Information Systems Development Using a Combination of Process and Rule Based Approaches

John Krogstie* Peter McBrien†
Richard Owens† and Anne Helga Seltveit*

During the last years, the time aspect in information systems development has been addressed by several researchers [2], [8],[6]. Organisations are dynamic by nature and thus, the importance of modelling time explicitly in systems engineering approaches is crucial. This paper proposes a way of integrating process and rule based approaches in information systems development. Both static and dynamic aspects including the temporal dimension can be described. We envisage an approach with incremental specifications where details are successively added until we arrive at a specification from which executable code can be automatically generated. The output from this process (i.e., a set of rules) should be compatible with a rule manager which controls the execution of the system. A prototype has been developed to demonstrate the feasibility of this approach and is briefly described.

Keywords: Top-down approach, incremental and iterative development, process modelling, rule modelling, temporal dimension, rule manager, code generation.

1 Introduction

The evolutionary aspects of information systems development are badly covered by traditional approaches [10],[11]. Specifications used in the early phases of development are often limited to natural language descriptions and specifications developed by informal operational (procedural) diagramming techniques, and accordingly there is a gap between system specifications and actual implementation of the target system. This has motivated an approach where we emphasise providing a conceptual model which has the necessary expressive power and formality and at the same time contributes to the bridging of the gap between the analysis and execution levels.

We present a modelling formalism which can represent not only the first approximation of a system's specification, but also all the subsequent refinements of it into an imple-

*Division of Computer Science, University of Trondheim, N-7034 Trondheim-Nth, Norway
†Department of Computing, Imperial College, London SW7 2BZ, UK

mentation. This is achieved by a tight coupling between the various components of the formalism, so that the initial incomplete specifications have a rigorous underlying model with respect to which the specification is refined.

To show the feasibility of our approach a prototype has been developed. We have implemented the proposed approach using a process modelling tool (PPM2201) and a temporal rule manager. The rule manager controls the execution of the target system and manages a *temporal database*. Given a PPM specification, TEQUEL rules (a temporal logic rule based language) and C code are generated on which the temporal rule manager operates.

The purpose of this paper is to present the *main principles* behind our approach and to outline its potential in information systems development. Thus we provide brief overviews of the different components of the conceptual model, rule manager and temporal database, and refer the interested reader elsewhere for detailed descriptions. Using this model as a basis, we present an "ideal" development scenario based on the experiences gained during the work on the prototype. In addition, the prototype and its environment are briefly described and we show via an example how a system can be implemented using the available tools. Finally, we discuss the potential of our approach and outline further research directions.

2 The Conceptual Model

The conceptual model has three components: the Phenomenon Model [16],[17],[12], the Process Model [5],[12], and the External Rule Language [18]. The Phenomenon Model is an extended Entity Relationship Model and describes the static aspects of the real world, whereas the Process and Rule Models describe the dynamic aspects including the temporal dimension. Rules are also used to express constraints and derivations on the static model.

The Phenomenon Model

The static aspects of the real world is modelled by the Phenomenon Model. The basic modelling constructs are: entities, objects, connections, and data types. The model distinguishes between data entities and non-data entities. It contains a formalisation of Entity Relationship modelling constructs. An example of a Phenomenon Model is given in Figure 5. For a detailed description of the Phenomenon Model, see [16], [17], and [12].

The Process Model

The Process Model is an extension of regular dataflow diagrams and is used to specify processes and their interaction in a formal way. This includes both the interactions

between the processes at the same level of abstraction and how processes at any level of abstraction relate to their decompositions. The basic modelling concepts are processes, stores, external agents, flows (may denote both control[1] and data flow), ports, and timers. An example of a Process Model is shown in Figure 6.

The modelling is based on a *top-down approach*, where specifications are developed in an *incremental* and *iterative* manner. The developer starts out by identifying processes (business functions) at a high level of abstraction. Details are added to the specification as the developer gains new knowledge about the target system and its environment. The processes are successively decomposed into lower-level processes until an appropriate level of decomposition is achieved. At the lowest level of abstraction, where we have arrived at a set of *non-decomposed* processes, the process logic (i.e. the internal working of a process) is described by a subset of the External Rule Language (ERL). Rules may both *describe* and *constrain* processes at any level, but the model only requires them for describing the lowest level. This is further discussed below (see "The coupling between the models"). For a detailed description of the Process Model, see [5], [12] and [18].

The External Rule Language

The External Rule Language (ERL) is used to describe the internal workings of a process and to specify constraints that can be associated with processes at any level of abstraction. The ERL is based on first-order temporal logic, with the addition of syntax for querying entity-relationship model. The general structure of a rule is as follows:

WHEN <trigger condition> IF <condition> THEN <conclusion>

where the WHEN and IF parts are optional. A particular feature of the ERL is its temporal component, which is of crucial importance when modelling the time aspect explicitly at the analysis level.

ERL rules have both declarative and imperative semantics. For the imperative semantics to be effective, during design a rule must be classified as being one of the following:

Constraint rules: These specify conditions which must not be violated. For example,

NOT (employee has salary(amount) AND amount < 5000)

expresses the constraint that all employees must be paid more than 5000 currency units.

Derivation rules: Information can be derived from other information already present in the system via these rules, for example

IF NOT payment for accountnum(this) SINCE warning for accountnum(this)
THEN account [has num(this), has status("bad debt")]

[1] Also called *triggering* flow.

which states that an account has a derived status of being a bad debt, if no payment for this account has been received since a warning was issued.

Action rules: The system may be instructed to perform actions such as updating the database, and communicating with external agents via action rules, such as

```
WHEN order [from customer(someone), wants item(thing)]
IF NOT( account [of customer(someone), has status("bad debt")])
THEN AT(NOW+7*days, deliver(someone, thing))
```

which says that upon receiving an order, provided the customer is not a bad debtor, deliver the goods in a week's time.

For a detailed description of ERL, see [18].

The coupling between the models

The relationship between the Rule Model and the Process Model is depicted in Figure 1. Each non-decomposed process should have associated a set of ERL rules describing the behaviour of the process. In addition, one may optionally specify the behaviour of decomposed processes, these rules being interpreted as constraints on the behaviour of the rules describing the non-decomposed processes.

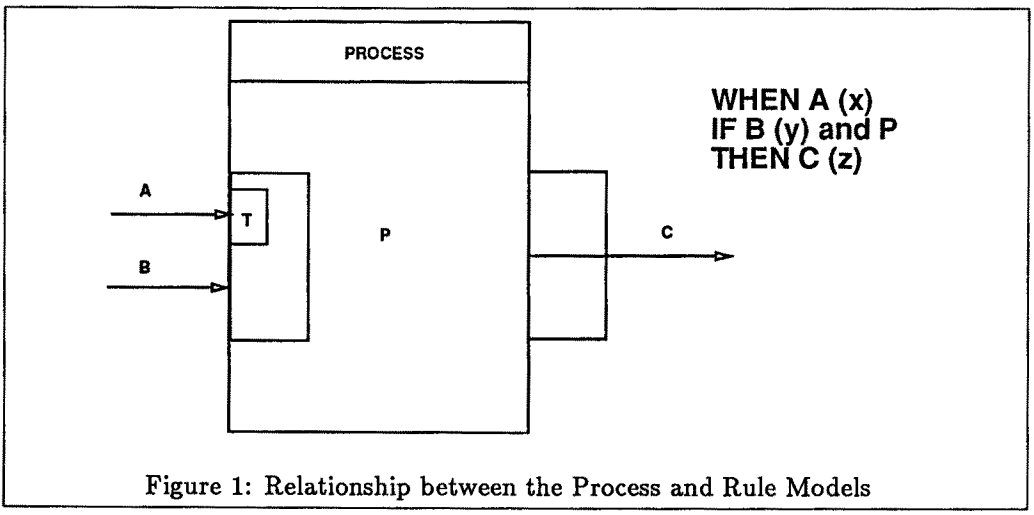

Figure 1: Relationship between the Process and Rule Models

The relationship between the rule model and the process model as depicted in figure 1, can be described as follows. The trigger part is extracted from the process structure (i.e., triggering flow) and corresponds to the WHEN part of an ERL rule. The conditional part is extracted from the process structure (i.e., non-triggering flow) and from the

process logic (expressed in a subset of ERL). This corresponds to the IF part of an ERL rule. The action part, that is, the THEN part of an ERL rule is extracted from the process structure (i.e. output flows).

The Process Model provides an overall structure to the ERL rules [18]. The ERL rules are grouped in clusters according to the process they are associated with.

3 Semantics of the Model

The Process Model is given semantics by an underlying temporal model of the dynamic aspects of the specified system, and it is via these semantics that the connection between processes and action rules can be made. In this section, we provide an informal description of the semantics of the underlying model, and refer the interested reader to [13] for a mathematically rigorous definition.

Rules and database changes

As the data in the system evolves over time, we model the most recent changes to the database in a simple temporal structure, as illustrated in Figure2. The δ and δ' represent the set of changes made to the database in moving from one version Δ^- to another Δ, as well as the most recent external input to the system. A simplistic view of the semantics of a single action rule (as given in the figure) refers to the set of changes δ which brought the system to the current database Δ, and to the set of changes δ' which will take the system to the next version of the database. The semantics of the action rule

WHEN trigger IF condition THEN action

are that whenever trigger was part of the set of changes δ, then provided condition is provable from the current database Δ (via a set of well-defined logical semantics, see for example [9]) the action must be part of the next set of changes to the database δ'. The definition of what it means for a trigger and an action to be part of a set of changes δ is intuitively easy:

- a trigger — which is either an insertion, deletion or update to the database, or an external input — is part of δ if there is a member of δ which can be unified with the trigger
- an action — which is a temporal formula with the atoms being either insertions, deletions or updates to the database, or external outputs — is part of δ if
  - for actions without temporal connectives, there is a member of δ which performs the appropriate insertion, deletion or update to the database

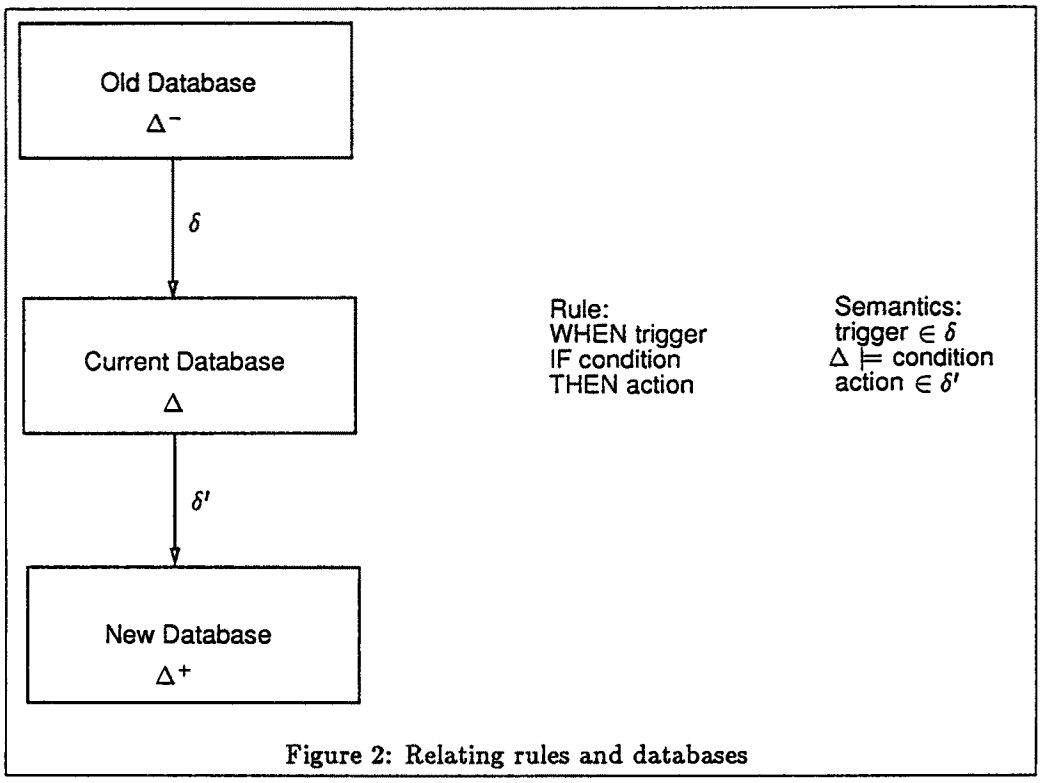

Figure 2: Relating rules and databases

- for actions with temporal connectives, there is a member of δ which is a commitment to perform the action in the appropriate future steps.

Given these semantics, we can derive a new database Δ^+ from the current database Δ, the most recent set of changes δ, and a single action rule. In practice, of course, there are many rules in use, and so we must extend the above definitions to cope with multiple rules. Moreover, assuming that we wish to implement transactions, we must provide mechanisms to hide the updates of each transaction from other transactions, until that transaction commits. While the model we use is rich enough to supply such mechanisms, it is beyond the scope of this paper to present the details here. The interested reader is again referred to [13].

Using temporal databases

In the discussion above the nature of the database, whether relational, deductive or object-oriented, was irrelevant, since the temporal model worked over complete databases. With TEQUEL, we have been using a *temporal database*, thus we have two flows of time.

The first is the flow of time along which the database changes, and the second is the flow of time which is modelled within each version of the database. Each individual version of the database can be imagined as a sequence of states as illustrated in Figure 3. At

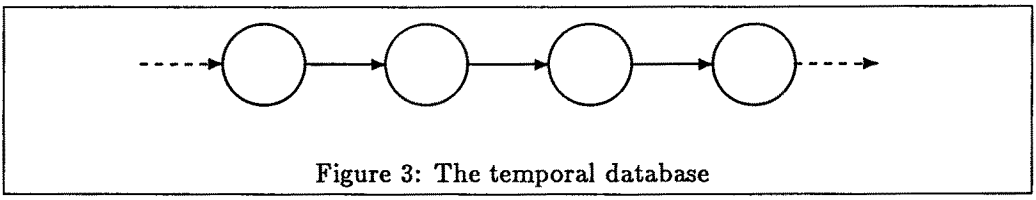

Figure 3: The temporal database

any given time, there is a distinguished state in the sequence known as the *current* state, which queries are evaluated with respect to. Queries can contain references to historical information, and relate previous values of information to current values. Languages for implementing ERL-like derivation rules have been devised by Abadi and Manna [1]; the semantics of such languages are well understood.

The temporal model which we use is therefore *two-dimensional*, with one-dimension tracking the database evolution, and the other dimension representing the ordeering of information within each version of the database.

4 The temporal rule manager

TEQUEL, the temporal rule manager, arises from a specialisation of work previously carried out into *executable temporal logics* such as USF [9] and MetateM [4]. A set of rules of the form

$$\text{formula about the past} \Rightarrow \text{formula about the future}$$

are evaluated with respect to a particular state in a temporal database, yielding a number of formulae about the future which must be made true, if they are not already true. The ERL rules in the specification can always be transformed into this "past $\Rightarrow$ future" normal form, thus we have a means of directly executing high-level specifications.

To tailor executable temporal logic to this specific use, we have created a temporal database using existing relational database technology, and put this under control of what are essentially compiled ERL rules. The rules can also access information for outside the system via calls to external programs, and can similarly initiate actions.

5 The "ideal" development scenario

Based on the experiences gained during the work on the prototype, we will first present an "ideal" development scenario assuming no limitations regarding the available tools[2]. We envision a development scenario close to the following:

1. **Develop conceptual model**
   Develop data, process, and rule specifications, that is, specifications at the analysis level (see description in Section 2).

2. **Generate database schemas from the Phenomenon model**
   A database schema is automatically generated from the Phenomenon model by a mapper.

3. **Add design details**
   The process structure[3] and the process logic (which are expressed in a subset of ERL) resulting from step 1 are further refined, that is, the process of design is undertaken. Activities such as identifying the boundary of the final target system, designing the user interface, and adding other design details to the specification are undertaken.

4. **Generate ERL rules from the process structure and process logic**
   Based on the design specification resulting from step 3, a rule model expressed in ERL may be generated. The description is complete, and the run-time system can be constructed from the rules without reference to the Process Model.Generate ERL rules from the process structure and process logic

5. **Generate TEQUEL rules from rule model**
   TEQUEL rules are generated from rule model (which is expressed in ERL).

6. **Run the system**
   The executable rules are fed into the rule manager which controls the actual execution of the rules. The database schemas and the executable rules (i.e., TEQUEL rules) constitute the specifications at the execution level (i.e., database level).

6 The functioning of the prototype

We have developed a prototype which integrates the PPM tool (called PPM2001) and TEQUEL, the temporal rule manager. The former is a tool for Process Port modelling developed by the Information Systems Group at NTH. The latter is developed at the

[2]The approach followed developing the example application in Section 7 is less ambitious due to the limitations of the prototype reported in Section 6.

[3]It should be emphasised that we refer to a process specification at the lowest level of abstraction (i.e. non-decomposed processes).

Department of Computing at Imperial College. The prototype is developed on SUN 3/60 workstations which run UNIX Sun OS 4.1. BIM-Prolog and PCE are used as the implementation languages.

In spite of practical limitations (e.g. the lack of a parser for ERL) the prototype has demonstrated the feasibility of our approach. The approach seems very promising with regard to the coupling between processes and rules including dynamic as well as temporal aspects. We have implemented the following: Given a PPM specification, TEQUEL rules and C code are generated on which the temporal rule manager operates. Here it should be emphasised that *both TEQUEL rules and C code are generated from the same initial specification* and it is performed *automatically.* The coupling between PPM and the temporal rule manager is depicted in Figure 4. In the next section, we

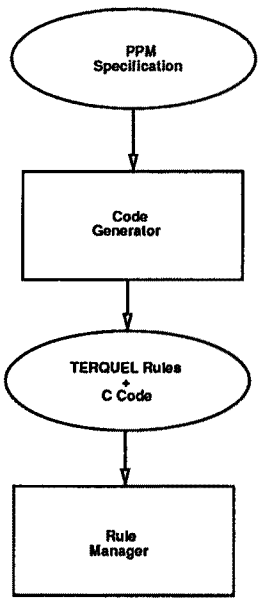

Figure 4: The coupling between PPM and TEQUEL

show how we can use the prototype to generate *executable* code directly from the PPM and PLD diagrams.

Compared to the scenario outlined in the previous section, the scope of the prototype is reduced in the following way:

- ERL is substituted by a formalism called PLD.
- Both TEQUEL rules and C code are generated from the initial specification instead of only TEQUEL rules.

The PLD formalism
Due to the lack of a parser for ERL, the declarative language is substituted by a formalism called PLD (Process Life Description). As mentioned above, PLD is used to describe the process logic of non-decomposed processes. It is a procedural language and provides the following constructs:

- RECEIVE: Receiving dataflow.
- SEND : Send dataflow.
- ASSIGNMENT:
- SELECTION : If-test.
- LOOPS : For and while-loops.

Figure 7 shows an example of a PLD specification. When the ports are designed in a PPM diagram, the skeleton of the PLD is generated automatically, taking care of the reception and sending of data. Further design details can then be added. By using this version of the prototype, it should be noted that we have collapsed step 1 and step 3 in the approach outlined in Section 5.

Generation of both TEQUEL rules and C code
From PPM's internal representation both TEQUEL rules and C code are generated. TEQUEL rules are generated from the PPM structure whereas C code is produced from the PLD specifications. Referring to the pattern depicted in Figure 1, the IF...THEN parts are implemented by external calls to C routines. The rule manager is extended (by adding a new predicate) in order to handle the processes implemented in C, and it is linked to a special interface to take care of the communication between C and BIM-Prolog.

7 An Example

To illustrate how a system can be developed, we will use an elementary banking example.

Banking Example

> The bank can open or close accounts for their clients. An account is identified by an account number and a client can hold one or more accounts. Furthermore, an account has to be opened before it can be closed or changed, otherwise an error message is to be issued by the system. The bank receives transactions in the form of debits and credits from their clients and the balance of the specified account is updated accordingly.
>
> A statement is produced whenever a change of the balance of an account has occurred and in addition, a statement is issued at the end of each week for each account.

The development steps from the specification of the banking example at the analysis level to an executable specification follows the scenario outlined in the previous section:

Step 1: Develop PPM specification
The PPM specifications are modelled by PPM2001 as shown in Figures 5, 6 and 7, respectively.

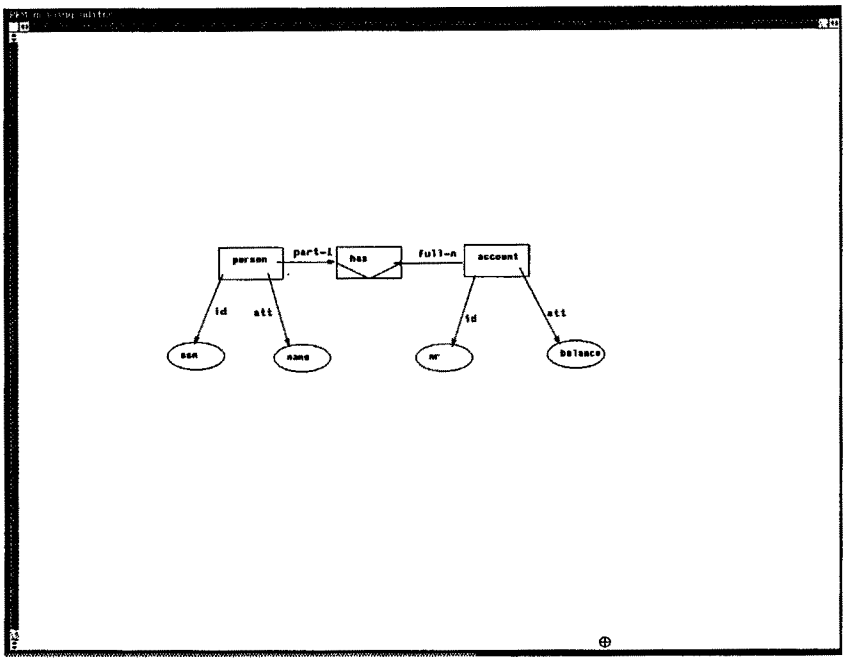

Figure 5: The phenomenon specification of the banking example

Step 2: Generate database schema from the Phenomenon model
From a subset of the phenomenon model (entities, objects, connections, and datatypes) a set of Prolog facts is generated. The flows to and from stores are linked to views of the static model and is in this way related to the Prolog facts.

Step 3: Generate TEQUEL rules and C code
From PPM's internal representation of the example, TEQUEL rules and C code are generated. TEQUEL rules are generated from the process structure whereas C code is produced from the PLD specification. Parts of the produced TEQUEL rules and the C code are shown in figure 8 and figure 9, respectively.

Step 4: Run the system
The specifications generated in step 3 are used as input to the temporal rule manager. Figure 10 shows the execution[4] of the specification during ticks 0 through 2.

[4]The rule manager is run in verbose mode.

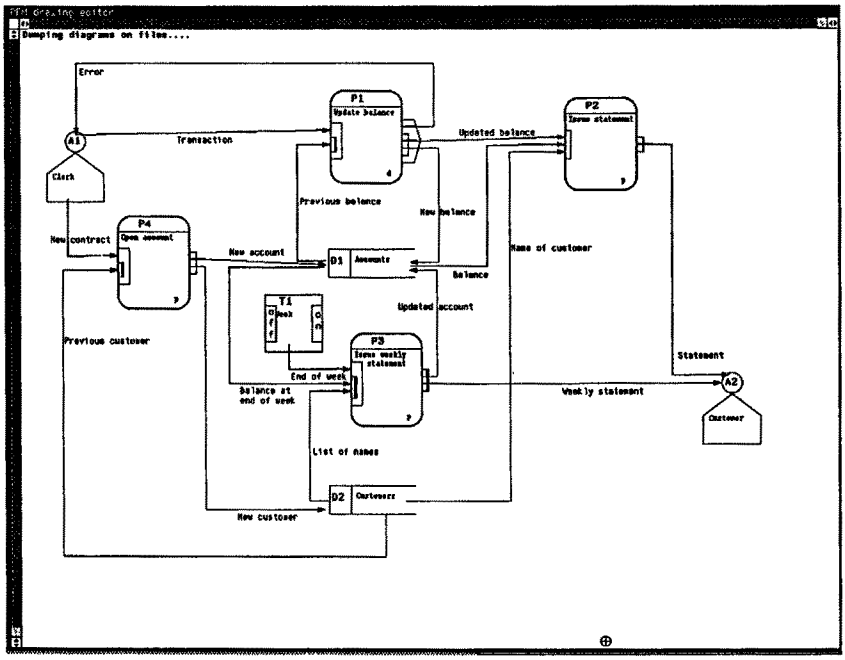

Figure 6: The process specification of the banking example

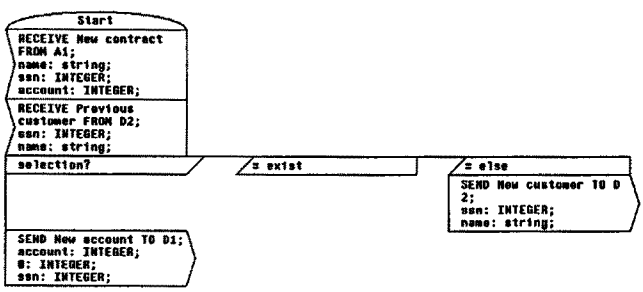

Figure 7: The PLD specification of process P4 in the banking example

```
                                        ⋮

          issuestatement(X1,X2)=action(
               dotrue:        {call cissuestatement(X1,X2)}
               );

          issueweeklystatement(X1)=action(
               dotrue:        {call cissueweeklystatement(X1)}
               );

                                        ⋮

rules
          issuestatement(X1,X2)    <=
                    <+ updatedbalance(X1,X2).

          issueweeklystatement(X1)    <=
                    <+ endofweek(X1).

          openaccount(X1,X2,X3)    <=
                    <+ newcontract(X1,X2,X3).

                                        ⋮
```

Figure 8: Some TEQUEL rules generated from the specification in step 1

8 Potential of the Approach

Based on the experience using the prototype, the proposed approach seems very promising with regard to the coupling between processes and rules including dynamic as well as temporal aspects. First of all combining the different model-orientations gives us a very *expressive language* where the static and dynamic aspects including the temporal dimension are covered. The approach allows for *explicit* modelling of temporal aspects and provides a *temporal database*, something which is necessary in order to deal with the evolutionary nature of organisations and accordingly, information systems development.

Furthermore, by using the process model we end up having a structured rule base avoiding the disadvantages of a flat rule base. The process model has also appeared to be very useful in the process of acquiring rules [7].

The proposed approach lends itself to *prototyping*. It is very easy to generate prototypes from a specification. By bridging the gap between specification at the analysis level and a specification at the execution level, we also opens up for the possibility of performing

```
                                    ⋮

pred_name=BIM_Prolog_string_to_atom("updatedaccount");
pred=BIM_Prolog_get_predicate(pred_name,2);
BIM_Prolog_call_predicate(pred
                ,BPM_IN|BPS_SIMPLE|BPT_INTEGER,account
                ,BPM_IN|BPS_SIMPLE|BPT_INTEGER,amount-2
                                );

printf("%d",account);
printf("%s",BIM_Prolog_atom_to_string(name));
printf("%d",amount-2);

                                    ⋮
```

Figure 9: C code fragments generated from the specification in step 1

maintenance at a higher level of abstraction.

9 Future Work

When the appropriate tools are developed, the scenario would be slightly changed from the current implementation. In order to arrive at an approach as described in Section 5, we plan to undertake a number of tasks:

- Refine ERL and develop an ERL parser.
- Develop an interface to a user interface design package.
- Develop method description/guidelines for how to develop systems following the proposed approach.
- Develop a general mapper from a phenomenon model to a database schema
- Extend the rule manager to handle parallelism.

In particular, the PLD will be substituted by ERL when a parser for the language is developed. When the process model is decomposed to an appropriate level of abstraction, the process logic is expressed in a subset of ERL. Based on the process structure and the process logic, TEQUEL rules may be generated according to the pattern outlined in Section 2. This will make calls to C routines unnecessary.

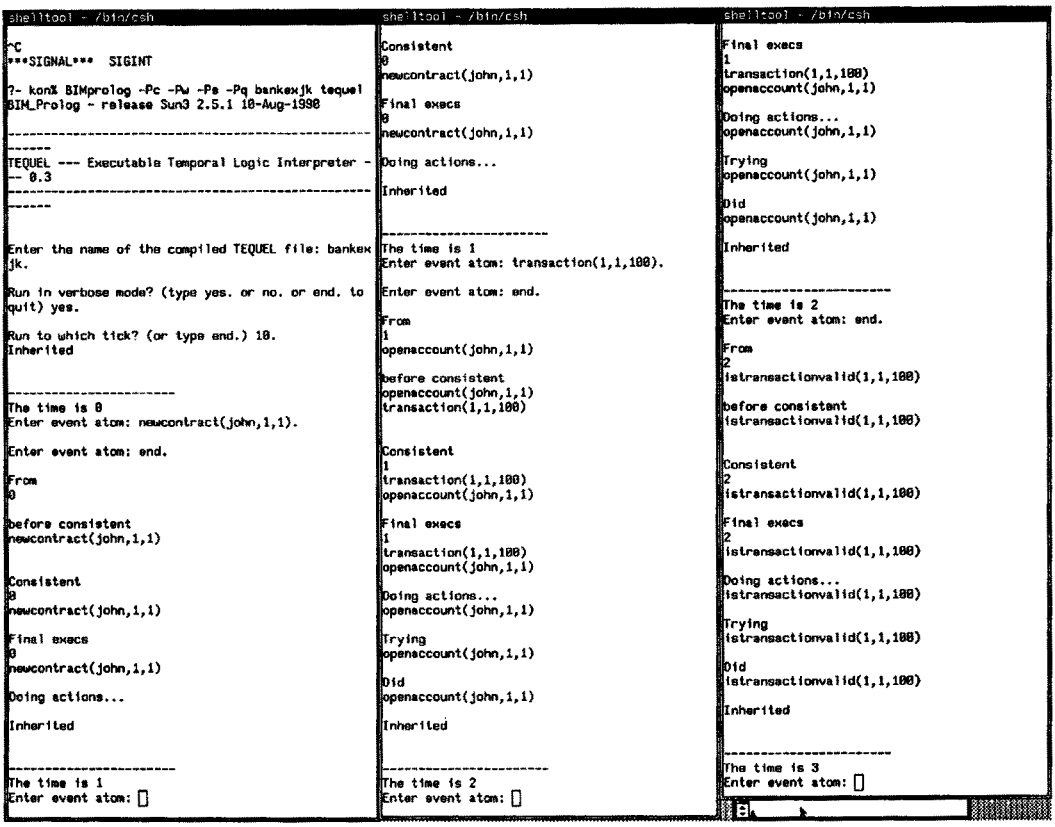

Figure 10: Running the system through the ticks 0, 1, and 2

In addition we have to look into how we can keep the structuring knowledge provided by the process model in the mapping step. In the current approach this information is lost and what is left is a *flat* rulebase. Thus, we may need techniques for pruning the solution space (selection of rules for execution) at the execution level. Applying clustering of rules/clustering hierarchies for pruning the solution space is heavily addressed in the Expert Systems area (see for instance [3]). In our context, we may achieve this by utilising the clustering of rules provided by the process model. However, how this can be done and at the same time take into account the efficiency aspect, has not yet been considered in detail.

10 Concluding Remarks

We have suggested an approach where we start out specifying a conceptual model in a top-down manner. The specification is successively refined in order to capture more details and accordingly, more powerful constructs in the language are introduced. Fi-

nally, we arrive at a specification from which code can be generated. The code (i.e., a set of rules) is used as input to a temporal rule manager which controls the execution of the final information system.

This approach seems very promising with regard to the coupling between different languages capturing different aspects of the real world. Both static and dynamic aspects including the temporal dimension can be described. Furthermore, the approach demonstrates how the gap between specification at the analysis level and a specification at the execution level can be bridged. This opens up for prototyping and also the possibility of performing maintenance at a higher level of abstraction.

11 Acknowledgement

The work reported has partly taken place in the ESPRIT II Project TEMPORA and partly as a collaborative research effort between The Information Systems Group at NTH in Norway and Department of Computing at Imperial College in the UK. We are grateful to our colleagues Rudolf Andersen, Dov Gabbay and Arne Sølvberg for their helpful comments.

The TEMPORA project is funded by the Commission of the European Communities under the ESPRIT R&D programme. The partners in the TEMPORA consortium are: BIM (Belgium), Hitec (Greece), Imperial College (UK), Logic Programming Associates (UK), SINTEF (Norway), SISU (Sweden), University of Liege (Belgium) and UMIST (UK).

References

[1] M. Abadi & Z. Manna: *Temporal Logic Programming*, IEEE Symposium on Logic Programming, 1987.

[2] G. Ariav: *Design Requirements for Temporally Oriented Information Systems*, Proceedings IFIP TC 8/WG 8.1 Working Conference on Temporal Aspects in Information Systems, May 1987.

[3] Barker et al: *Expert Systems for Configuration at Digital: XCON and Beyond*, Communications of the ACM, Volume 32, Number 3, March 1989.

[4] H. Barringer, M. Fisher, D. Gabbay, G. Gough & R. Owens, METATEM: *A Framework for Programming in Temporal Logic*, in REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness, Mook, Netherlands, June 1989. LNCS Volume 430, Springer-Verlag, 1990.

[5] S. Berdal, S. Carlsen: *PIP - Processes Interfaced through Ports*, Technical Report, IDT, NTH, 1986.

[6] A. Bolour, L. Anderson, L. Dekeyser, H. Wong: *The Role of Time in Information Processing: A Survey*, ACM-SIGMOD Record 12, 1982.

[7] N. Brummenæs: *A Practical Evaluation of RUBRIC*, Technical Report, IDT, NTH, May 1989.

[8] J. Bubenko: *The Temporal Dimension in Information Modeling*, 1977.

[9] D. Gabbay: *The Declarative Past and Imperative Future: Executable Temporal Logic for Interactive Systems*, in Proceedings of Colloquium on Temporal Logic in Specification, Altrincham, 1987, pages 402–450, LNCS Volume 398, Springer-Verlag, 1989.

[10] G.R. Gladden: *Stop the life-cycle, I want to get off*, ACM SIGSOFT, Software Engineering Notes, Vol. 7, No. 2, April 1982.

[11] D.D. McCracken, M.A. Jackson: *Life Cycle Concepts Considered Harmful*, ACM SIGSOFT, Software Engineering Notes, Vol. 7, No. 2, April 1982.

[12] A.L. Opdahl: *RAPIER - A Formal Definition of Diagrammatic Systems Specifications*, M.Sc. Thesis, Dept. of Electrical Engineering and Computer Science, IDT, NTH, 1988.

[13] R.P. Owens: *Notes on the TEMPORA Computation Model*, E2469/IC/3.4/7/1, December, 1990.

[14] A.H. Seltveit, K.W. Løvseth, P. McBrien: *"External Borrowing" - A Subset of the Library Case Study*, E2469/SINTEF/T5.1/19/1, October, 1990.

[15] A.H. Seltveit, J. Krogstie: *The Design Layer in TEMPORA*, E2469/SINTEF/T5.1/20, October, 1990.

[16] A. Sølvberg: *Software Requirement Definition and Data Models*, Proceedings Conference on Very Large Data Bases, October 1979.

[17] A. Sølvberg: *A Contribution to the Definition of Concepts for Expressing Users' Information Systems Requirements*, Proceedings International Conference on Entity-Relationship Approach to Systems Analysis and Design, December 1980.

[18] TEMPORA: *Concepts Manual*, September 1990.

[19] K.W. Løvseth: *TEMOCCA: TEMPORA Modeling Concepts - A Case Study*, E2469/SINTEF/T5.1/14, 1990.