A Rule Language to Capture and Model Business Policy Specifications

Peter Mc Brien[1], Marc Niézette[2], Dionysios Pantazis[3], Anne Helga Seltveit[4], Ulf Sundin[5], Babis Theodoulidis[3], Gregoris Tziallas[6] and Rolf Wohed[7]

Abstract

The TEMPORA paradigm for the development of large data intensive, transaction oriented information systems explicitly recognises the role of organisational policy within an information system, and visibly maintains this policy throughout the software development process, from requirements specifications through to an executable implementation.

This paper introduces the External Rule Language of the TEMPORA conceptual modelling formalism, and describes how it is used to captured and model business organisational policy. The syntax and semantics of the language are presented, together with a number of examples drawn from a realistic case study.

Introduction

Much has been written about the benefits and problems of contemporary approaches to the development of large scale information systems, some examples being [Mad83, Oll83, Oll86]. Despite the undeniable achievements of these various approaches, two challenges still remain unresolved. Firstly, the issue of maintenance and system evolution continue to be a major problem in commercial software, and secondly, the information system is of little use in developing any changes to business policy. These two problems may be seen as a manifestation of the same deficiency in current approaches; namely that there is very little explicit correspondence between *business policy* and the *program* of an information system which implements it. Business policy is often embedded together with programming code, the function of which is concerned with such issues as file accessing, sequencing of operations, and so on. As a consequence, the information system can not be easily examined at any time to ascertain whether it is still aligned to the business policy, making maintenance difficult, and the use of the information system for business modelling impossible.

1. Imperial College, Dept. of Computing, 180 Queen's Gate, London SW7 2BZ, UK
2. Service d'Informatique, Universite de Liège, B-4000 Liège, Belgium
3. UMIST, Dept. of Computation, Manchester M60 1QD, UK
4. Div. of Computer Science, University of Trondheim, N-7034 Trondheim-Nth, Norway
5. Infologics, Goteberg, Sweden
6. Hitec Ltd, 18 Poseidonos Avenue, Killithea, Athens 17674, Greece
7. SISU, PO Box 1250, 16428 Kista, Sweden

To illustrate the differing program solutions that a simple statement may be given, consider the implementation of a policy rule ‘every customer must have an account’. This may be achieved in a variety of ways, some of which are:

(i) Store both *customer* and *account* in the database, and make any update of *customer* update the *account*, and any update of *account* update the *customer*.
(ii) Store just the *customer*, and derive from that the existence of an *account*.
(iii) Store just the *account*, and derive from that the existence of a *customer*.
(iv) Store both *customer* and *account* in the database, and raise an error if the number of each should ever differ.

Although the choice of implementation will effect the overall efficiency and data security of the information system, it is not a consideration that needs to be taken into account at the *analysis phase*, which is concerned with capturing the functions and policies of a business as a *conceptual model*. The additional information involved in making the policy executable acts to obscure the original rule that we are attempting to model. The decision process involved in constructing executable specifications from the conceptual model should be left until a distinct *design phase*. The TEMPORA paradigm [Lou90] proposes that developers should be provided with a process which assists both in modelling business policy, and in linking this policy to the software development process. To this end, three conceptual models are provided:

- An extended entity-relationship formalism with temporal semantics (the *ERT*) provides a graphical representation of the *structural* aspects [The90].

- A modified *Process Interaction Diagram* (*PID*) provides a graphical representation of the *behaviour* aspects [Kro91].

- The *External Rule Language* (*ERL*) is used to describe business policy in terms of its affect on the ERT and PID. For the ERT it describes constraint rules on the number and relationships of entities, and for the PID it describes the details of the flow of information, and its interaction with external processes.

The ERL provides a formal logic language in which to describe *business policy*, which is in this context statements about the structure of the business (as represented by the ERT), and statements about the behaviour/procedures followed by the business in the use of the data in carrying out its activities (as represented by the PID).

In this paper we shall concentrate on how business policy may be modelled by rules in the ERL, introducing concepts of the ERT and PID models as necessary. To begin with, we give a brief description of some related work, before giving an overview of the ERL, and conclude by outlining the decision process necessary when the rules of the ERL are translated by a design process into executable specifications.

1 Related Work

Approaches to conceptual modelling have been defined for many strands of computer science and other sciences. However, approaches used in modelling the requirements of information systems are usually related to a corresponding development method. A review of the work in this area is reported in a series of conferences by the IFIP WG8.1 [Oll82, Oll83, Oll86].

In general, these approaches can be categorised according to the method’s strongest orientation, namely *data oriented* and *process oriented*. In each case, the name indicates the prominent construct(s) of the approach. For example, data oriented methods emphasise the structure of the application domain and are concerned with its passive components [Ver82]. Process oriented methods emphasise the behaviour of an application domain and describe application activities, their inter-relationships and how they are decomposed [Ros77]. Mixed approaches also exist, termed *logic oriented* [Søl90], and in addition, formal approaches [Jon81] and approaches based on knowledge representation formalisms such as semantic networks and frames [Bro82] have been proposed.

Recently, a number of conceptual modelling formalisms have been proposed that are not attached to a specific development method but which are of great importance since they represent a key development in the state of the art in the area of information systems in general, and requirements specification in particular [Bor85, Ber86, Dub86].

Whilst all of the above approaches cater for the representation of business policy in the requirements specification, none of them explicitly distinguishes where this policy is defined. In addition, few of them employ a declarative, rule-based language that permits the specification of temporal business rules in a user oriented way. In this paper, it is argued that business policy is best acquired and represented through the use of a declarative, rule based specification language with semi natural language semantics.

The TEMPORA approach is based on that of RUBRIC [vanA88, Lou89], extended by utilizing a commercial DBMS as the underlying data management mechanism, and providing capabilities for modelling and management of temporal data and business rules.

2 An Example of the Tempora Approach in Modelling Business Policy

To illustrate the use of the ERL in describing the structural characteristics of a business and its policy rules, we must first construct ERT and PID models for that business. As an example for this paper, we have used a case study of the Irish Electricity Supply Board's (IESB) debt management procedures made by the Rubric ESPRIT project [Rub89], and in particular focused on the IESB's *arrangement handling* system.

2.1 Structural Information

An *arrangement* is an agreement between a customer and the IESB for the customer to make some set of fixed payments (*instalments*) at given dates. Both the *instalments* and actual payments (a particular *type* of *movement*) made must be recorded for future reference. The ERT model for the arrangement handling system is given in figure 2.2; figure 2.1 describes key aspects of the model's constructs.

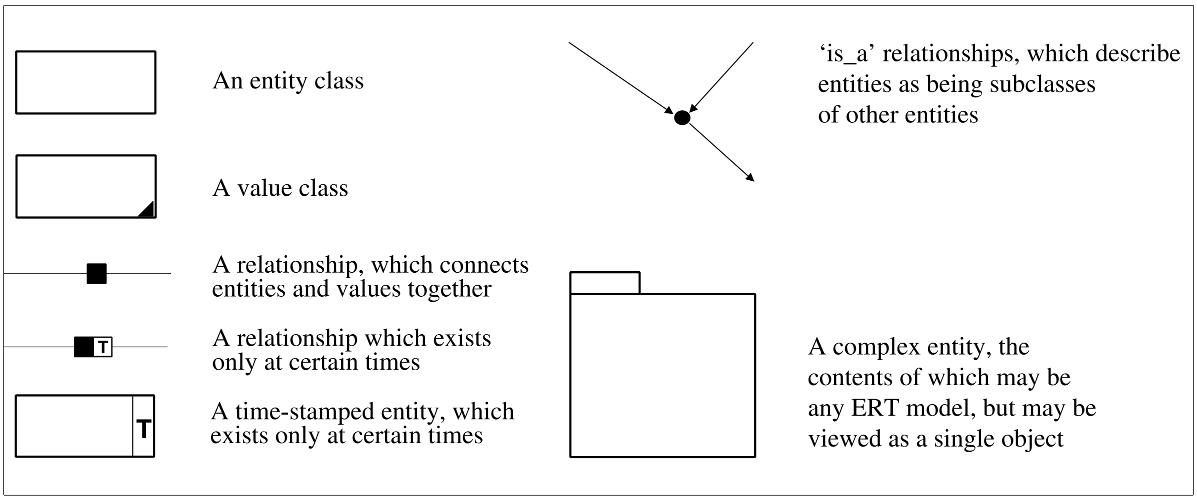

Figure 2.1: Description of ERT Model Constructs

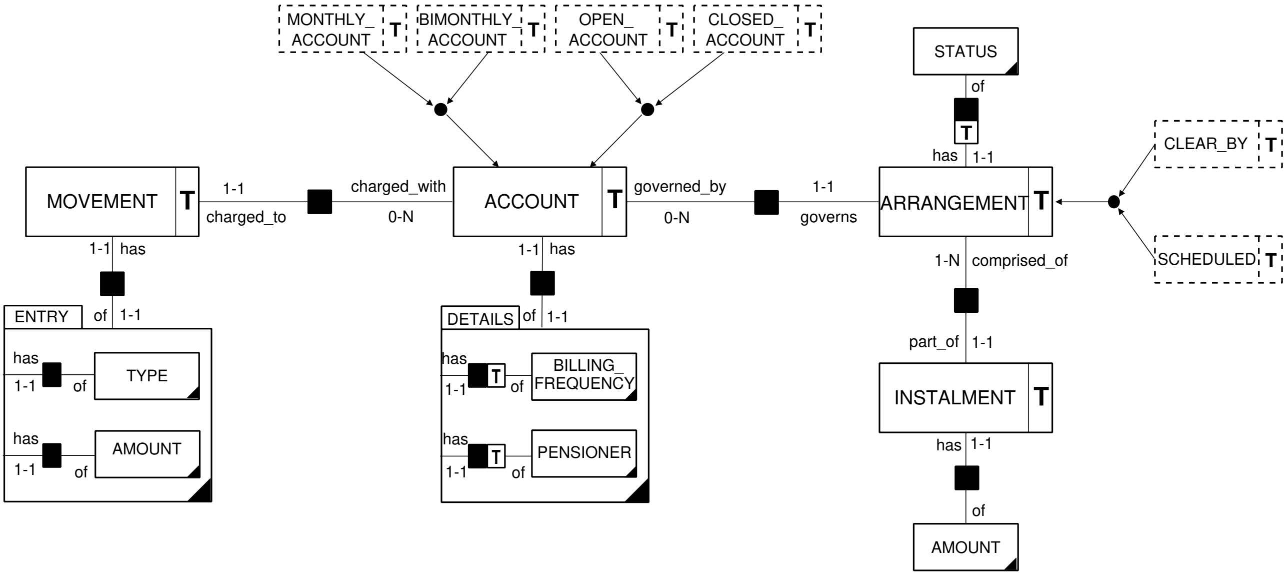

Figure 2.2: ERT Model for the IESB Arrangement Handling System

We mark entities such as *account* as time varying since we are interested in knowing the times at which the entity exists (which for the example of an *account* may be considered as the period it is open). As a design decision (not taken as the initial capture stage) we can mark entites as derived from other entities, denoted graphically by dotted outlines for the entity boxes. Complex entities may be considered to be *complex values* if they contain only value classes, and marked accordingly. Note that certain business policy rules will relate to the ERT model. For instance, in the case study it was found that there was a limit of 1,300,000 to the number of accounts permitted to exist at any time, and that would naturally be considered as part of the description of the *acount* entity.

2.2 Behaviour Information

The behaviour of the business, that is the procedures that it follows in processing structural information (the ERT) is described initially using the PID model. This shows any policy rule which describes the processing of information as a *process* box, with *dataflows* representing the inter-change of information between these rules and the *datastores* (views on the ERT) of the business, and any person or system outside the system being modelled as an *external agent*. Figure 2.3 illustrates the PID for the IESB example; figure 2.4 describes the constructs of the PID model we have used.

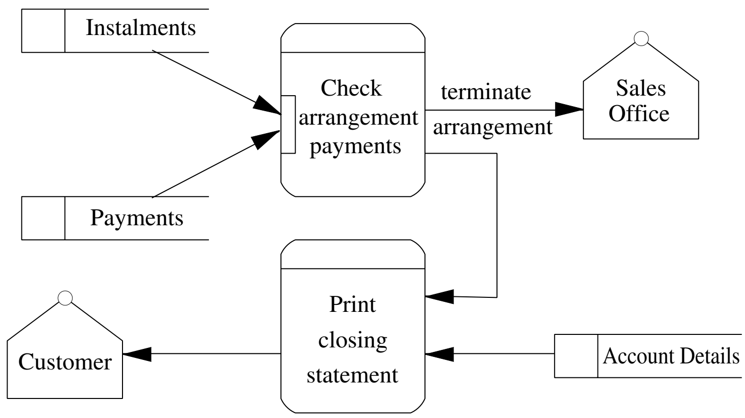

Figure 2.3: PID for the arrangement checking in IESB

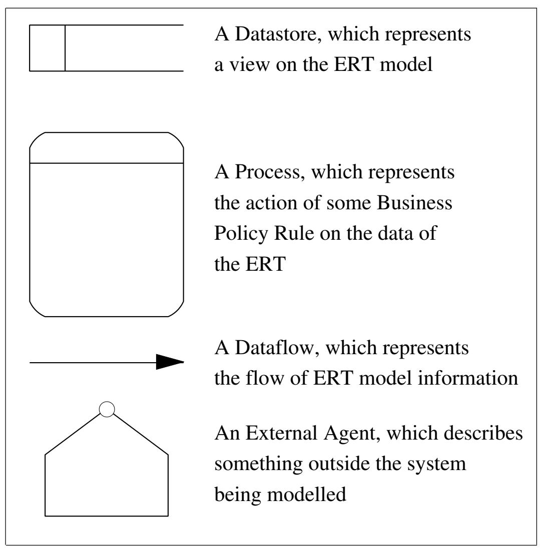

Figure 2.4: Description of PID Constructs

3 The External Rule Language

All ERL rules are given a single general structure, interpreted as describing some logical constraint on the ERT model, which must hold at every moment, or *tick*, whilst the information system is active. The system analyst shall regard the execution of the information system as stepping through a discrete series of these ticks, and at each tick ensuring that all the ERL rules hold, the system taking such actions as is necessary to make this so.

The basic structure of an ERL rule is given by the list below. The *consequence*, *condition* and *event* fields all have the single syntax of an ERL *expression*, and any free variables are implicitly universally quantified across the whole rule.

- **when** <event> **if** <condition> **then** <consequence>
- **when** <event> **then** <consequence>
- **if** <condition> **then** <consequence>
- <consequence>

The semantics of the fields are as follows: if at any tick the *event* of a rule has just begun to hold, and *condition* holds, then *consequence* must also hold. The *event* or *condition* fields may be omitted, in which case they are taken to always hold. For example, the statement of the *consequence* alone states that it must hold at all ticks of run-time system.

3.1 ERL Expressions

The basic elements of ERL expressions are the instances of classes and data-types from the ERT model. Variables may be used as place holders for any such element. ERL expression are based on three types of constructs; namely

(i) a *selection* of data from the ERT model, using relationships to restrict the selection of instances from classes,
(ii) *sets* of data obtained from the ERT, and
(iii) *predicates*, which name tuples of data selected from the ERT.

The basic expressions are constructed into compound expressions by

(i) the usual connectives of classical first order logic, and
(ii) a group of temporal connectives, which transpose the effective time at which the query is evaluated, by using the historical database to 'simulate' the data at past or future times.

The following sections describe the ERL under the headings listed above, and refer to the ERT and PID models for IESB example given earlier. When giving examples of ERL rules, any language tokens are given in **emboldened typeface**, so as to emphasize the structure of the rules.

3.2 Selecting Data from the ERT Model

For ERL rules to constrain and select data from the ERT model, we must be able to describe in a very general manner the components of the model. In particular, we will wish to describe not only the members of the entities and value classes, but also name specific instances of value classes in rules, and relate them to each other. For instance, one may wish to find all the *amounts* above ten thousand, but the data item ten thousand is not in the model. It is thus necessary to extend notionally the ERT model to contain value classes for the basic types used, and provide relationships to the value classes in the model which used those basic types. Using this extension, we are able to describe the relationship between various members of ERT classes and data-types in a uniform manner, by naming a ‘chain’ of class/data-type instance related to another class/data-type instance. A list of relationships with a single instance may given, enclosed in square brackets. A variable in parenthesis after the class name obtains variable bindings for the instances taken from a class. The following expressions relating to the IESB ERT model will illustrate this method.

- account(a) governed_by arrangement comprised_of instalment(i)

  Finds all pairs of *account* references *a* and *instalment* references *i,* related to a common *arrangement.*

- movement(m) [has entry [has type = ‘payment’,
  has amount < 2000],
  charged_to account [governed_by arrangement(a),
  has details has billing_frequency = ‘monthly’]]

  Finds all pairs of *movement* references *m* and *arrangement* references *a* related to a common *account*, for which the *movement* is of type *payment* and has a value less than 2000, and the *account* is on a *monthly* billing frequency.

  When comparing this ERL rule with an SQL statement (to achieve a roughly equivalent query), we can see the benefits that having an ERL bring about in clarity, and relating rules to our conceptual models instead of the run-time system.

```
SELECT   m,a
FROM     movements,arrangements,account
WHERE    movements.m=account.m AND account.a=arrangement.a
         AND type = ‘payment’ AND amount < 2000
         AND billing_frequency = ‘monthly’
```

- $1 \leq$ amount(a) ≤ 100

  Finds all instances *a* of value class *amount* in the range 1 to 100.

- amount(a) **and** $1 \leq a \leq 100$

  Equivalent to the previous example.

3.3 Collecting Sets of Data from the ERT Model

A set collection construct is provided, together with a group of set operators. This allows us to group all variable substitutions for which an expression holds, and perform operations such as **number_of** to count the number of instances, or **intersect** to find common substitutions from two set expressions. For example, we can enforce a cardinallity constraint on the number of accounts permitted to be present in the system at any single time by:

number_of {a **for_which** account(a) } < 1,300,000

3.4 Predicates

The ERL predicates shall have no more meaning other than being table names for tuples of data, but allow several ERL rules to ‘share’ common expressions or data, by naming it in a predicate. They take the syntactic form *predicate_name(ERT_data , ERT_data)*, where there can be any number of arguments, describing instances of entites and values of the ERT. During the specification of a TEMPORA system, the system analyst will use a *predicate table* to give a textual description each predicate's meaning. Two situations necessitate the use of predicates:

- To name the dataflows and external agents of the PID with which the system must communicate. For example, the analyst may talk about the predicate *sales_office(x)*, represent it in the PID as an external agent (i.e. outside the system being modelled) and describe it as using a certain printer to print out a report for the sales office that an account *x* has had its arrangement terminated due to the customer failing to meet instalment deadlines. It is the responsibility of the programmer during the design phase to realize this, for example by calling some external report generator which can print cheques in the correct format.

- When the system analyst wishes to avoid the specification of certain functions of the system which are better handled by a programmer, and leave them to be made in the design phase. These will represent the internal details of the processes in the PID. For example, the system analyst may require to know the total value of all *instalments* made for a certain *arrangement*, but feel unhappy as writing a specification in the ERL to achieve it. By naming a predicate *sum_instalments(arr,total)*, and giving a textual description of its meaning, the system analyst by delegate this task to the programmer responsible for mapping the ERL to executable TL rules in the design phase.

3.5 Logical Connectives

The connectives for conjunction (**and**), disjunction (**or**) and negation (**not)** from classical first order predicate logic are provided, and given their usual meaning. The following examples illustrate their use:

- *account(e) charged_with movement(m)* **and** *account(e) governed_by arrangement(a)*

  Find sets of instances for the variables *e*, *m* and *a*, where the *e* is given the same value (instance of the *account* entity) in both ERL reference expressions.

- *account(e) charged_with movement(m)* **and not** *account(e) governed_by arrangement*

  Finds sets of instances for the variables *e* and *m* for which the first ERL reference holds, but the second does not.

- *account(e) charged_with movement(m)* **or** *account(e) governed_by arrangement(m)*

  Finds sets of instances for *e* and *m*, which hold for either (or both) of the ERL reference expressions.

3.6 Temporal Connectives

The underlying temporal database allows us to express the entities and values of the ERL as being ‘present’ in the current, future or past database, and thus phrase rules in terms of current, expected or old records. In reality all information is present in one database, but the ERL provides an abstraction mechanism to make it appear as a series of databases over time. This ‘shifting’ of time at which we inspect the information in the database can be achieved either implicitly with reference to the present time (‘was something true sometime in the past’), or explicitly (‘was something true at noon on Tuesday’).

For example, having marked the *account* entity as time varying in the ERT model (by placing a **T** in the class box), we can find if an account is *open* by seeing if it is present in the database, and can define a *closed* account as one which used to be present in the database (**sometime_in_past**), but is not longer present. Note here how we derive the existence of an entity in the present from information about entities which held in the past.

if account(a) **then** open_account(a)

if not account(a) **and sometime_in_past** account(a) **then** closed_account(a)

To use explicit time references to time, the ERL provides intervals and points, along with operations on them. The time is described in the format *Hour:Minute:Second*, and the date in the format *Day/Month/Year*. For example the following ERL rule derives a new entity *account_closed_today* from knowing that there was an account *e* at midnight, but that account *e* no longer exists.

if account(e) **at** 00:00:00 **and not** account(e) **then** account_closed_today(e)

We may also restrict the evaluation of rules to a given combination of time and date using the **time_is** predicate to check the time. For example, to make the external procedure *print_account_details* be used at noon each day to print the information about all *accounts*, we could say:

if account(e) **and time_is** 12:00:00 **then** print_account_details(e)

To view the data in a time independent manner, and thus achieve the effect of a non-temporal database, the **at_any_time** connective is used, which finds if its argument holds for any time past, present or future. For example, to define a clear-by arrangement as one which only has one instalment during its entire history, one could write

if number_of {i **for_which at_any_time** instalment(i) comprises arrangement(a) } = 1
then clear_by(a)

The conciseness and expressive power of the temporal connectives may be demonstrated by the representation in the ERL of this complex business rule:

if it is seven days after an instalment was due on a scheduled arrangement, and the sum of all payments made during the period of the arrangement is found to be less than the sum of all previous instalments and half of the last instalment, then the arrangement shall be terminated.

This may be translated to the following ERL rule, which shows a close relation to the original business rule:

```
if 7*days ago
      ( instalment [ has amount(install), comprised_of scheduled(a) ]
        and previous_instalments(a,previous_i)
      )
and previous_payments(a,previous_p)
and previous_p < previous_i + 0.5*install
then terminate_arrangement(a)
```

The expression *time* **ago** *exp* holds for each instance of *exp* which held at *time* before the present, so in the example we transpose the time we are looking for an *instalment* to seven days before the present time. Working at the time of the instalment, we sum all *instalments* due for the *arrangement*, and outside the scoop of the **ago** connective, i.e. working from the present, we sum all the payments made into the *account* whilst governed by the *instalment*. The example demostrates how sub-class name (*scheduled*) can be used instead of its parent entity (*arrangement*). The predicates *previous_payments* and *previous_instalments* abstract away details of how to sum the values of instalments and payments, leaving them to be specified later in analysis, or even in the design phase. In this case it is natural to define these predicates as ERL expressions:

if previous = **sum** { p **for_which sometime_in_past** movement
    [has details [has amount(p),
        has type='payment'] ,
    charged_to account governed_by arrangement(a)]
then previous_payments(a,previous)

if previous = **sum** {old **for_which**
    sometime_in_past instalment [has amount(old), part_of arrangement(a)] }
then previous_instalments(a,previous)

4 Translating the ERL to Executable Rules

At the analysis level the ERL formalism does not distinguish explicitly between different rule types, in order to relieve the analyst from procedural considerations. However, this is can not be true for the design level, where these procedural semantics are a necessary and important consideration in the construction of an executable specification of the system [Kro91]. The two stages in this process are described below.

4.1 Categorizing Rules

The initial stage in this design process is to categorize the rules into classes according to their implementation. By choosing one of the three classes it is decided whether an ERL rule is to be used to ensure data security (constraint), provide deductive information (derivation) or cause changes to be made (action) in the run-time system.

- *Constraint rules* which are concerned with the integrity of the ERT components. They can be further subdivided to *static constraint rules* which are expressions that must hold in every valid state of an ERT database, and *transition constraint rules* which are expressions that define valid state transitions in an ERT database. An example of an static constraint rule is 'the number of accounts must never exceed 1,300,000', and of a transition constraint rule is 'a closed account may never be re-opened'.

- *Derivation rules* which are expressions that define the derived components of the ERT model in terms of other ERT components (which themselves may be derived). There must be exactly one derivation rule for each such component. As with constraint rules, derivation rules can be subdivided into *static derivation rules* and *transition derivation rules,* depending on whether the derived ERT component is time-stamped or not. Since the IESB ERT model has only time-stamped derived entities, we will have no static derivation rules for our case-study. An example of a transition derivation rule is 'a clear-by arrangement is one which has exactly one instalment'.

- *Action rules* which are concerned with causing some event to occur, which may be an update to the database, communication with some external agent or the invocation of some process. An example of an action rule is 'when an arrangement is terminated, at sometime print a statement for the account which had the arrangement'.

The inclusion of a temporal modal operators (such as **sometime_in_future**) in the ERL allows for the greater expressive power in a first order logic language.

As specified previously, constraint rules express restrictions on the ERT components by constraining individual ERT states and state transitions, where a state is defined as the version of the database at any tick. More specifically, static constraint rules represent definitions of conditions which must hold between different classes (entity, value or relationship classes) in any individual state, independent of the time that the state relates to. These rules are also known variously as *extensional constraints* [Cli83], *functional dependency rules* or *multi-valued dependency rules*, and their mapping to first order logic formulas is well defined [Nic78a]. Transition constraint rules place restrictions on two or more states of the database, by specifying valid state progressions. This type of rule can be expressed directly in the proposed formalism, because of the explicit modelling of the evolution of data. These rules are also known variously as as *intentional constraints* [Cli83], *dynamic constraints* or *constraints upon update operations* [Smi77, Nic78b].

4.2 Considering the Computational Model

The ERL specifically avoids having a computational model. However the target run-time language which only be able to handle rules which fit its computational model, and will handle certain computations more efficiently than others. These considerations may lead to some changes being necessary in the ordering of terms within a rule. An simple example will clarify this point: we are given the business rule that 'at the end of the year, all open accounts without a movement should be closed'. This might result in the following ERL rule:

> **when end_of_year**
> **if** **not** account(x) charged_with movement **during this_year**
> **and** open_account(x)
> **then** shut_account(x)

The run-time rule language in TEMPORA provides a temporal Prolog-like query language, and thus will not permit a free variable inside the scope of a negation operator. We must therefore place the *open_account* term before the negated expression to ensure that the variable is ground when it is checked for a movement.

Conclusions

The ERL provides a suitable platform in which to express the textual rules and procedures of a business, in the context of the ERT and PID models. It provides a declarative language for the system analyst to use in modelling business policy rules in an information system, without need to consider issues relating to the execution of those rules until a distinct design phase. In using these three models in the modelling of business specifications, the TEMPORA project develops upon current approaches to information systems in provideing a clear separation between models to capture external specifications, and the internal run-time model of the information system, whilst providing a systematic method of linking the two.

Future work will need to develop further the link between the ERL, ERT and PID models, in terms of providing validation between the models, and to investigate the semi-automated production of the run-time model, involving human interaction with a design tool.

Acknowledgements

The work reported in this paper has been partly funded by the Commission of the European Communities under the ESPRIT R&D programme. The TEMPORA project (number E2469) is a collaborative project between: BIM, Belgium; Hitec, Greece; Imperial College, UK; LPA, UK; SINTEF, Norway; SISU, Sweden; University of Liège, Belgium and UMIST, UK. SISU is sponsored by the National Swedish Board for Technical Development (STU), ERICSSON and Swedish Telecomm.

We would like to thank all the follwing members of the TEMPORA project for their contributions during the design of the ERL: R. Andersen, P. Bergsten, J-L. Binot, J. Bubenko Jnr, G. Diakonikolaou, D. Gabbay, P. Loucopoulos, R. Owens, U. Persson, F. Schumacker, A. Sølvberg, P. Vasey, B. Wangler and P. Wolper.

References

[Ber86] Berztiss, A. *The Set-Function Approach to Conceptual Modelling*, in 'Information system design methodologies: improving the practice' Olle, T.W., Sol, H.G, Verrijn Stuart, A.A., (eds), North Holland (1986).

[Bor85] A.Borgida, *Language Features for Flexible Handling of Exceptions in Information Systems*, ACM Transactions on Database Systems, Vol. 10, No. 4, December 1985, pp. 565-603.

[Bro82] Brodie, M.L . & Silva, E. *Active and Passive component modelling (ACM/PCM)*, in 'Information system design methodologies: a comparitive review', eds Olle T.W., Sol H.G. and Verrijn Stuart A.A., North-Holland (1982) .

[Cas79] Casanova M.A., Bernstein P.A *The Logic of a Data Manipulation Language*, In Conference Record of the Sixth Annual ACM Symposium on Principles of Programming Languages (San Antonio, Texas), ACM, New York, 1979.

[Cli83] Clifford J., Warren D.S. *Formal Semantics for Time in Databases* ACM TODS Vol.8, No.2, June 1983.

[Cli85] Clifford J. *Towards an Algebra of Historical Relational Databases* ACM SiGMOD Intern. Conf. on Management of Data, Austin, Texas, May 1985.

[Dub86] Dubois E., *et al* *The ERAE Model : A Case Study* in [Oll86].

[Jon81] Jones C.B., *Software development: a rigorous approach*, Prentice Hall, London, 1981.

[Kro91] Krogstie J., McBrien P.J., Owens R.P., Seltveit A.H., *Coupling between Process and Rule Based Approaches*, CAiSE '91.

[Lou89] Loucopoulos, P. *The RUBRIC Project - Integrating E-R, Object and Rule-based Paradigms*, Workshop session on Design Paradigms, European Conference on Object Oriented Programming (ECOOP), 10-13 July 1989, Nottingham, U.K.

[Lou90] Loucopoulos P., *et al* *TEMPORA - Intergrating Database Technology, Rule Based Systems and Temporal Reawoning for Effective Software*, Esprit '90 Conference Proceedings, Kluwer Academic Publishers (1990).

[Lun89] Lundh J. & Rosengren P., *HYBRIS - A First Step Towards Efficient Information Management,* SISU rapport nr 5 (1989), Swedish Institute for Systems Development, Sweden

[Nic78a] Nicolas, J.M., Gallaire, H. *Data Base: Theory vs Interpretation,* In *Logic and Data Bases*, Gallaire, H., Minker, J. (eds), Plenum Press, New York, 1978.

[Nic78b] Nicolas, J.M., Yazdanian, K. *Integrity Checking in Deductive Databases*, In *Logic and Data Bases*, Gallaire, H., Minker, J. (eds), Plenum Press, New York, 1978.

[Oll82] Olle, T.W., Sol, H.G., Verrijn Stuart, A.A. *Information systems design methodologies: a comparitive review*, IFIP WG 8.1 CRIS I, North Holland (1982).

[Oll83] Olle, T.W., Sol, H.G., Tully, C.J. *Information systems design methodologies: a feature analysis*, IFIP WG 8.1 CRIS II, North Holland, 1983.

[Oll86] Olle, T.W., Sol, H.G., Verrijn-Stuart, A.A. *Information Systems Design Methodologies: Improving the Practice*, Proc IFIP WG 8.1 Working Conference on Comparative Review of Information Systems Design Methodologies: Improving the Practice, Noordwijkerhout, The Netherlands, 5-7 May, 1986.

[Ros77] Ross, D.T. and Schoman, K.E. *Structured Analysis for requirements definition*, IEEE Trans SE 3(1), 1977, p.p. 1-65.

[Rub89] *Case Study Report: Irish Electricity Supply Board,* UMIST, Rubric Esprit Project 928

[Smi77] Smith J.M., Smith D.C.P. *Database Abstractions: Aggregation and Generalization*, ACM TODS 2(2), June 1977.

[Søl90] Sølvberg A., Kung C. *Information Systems Engineering*, Draft version, University of Trondheim, Norway, January 1990.

[The90] Theodoulidis, C., Wangler, B. and Loucopoulos, P. *Requirements Specification in TEMPORA*, 2nd Nordic Conference on Advanced Information Systems Engineering (CAiSE90), Kista, Sweden, May 1990.

[vanA88] van Assche, F., Loucopoulos, P., Speltincx, G., Venken, R. *Development of Information Systems: A Rule Based Approach*, Proceedings IFIP TC2/TC8 Working Conference on 'The Role of AI in Databases and Information Systems', Canton, China, July, 1988, North Holland.

[Ver82] Verheijen G. and Van Bekkum, J. *NIAM: an information analysis method , in ISDM, a comparative review*, North Holland (1982), IFIP.