former notifies users that work is to be done, but leaves the final control over the (point in time of) transformation to the users. The latter option suggest that users monitor, open, and modify the work whenever they want to.

2. **Contingency:** If an application is not available, or a serious error occurred during the startup, alterna-

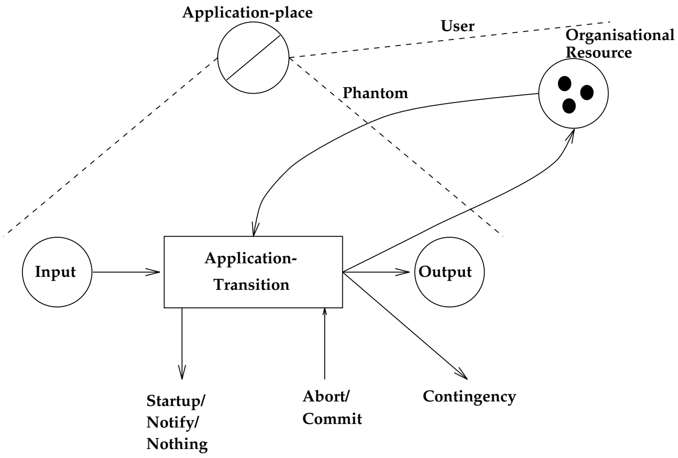

Figure 3: *Interaction between LUW-states and applications.*

tive actions must be executed. If not, work gets stalled at this state in the process.

3. **Abort/Commit:** If an application finishes its transformations, it has either completed successfully or not at all (atomicity principle). An abort must lead to a redo or contingency, whereas the commit signal sequences the next actions in the process.

As noticed, the type of the business activity[2] defines the supporting application type. These are divided into two classes here.

- **User applications** support ill-structured activities. This application needs human intervention to complete, and carries the identifiers of all users involved. This provides the link to the resources of the organization, which can be a very complicated structure of capacity and resource allocation. It is not mentioned here any further.

[2] In general: process activity task modelling is more complex. Here it is assumed that an activity is supported by one application.

- **Phantom applications** support highly structured activities without human intervention. These applications can be bounded to other resources, e.g. license numbers, clock, or computer resources.

The interaction protocol for both application types in a HLPN place is graphically represented as a $\oslash$ symbol in Figure 3. This symbol hides an input place, an application transition, and an output place. The input and output places serve as *baskets* for incoming and finished work. From its input, the application transition can execute startup actions. After transformation, the work is put in the output. Contingency and abort both lead to a redo or alternative action.

A User application needs external resources to startup its executions. This is modelled in the 'Resource place', containing all the resources that can be available at one moment. If work is assigned in the input, the application transition consumes an available resource and momentarily makes it unavailable.

A commit signal leads to the consumption of the LUW token from the input and a generation of it in the output, specifying that the LUW is transformed. Also, the resources are released for this job and they return to their place to be needed again latter.

2.4 Products

The products of a structured business process are actually transformed within the business process, and are called **Logical Units of Work** (LUWs). It can be seen in analogy with a traditional form, only it can be logically integrated/separated by database primitives, like views. The data can be maintained in many differing ways in databases. Traditionally, they were constructed from Entity Relation models and implemented in Relational DBMSs. Nowadays, other representations, like Object Orientation, are of gaining interest in the database community. The only requirement for this purpose is that their attributes are defined explicitly.

To support transaction processing of LUWs, the aDBMS needs to be enhanced to include meaningful business process properties, such as time and LUW-states. Time is added in HLPNs for the particular usage of logistics [7], and the **LUW-states** can be implemented by means of conventional DBMS techniques. The input and output of any application are such meaningful LUW-states, but the concept can be generalized to capture all sorts of LUW-states within a process.

If a HLPN is constructed with application places, coordination transitions, and LUWs (see Section 4), a business process can be analysed on its dynamic behaviour. Since the aDBMS rules are implemented out-of-context, rule analysis is a difficult task and therefore has to be done at definition stage in the HLPN. Some of the HLPN analysis techniques that are useful for aDBMS design and Workflow analysis are:

1. Simulation: the process can be executed to analyse its dynamic behaviour.
2. Termination: each desired end state of a business process must be reachable when tokens are assigned to the first application in the process. To guarantee termination, the inscriptions must cover all input domains and link them to at least one output[3].
3. Lockings and loopings: analysing the flow of tokens to detect (unwanted) livelocks or deadlocks. Similar, it is sometimes unwanted that a tokens can 'loop' in a process.

When a business process is fully specified and analysed, the link to IS implementation must be made. The aDBMS rules support this migration as being the implementation of coordination and application transitions.

3 Active Database Technology

Although solutions appeared to extend DBMSs to support active behaviour, most of the research on aDBMSs have opted for an integrated approach [12]. Such includes: rule definition, event monitoring, condition checking, and extended transaction management to support concurrent rule processing. It is the

[3]Else, LUWs that would have characteristics of uncovered variables would not be transported and would therefore be stalled.

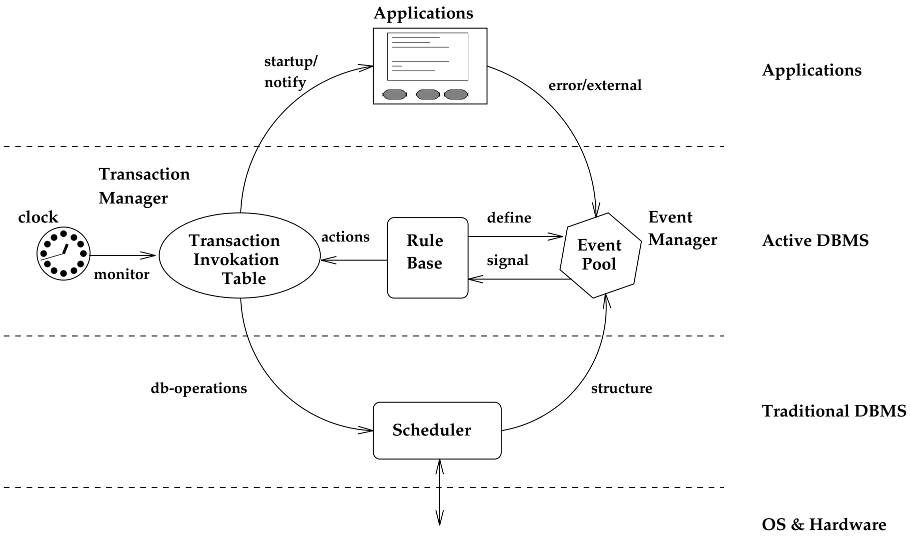

Figure 4: *Middleware enhanced with active database technology.*

aim to implement the formalised business process in an integrated aDBMS architecture. This implies that rules must be defined for both a coordination and application function, it must be possible to monitor business events, conditions must be checked, and extended transactions must be fired from a rule.

Figure 4 shows an integrated architecture for the purpose of activity management. This section describes these active components that are situated within the middleware.

3.1 Rule Base

A rule is defined in a Rule Definition Language, general defined as:

create rule <name>
when <event>
if <condition>
then <action>

It is commonly referred to as the triple Event-Condition-Action, or ECA. Modern aDBMSs differ in the way they explicitly separate these parts. Some integrate the condition and action part while others keep them separated. Both strategies have their advantages, as will be shown when the rules are defined. The ECA rules reside in the Rule Base, from which the event part is distributed to the event manager for detection purposes, and the action part is send to the transaction manager if the event occurred and the condition was evaluated.

ECA rules implement the coordination and application function. Thus, there are five coordination rules implementing the coordination mechanisms, and there is one application rule handling the interaction with each individual application.

Coordination Rules: They provide the 'data logistic' function by moving LUWs between the output and input state of applications. When LUWs are inserted in the output, a coordination rule is *triggered*. Action is taken to send this LUW to the next application's input, according to the inscription of the formalised coordination transition.

The inscriptions of the transitions are translated to the event and condition parts of aDBMS rules. If rules are defined with an explicit condition clause, the inscription can be *broken up* over a number of rules. Thus, a complex inscription with a nested if-then structure of n deep would be implemented by n ECA-rules. These rules fire on the same event and check a part of the condition.

In the sample process of Section 4 the condition and action parts are integrated. The same nested if-then structure is here implemented in the action part. The advantage of this for event management is that only one rule is fired, but as the action part becomes more complex, rule-management becomes difficult.

Application Rules: They are triggered when a LUW is assigned to its input and can react according the triple Start/Notify/Nothing. A contingency rule is triggered on an error code, invoking some contingency action. Finally, if an application is closed, the work is send to the output if the execution is handled correct, or else other actions are implemented.

A Phantom application is executed with the proper data passed as a parameter. The startup is handled by the transaction manager. The startup rule needs to be enhanced for User applications. Human resources must be glued to this representation of activities, and this can be done in numerous ways. In its most elementary form, an abstract notion of *agents* can be introduced that are in some pre defined state (free or executing), to be detected by a rule [11]. Organizational concepts of staffs, groups, and individuals have also been exposed to this approach [13]. Here, an example is given of individuals that are linked to parameters in the rule system.

Suppose there is a User application that can be executed by a set of users S. Work can be assigned to one of these users if they are available. When they are committed to this job, they cannot be assigned to other work, so they must be removed from the list of free users. A general User rule performing a startup function in case of an agent situation looks like:

```
Create rule Application Startup
on insert x Application-input
if free(S)
then {
    take(s,S)
    Call Application(s,x)
    }
```

User s is selected and gets a certain application assigned with the correct LUW to work upon. After the job is done, the human resource is 'released'. This rigid approach of human resource scheduling is merely a scientific. Solutions to link groups and users on an elegant way to systems is not yet presented.

Note that the coordination rules are internally focussed on the traditional DB-environment of LUW-states and their manipulations, whereas application rules have part of their actions and events situated in the external environment. This difference will return in the next two sections, where the remaining components of the architecture are discussed.

3.2 Event Manager

The Event Manager receives event parts of ECA rules for the purpose of event detection. Primary interest of event detection are the *event sources* and the *consumption policy*. The former identifies which part of the system is monitored, the latter describes the type of reaction when one event is detected.

Event Sources for this purpose are: structure, error, and external. This is a subset of a larger set of event sources identified in [14]. The **structure events** are DB operations, such as insertions and deletions. The coordination rules monitor each application's output on such events. The **error event** is monitored by the contingency rule. This error can also be part of an external system, e.g. an operating system. The **external events** are the signals from the applications, and are restricted here to abort and commit. These signals originate from other systems.

When the events are defined, they are placed in a pool [15]. Every detected event is searched for in the pool. If no rules are attached, the event is removed. If an event is composed of a set of primary events, the event manager has to perform event analysis[4]. Suppose two rules are defined **on A,¬B** and **on ¬A,B**; meaning that a rule fires if either A or B occurs. If a new rule is added to the pool firing **on A,B** it would never fire; A and B are directly removed from the event pool when they occurred, after firing a previously defined rule. HLPNs can be used in the analysis of these situations, like it is used in SAMOS [10]. The difference is, that this approach also checks the condition parts to guarantee that at least one rule's action part fires when an event occurred. This serves the analysis of process termination.

[4] Also called event algebra [15].

The **consumption policy** identifies the rule(s) to be signaled on an event occurrence. In general, this can be an **all fire** or a **one fire** policy. If the ECA rules are separated for each coordination rule, an all fire policy has to be executed, since all the rules check some part of the inscription. When the CA part is integrated only one rule has to be fired.

Consumption policy gets more complicated when there are more rules defined on applications[5]. In this situation, **priorities** can be assigned to rules. These priorities guide the consumption of events, and with that, the behaviour of the process.

3.3 Transaction Manager

Once events are detected and their conditions are evaluated, the action part is placed in the transaction manager. They are inserted in a transaction invokation table on their occurrence [16]. From this table, DB actions are routed to the scheduler of the aDBMS, and application actions invoke 'calls' outside the aDBMS. An extra function of the transaction manager is that is monitors the clock[6]. This enables time based rules to be installed in the table before they have to be executed.

An important issue in rule management that is covered by the transaction manager, is the **coupling mode**. A complete overview of this matter is presented in [9], showing that the coupling of the ECA parts can change the behaviour of the transaction's result. For this purpose, actions can be executed in an **immediate** or in **detached causally dependent** mode. The for-

[5] When they take part in multiple business processes, for example.

[6] Most architectures do not specify a particular place for time detection.

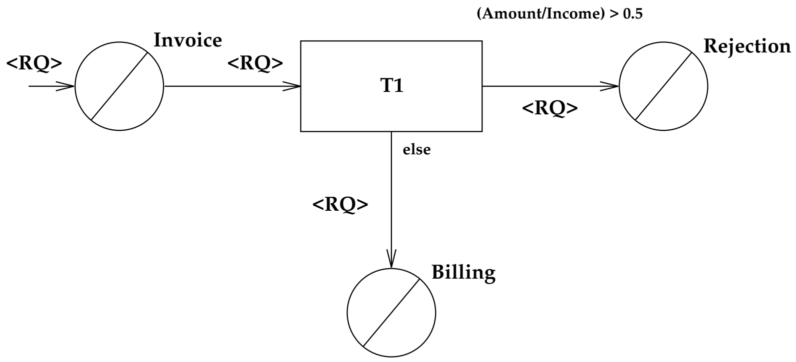

Figure 5: *The PrT Net of the sample process.*

mer means that the actions are executed before the transaction that started the action committed. The latter is executed in a separated transaction that can only commit if the transaction that started the action committed. Thus, in immediate mode, the actions can be **rolled back** when the originating actions abort. This situation occurs for coordination actions that have an internal focus. When calls to the outside world are made, roll back is not possible so an ultimate guarantee has to be given on its correctness. Therefore, the detached causally dependent mode is used for application rules. They can only startup/notify an application if the work *is* assigned. This is implemented through commit dependence.

4 An Example

"A loan department of a large bank must handle all loan-requests of its clients. These enter the department at the 'Invoice', where these are checked on completeness. A request is sent to 'Billing' if the income of the client is sufficient, i.e. the requested amount of a client may be up to 50% of the annual income. If it is more, the loan request is sent to 'Rejection'."

The three activities of this sample process are Invoice, Billing, and Rejection. These are given in a HLPN by the ⊘ symbol. Only one coordination mechanism is defined here, and it is an OR-SPLIT that guides the requests to either Billing or Rejection based on the rule that:"...the requested amount may be up to 50% of the annual income." The only product that is transformed here is the request. An abstraction of it is the token <RQ>. The HLPN of this situation is presented in Figure 5.

Analysis of the inscription shows that a token that is presented at the output of the invoice will always be transported. The variable *amount/income* is covered, since it is sent to rejection if it is more than 50% and else to billing.

The three applications that support this process are presumably of a different nature and may operate in separated locations. These applications operate on the request (=LUW) and must therefore be managed to enable easy communication. To simplify distribution of LUWs, only their identifier is sent to the inputs of applications. With it, applications can access, modify, and store the (distributed) LUWs[7]. From the process specification the triggers for coordination and application functions can be derived. These are presented below.

Coordination Trigger T1 is defined as or-split. The inscription sends the data to either Billing or Rejection, based on the *amount/income* variable. In this sample process, the condition and action parts are merged.

```
create rule T1
when {append LUW.id to I-out}
then
 /* select LUW data */
 select RQ.Amount, RQ.Income
 from RQ
 where RQ.id==LUW.id

 if ((RQ.Amount/RQ.Income) > 0.5) then
 append LUW.id to R-in /* reject */
 else append LUW.id to B-in /* billing */

 delete LUW.id from I-out
 }
```

The event manager is programmed with the when part as it will monitor the I-out LUW-state. Every append to this state leads to a 'raise event' and the rule will be fired with the correct identifier. The transactions that are assigned to the transaction invokation table are a select, an append, and a delete. Thus, these are all sent to the DB-scheduler.

Application Triggers are defined here for three Phantom applications for brevity. Therefore, these applications can start immediately when work is assigned to their input. The startup trigger is only shown here for the invoice application.

```
create rule invoice
when {append LUW.id to I-in}
then {call <invoice-application> LUW.id}
```

The contingency and abort triggers are merged here, and startup the same application. Note that a complete ruleset of errors and their responses could be defined this way. The commit trigger is executed on correct transformation of the application. It moves the identifier to the output. Both examples are given for the invoice application.

```
create rule error-invoice
when {error <invoice-application> LUW.id}
then {call <invoice-application> LUW.id}
```

```
create rule commit-invoice
when {commit-<invoice-application>
then                                   LUW.id}
 {
 append LUW.id to I-out
 delete LUW.id from I-in
 }
```

The structure of the applications is hidden from the triggers. The above trigger-statements show that exactly the same triggers can be used for the other applications by replacing it with the name of the invoice-application. The event manager is programmed to monitor an append on I-in, error, and commit. The appropriate rule is selected and the identifier passed. The actions that are generated by the transaction invokation table are calls, appends, and deletes. The calls are executed in a detached causally dependent mode.

[7]The issue of autonomy can be discussed here in more detail, as this description implies choices in the design, communication, and execution autonomy of systems. Future research must cover these issues explicitly.

5 Concluding Remarks

Dynamic models and aDBMSs support the specification and implementation of cooperative systems when they are added to the middleware. Several benefits of this approach can be identified. First, dynamic models specify a business process unambiguously and rapidly. It abstracts from local situations and defines cooperation of tasks (applications) at a high level. The results of the specification can be analysed, covering issues such as termination and rule behaviour. Second, rules of the aDBMS provide for a complete implementation strategy derived from the dynamic model. The components of an architecture supporting activity management were outlined. It was shown that aDBMSs can guide long transaction management, linking high level business activities to low level database transactions.

This study evolves in three dimensions. The fist dimension is to establish a tight link between the dynamic representation of a business process and aDBMS implementation. This opens a possible route to a new generation of CASE-tools. Further, this approach has an impact on the autonomy of systems, in particular design, communication, and execution autonomy. When one model is created for multiple systems, the local data-models and execution schemes must be integrated or joined in a federation. Finally, the link between ISs and organizations has to be defined in more detail. As was briefly shown, users and user groups are related to the execution of User applications. In general, organizational principles, like capacity planning and resource scheduling have to be examined on their implications for the business tasks that are supported, controlled, and executed by software.

References

[1] Hamel, G., C.K. Prahalad,'Competing for the Future', HBS Press, 1994.

[2] Hammer, M., J. Champy 'Reengeneering the corporation', HarperBusiness, New York, 1993.

[3] Colomb, R.M., M.E. Orlowska,'Interoperability in Information Systems', Technical Report No.263, Key Centre for Software Technology, Department of Computer Science, University of Queensland, 1993.

[4] Georgakopoulos, D., M. Hornick, A. Sheth,'An Overview of Workflow Management: From Process Modelling to Workflow Automation Infrastructure', Distributed and Parallel Databases, 3, Kluwer Academic Publishers, Boston, pg.119 153, 1995.

[5] Sipior, J.C., G.L. Sanders,'Definitional Destinctions and Implications for Managing End User Computing', Information & Management, No.13, p.51 67, 1993.

[6] Goveran, D.,'A Guide to Sybase and SQL Server', Addison Wesley Publishing Company, 1992.

[7] Hee, K.M. van, W.M.P. van Aalst, G.J. Houben,'Modelling and analysing workflow using a Petri Net approach', Petri Net conference, 1994.

[8] Dayal, U., D.R. McCarthy, 'The Architecture of an Active Database Management System', Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, Portland, Oregon, May 31 June 2, 1989, pg.215 224.

[9] Branding, H., A. Buchmann, T. Kudrass, J. Zimmermann,'Rules in an Open System: The REACH Rule System', Proceedings of the 1st International Workshop on "Rules in Database Systems", Department of Computing and Electrical Engineering, Heriot Watt University, Riccarton, Edinburgh, 30th August 1st September, 1993.

[10] Gatziu, S., K.R. Dittrich,'Events in an Active Object Oriented Database System', Proceedings of the 1st International Workshop on "Rules in Database Systems".

[11] Bussler, C., S. Jablonski,'Implementing Agent Coordination for Workflow Management Systems Using Active Database Systems', IEEE Conference on Data Engineering, Houston, Texas, p.53 59, 1994.

[12] Chakravarthy, S.,'Architectures and monitoring techniques for active databases: An evaluation', Data & Knowledge Engineering 16, Elsevier Science B.V., p.1 26, 1995.

[13] FlowMark, IBM Vienna Software Development Laboratory, Austria, 1994.

[14] Paton, N., O. Diaz, M.H. Williams, J. Campin, A. Dinn & A. Jaime,'Dimensions of Active Behaviour', Proceedings of the 1st International Workshop on "Rules in Database Systems".

[15] Kersten, M.L.,'Construction of an Active Component for a Database Kernel', International Workshop 'Rules in Databases', RID'95, Athene, September, 1995.

[16] Grefen, P.W.P.J.,'Concepts and techniques for an active database approach to office procedure support', University of Twente, 1993.