

# Using DEMO and ORM in Concert—a Case Study

Jan L.G. Dietz  
Delft University of Technology  
P.O. Box 5031, NL-2600 GA Delft  
j.l.g.dietz@its.tudelft.nl

Terry Halpin  
Northface University  
Salt Lake City, Utah, USA.  
terry.halpin@northface.edu

The Demo Engineering Methodology for Organizations (DEMO) enables business processes of organizations to be modeled at a conceptual level, independent of how the processes are implemented. DEMO focuses on the communication acts that take place between human actors in the organization. The Object-Role Modeling (ORM) approach enables business information to be modeled conceptually, in terms of fact types as well as the business rules that constrain how the fact types may be populated for any given state of the information system and how derived facts may be inferred from other facts. ORM also includes procedures to map conceptual data models to physical database schemas. Both DEMO and ORM treat fact types as fundamental, and require that their models be expressible in natural language sentences. This suggests that the approaches may be synthesized in a natural way, resulting in a more powerful method for business modeling. This paper discusses an exploratory case study in which both methods were used in concert, and identifies some lessons learned.

## 1 Introduction

*Demo Engineering Methodology for Organizations* (DEMO) is a method for organization engineering, an emerging discipline concerning the design and implementation of organizations (Dietz, 1994; Dietz, 1999; Dietz, 2003a; Dietz, 2003b; Van Reijswoud, Mulder, and Dietz, 1999). Traditional organization science is based on a teleological system definition, which is concerned with the function and the behavior of a system in its environment. The corresponding dominant paradigm for studying organizations is the IPO-paradigm (Input-Process-Output). The matching model type is the black-box-model. *Organization engineering* is based on an *ontological* system definition, which is concerned with the construction and operation of a system. Its dominant paradigm for studying organizations is the *PSI-paradigm* (Performance in Social Interaction). The matching model type is the *white-box-model*.

Organization science and organization engineering are complementary fields. The former is particularly useful for *managing* organizations (strategic, tactic and operational management), while the latter is especially useful for *changing* organizations (redesign/reengineering of business processes, forming networks of organizations etc.). The PSI-paradigm states that an organization consists of people who, while communicating, enter into and comply with commitments (social interaction) about the things they bring about in reality (performance). This reality therefore is to a large extent an inter-subjective reality. Put differently, in their social interaction people engage in obligations about actions to take, and reach agreement about the results of those actions. The PSI-paradigm is made more specific and operational in DEMO as described later. DEMO belongs to a group of modeling approaches that are all based on the Language/Action Perspective (e.g. Goldkuhl, 1996; \\* MERGEFORMAT Medina-Mora, Winograd, Flores, and Flores, 1992 \\* MERGEFORMAT ). Van Reijswoud and Dietz (1999) provide a detailed description of DEMO \\* MERGEFORMAT .

*Object-Role Modeling* (ORM) is a fact-oriented approach for modeling information at a conceptual level. An overview of ORM is given in (Halpin, 1998a) \\* MERGEFORMAT , and a detailed treatment in (Halpin, 2001a) \\* MERGEFORMAT . ORM includes a family of closely related variants, including Natural Information Analysis Method (NIAM) (Wintraecken, 1990) \\* MERGEFORMAT , Natural Object Relationship Method (NORM) (De Troyer and Meersman, 1995) \\* MERGEFORMAT . Predicate Set Model (PSM) (ter Hofstede, Proper, and Weide, 1993) \\* MERGEFORMAT , and Fully Communication Oriented Information Modeling (FCO-IM) (Bakema, Zwart, and van der Lek, 1994) \\* MERGEFORMAT . Unlike Entity-Relationship (ER) modeling (Chen, 1976) \\* MERGEFORMAT and the class diagram technique of the Unified Modeling Language (UML) (OMG UML RTF, 2003) \\*

MERGEFORMAT, ORM makes no use of attributes as a base construct, instead expressing all fact types as relationships. This attribute-free approach leads to greater semantic stability in conceptual models and conceptual queries (Bloesch and Halpin, 1997; Halpin, 1998b) and enables ORM fact structures to be directly verbalized and populated using natural language sentences.

ORM supports mixfix predicates of any arity (unary, binary, ternary etc.), so its constraints and derivation rules can also be directly verbalized in sentential form. For details on business fact and rule verbalization in ORM, see the series of articles initiated by (Halpin, 2003). Moreover, ORM's graphic constraint notation is far more expressive than that of UML class diagrams or industrial ER versions. ORM is now supported by a number of modeling tools, which can automatically transform ORM schemas into physical database schemas (e.g. see Halpin, Evans, Hallock, and MacLean, 2003). For such reasons, ORM is being increasingly used for conceptual analysis of information, as well as ontology specification (Spyns, Meersman, and Jarrar, 2002) \\* MERGEFORMAT, and is currently being considered as a candidate for a standard business rule modeling language within the Object Management Group.

Both DEMO and ORM treat fact types as fundamental, and require that their models be expressible in natural language sentences. This suggests that the approaches may be synthesized in a natural way, resulting in a more powerful method for business modeling. This paper discusses the first attempts to explore the feasibility of this synthesis, and identifies some lessons learned, using a running example of a library application to illustrate the main ideas.

Section 2 summarizes the essential concepts and model types underlying the DEMO approach, and discusses how the library application is modeled using DEMO. Section 3 explains the main concepts and notations of ORM, and shows how the library application may be modeled in ORM. Section 4 identifies some ways in which ORM supplements DEMO by providing additional constructs and techniques for modeling the information system. Section 5 explores possible benefits of DEMO for ORM, including integration of data and process views. Section 6 summarizes the main results, suggests topics for future research, and lists references for further reading.

## 2 DEMO

An organization consists of social individuals (people) or *subjects* that perform two kinds of acts. By performing *production acts*, the subjects fulfill the mission of the organization. A production act (P-act for short) may be material (e.g. a manufacturing or transportation act) or immaterial (e.g. approving an insurance claim, or electing someone president). By performing *coordination acts* (C-acts for short), subjects enter into and comply with commitments. In doing so, they initiate and coordinate the execution of production acts. To abstract from the particular subject that performs an action, and to concentrate on the organizational role of the subject in performing that action, the notion of *actor role* is introduced. An actor role is a particular, atomic 'amount' of authority, viz. the authority needed to perform precisely one kind of production act. Actor roles need not correspond exactly with organizational functions. An actor role may be fulfilled by many subjects, and a subject may concurrently play a number of actor roles. A subject in his/her fulfillment of an actor role is called an *actor*. Figure 1 summarizes the main aspects of this organizational view.

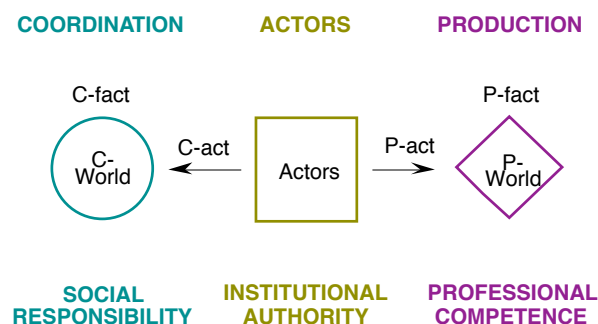


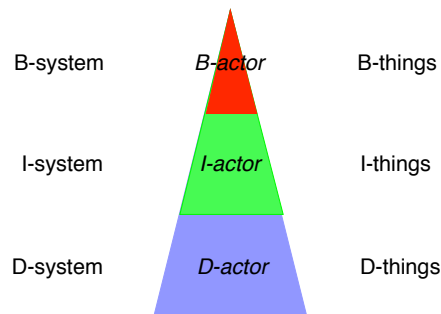
Figure 1 The white-box model of an organization

The result of successfully performing a P-act is a *production fact* or P-fact. P-facts in our library example include “membership M has started to exist” and “the late return fine for loan L is paid”. The variables M and L denote an instance of membership and loan respectively. All realization issues are fully abstracted out. Only the facts as such are relevant, not how they are achieved. Examples of C-acts are requesting and promising a P-fact (e.g. requesting to become member of the library).

The result of successfully performing a C-act is a *coordination fact* or C-fact (e.g. the being requested of the production fact “membership #387 has started to exist”). Again, all realization issues are ignored (e.g. whether the request is made by a letter or e-mail or via a website). Just as we distinguish between P-acts and C-acts, we also distinguish the two worlds in which these kinds of acts have effect: the *production world* or P-world and the *coordination world* or C-world respectively. Both the P-world and the C-world are at any moment in a particular state. A state is simply defined as a set of facts. So, a state of the P-world is a set of P-facts and a state of the C-world is a set of C-facts. State changes, also called *transitions*, take place instantaneously. The occurrence of a transition at a particular point in time is called an *event*. An example of an event is the creation of the P-fact “membership #387 has started to exist”). Events occur at discrete points in time, and the number of events in any finite time interval is finite.

P-acts and their related C-acts appear to occur in generic recurrent patterns, called *transactions*. A transaction has three phases: the order phase, the execution phase, and the result phase. It is carried out by two actors, who alternately perform acts. The actor who starts the transaction and eventually completes it, is called the *initiator*. The other, who actually performs the production act, is called the *executor*. The order phase is a conversation that starts with a request by the initiator and ends (if successfully) with a promise by the executor. The result phase is a conversation that starts with a statement by the executor and ends (if successfully) with an acceptance by the initiator. In between these two conversations there is the execution phase in which the executor performs the P-act. The process of a transaction can be more complicated but its complexity is always limited (Dietz, 2003b). Transactions are the molecules of business processes (Dietz, 2003a), the C-acts and P-acts being the atoms. A *business process* is defined as a (arbitrarily large) structure of causally linked transactions. A transaction T02 is causally linked to a transaction T01 if and only if T02 is initiated during the course of T01 by either the initiator or the executor of T01. Usually, T01 has to wait for the completion of T02 before proceeding.

Concerning production acts, and hence actors, three levels of abstraction are distinguished (see Figure 3). These levels may be understood as ‘glasses’ for viewing an organization. Looking through the *essential* glasses, one observes the core business actors, who perform production acts that result in original (non-derivable) facts, and who directly contribute to the organization’s function (e.g. approving a membership application, or diagnosing a patient’s medical problems). These essential acts and facts are collectively called *B-things* (from Business). Looking through the *informational* glasses, one observes intellectual actors, who execute informational acts like collecting, providing, recalling and computing knowledge about business acts and their results. Informational acts and facts are collectively called *I-things* (from Information and Intellect). Looking through the *documental* glasses, one observes documental actors, who execute documental acts like gathering, distributing, storing, copying, and destroying documents containing the aforementioned knowledge. Documental acts and facts are collectively called *D-things* (from Documents and Data).



**Figure 3** The three levels of abstraction.

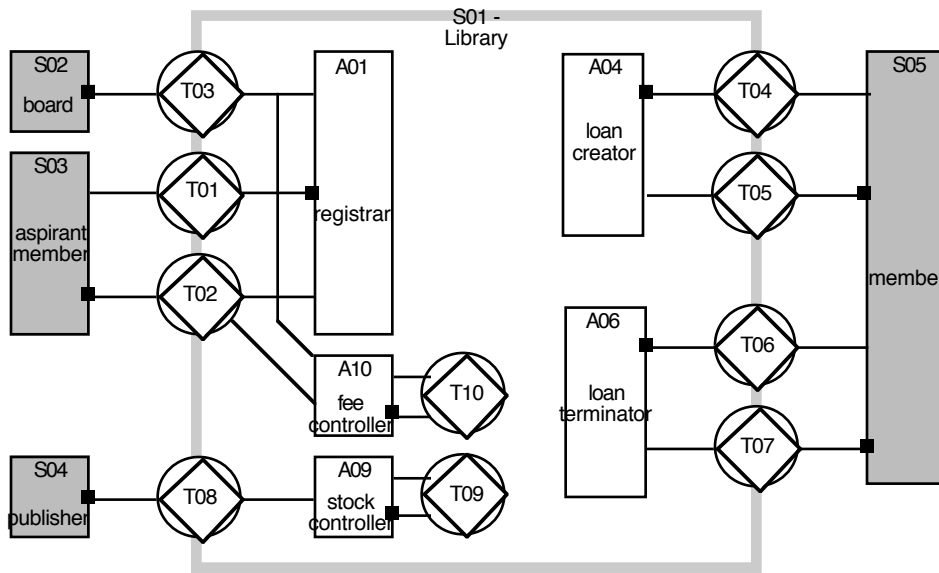
The three kinds of actors are called B-actors, I-actors and D-actors. They are elements of three corresponding aspect systems of an organization: the B-system, the I-system, and the D-system. The starting point and emphasis in DEMO is the B-system. Only in the B-system may new original facts be created, to contribute to fulfilling the organization's mission. The corresponding I-system and D-system are part of the realization of the B-system, and so can be designed only after the B-system is designed. Information and communication technology can be applied without any risk or harm to the I-system and the D-system. However, one must be cautious in applying it to the B-system, to prevent machines from taking over the responsibility of B-actors. One can only mimic or simulate B-systems. The triangular shape of the levels in

Figure 3 shows that there is nothing 'above' the B-system, and that generally the amount of D-things in an organization is much more than the amount of I-things, and that the amount of I-things is much more than the amount of B-things.

The complete model of the B-system of an organization in DEMO is called the *essential model* of the organization. It consists of an integrated set of four aspect models: the *Construction Model* (CM), the *Process Model* (PM), the *State Model* (SM), and the *Action Model* (AM). The CM shows the actor roles and the transaction types in which they play (as initiator and/or executor). The AM specifies the action rules that the actors apply in carrying out their transactions. Based on the AAM, the PM shows how the transaction types are causally and conditionally related, and the SM models the fact types that are created and/or used in carrying out the transactions. Only the CM and the SM are elaborated in this paper.

Figure 4 shows the CM of the library case. The diagram (an Actor Transaction Diagram) shows the actor roles, transaction types, and the relationships between them (i.e. which actor roles are initiator and/or executor of which transaction types). An actor role is represented by a box; the transaction symbol is a diamond (production) in a disk (coordination). The small black box denotes which actor role is the executor of a transaction type. The boundary of the considered part of the library is represented by the gray-lined open box. Actor roles inside the boundary are elementary actor roles—they execute exactly one transaction type. Actor roles outside the boundary are (by definition) non-elementary, so-called system actor roles; they are colored gray. Actually, what is inside the boundary is the 'uncovering' of the system actor role S01 (Library).

The table below the diagram (called a Transaction Result Table) lists all transaction types and specifies for each the resulting P-event type. Actor roles A09 and A10 are self-activating actors: they are both initiator and executor of the same transaction. This is how DEMO models periodic activities.

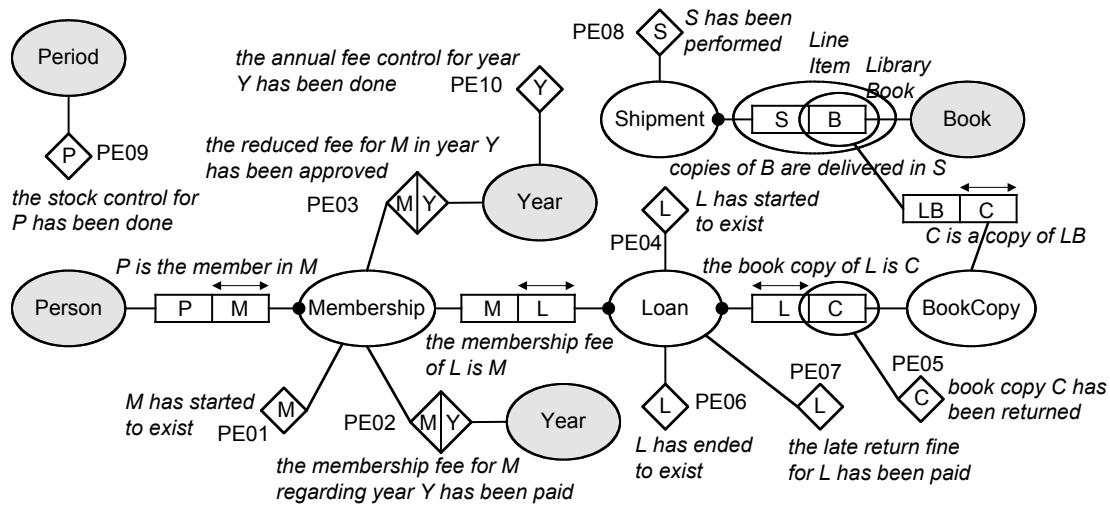


transaction type		resulting P-event type
T01	membership_registration	PE01 membership M has started to exist
T02	membership_fee_payment	PE02 the fee for membership M in year Y has been paid
T03	reduced_fee_approval	PE03 the reduced fee for membership M in year Y has been approved
T04	loan_start	PE04 loan L has started to exist
T05	book_return	PE05 book copy C has been returned
T06	loan_end	PE06 loan L has ended to exist
T07	return_fine_payment	PE07 the late return fine for loan L has been paid
T08	book_shipment	PE08 shipment S has been performed
T09	stock_control	PE09 the stock_control for period P has been done
T10	annual_fee_control	PE10 the annual_fee_control for year Y has been done

Figure 4 DEMO Construction Model (CM) of the library.

Figure 5 shows the SM corresponding to the CM of Figure 4. The diagram (an Object Fact Diagram), plus the table below it (an Object Property Table) may be viewed as a variant of the ORM model discussed in section 3. They specify all object types and fact types occurring in the action rules of the AM (of the B-system). The SM of (a part of) an organization is an *ontological* conceptual schema—it describes the types of things and facts (relationships) that can be observed, as well as the laws that appear to hold for the co-existence of these things and facts. The gray-colored boxes depict external object classes. They contain objects that play a role in the business processes, but their existence is determined by transactions other than those in the CM. The white-colored boxes depict internal object classes. The objects in these classes are created in the mentioned transactions. For the classes Membership, Loan, and Shipment, this is obvious. For BookCopy, these are the books delivered in shipments to the library.

The diamond shaped fact types are the production fact types that also appear in the Transaction Result Table of Figure 4. These fact types link the conceptual schema of the production world to the transactions that change the state of the production world. Consider the creation and termination of loans. There are two 'normal' fact types: "the membership of L is M" and "the book copy of L is C". A uniqueness constraint holds for the role of the loan in both fact types: a loan always relates to at most one membership and one book copy. A mandatory constraint also holds for Loan in both fact types. Hence a loan always relates to exactly one membership and one book copy. Therefore, the fact types "the membership of L is M" and "the book copy of L is C" are *existentially dependent* on Loan.



property type	object class	scale type + scale kind	int. / ext.
year_of_birth	PERSON	YEAR	I E
age (*)	PERSON	NUMBER	A -
author(s)	BOOK	AUTHORS	C E
year_of_publication	BOOK	YEAR	I E
category	LIBRARY BOOK	BOOK CATEGORY	C E
#copies_available (*)	LIBRARY BOOK	NUMBER	A -
#days_overdue (*)	LOAN	NUMBER	A -
incurred_fine (*)	LOAN	EURO	R -
minimal_age	YEAR	NUMBER	A E
standard_fee	YEAR	EURO	R E
reduced_fee	YEAR	EURO	R E
normal_loan_period	YEAR	NUMBER	A E
max_#copies_in_loan	YEAR	NUMBER	A E (=5)
daily_late_fine	YEAR	EURO	R E
control_increment	YEAR	NUMBER	A E
#copies_shipped	LINE_ITEM	NUMBER	A I
#books_in_loan (*)	MEMBERSHIP	NUMBER	A -

#days\_overdue (L) = < (start date of L) + (normal\_loan\_period) - (current date) >  
 #books\_in\_loan (M) = < the sum of book copies in loans of M that are not yet ended >  
 age (P) = current\_year - birth\_year (P) + current\_day\_of\_year div birth\_day\_of\_year (P)  
 incurred\_fine (L) = #days\_overdue (L) \* daily\_late\_fine (current\_year)

**Figure 5** DEMO State Model (SM) of the library.

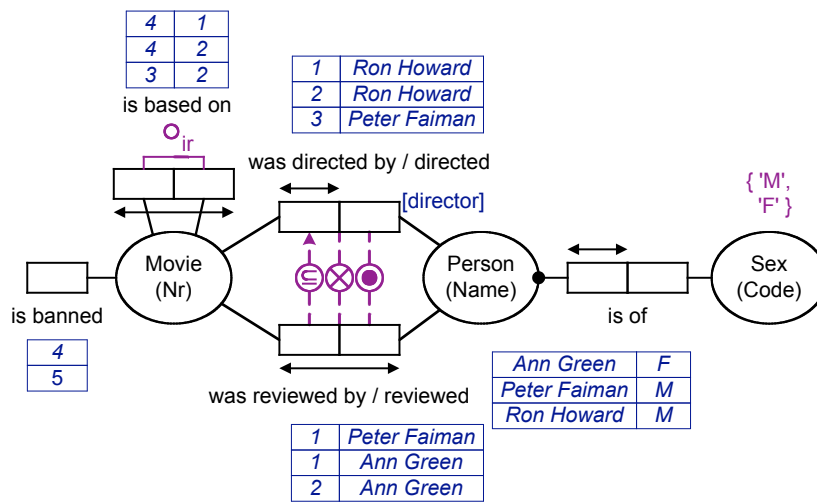
A new loan can be conceived of (and in a simulation game be generated) but that doesn't mean that it actually exists yet. In order to come into being, an event of type PE04 is needed. This event has a time stamp (the point in time at which it occurs). By definition this is the point in time at which the transaction T04 concerning L has successfully been completed (Dietz, 2003a). The loan ends its existence by an event of type PE06. During the lifetime of the loan, an event of type PE07 may occur (late return fine payment).

### 3 ORM

This section briefly explains the basic ORM graphical symbols, and then provides an ORM model for the library application. Object-Role Modeling is so-called because it views the universe of discourse (application domain) as a set of *objects* (non-lexical entities or lexical values) that play *roles* (parts in relationships). ORM stores all data in simple *fact types*, catering for unary, binary, and longer relationships, and allowing all fact structures to be easily populated with sample data to help validate business rules. Unlike ER and UML, ORM makes no use of attributes.

Graphically, object types are depicted as named ellipses (solid for entity types, and dotted for value types). As in logic, a *predicate* is a proposition with object-holes in it. In ORM, a predicate is treated as an ordered set of one or more roles, each of which is depicted as a box, which may optionally be named. A fact type is formed by applying a predicate to the object types that play its roles. Fact types in ORM must be given one or more readings. The *arity* of a predicate is its number of roles. For discussion purposes, each fact type may be populated by entries in a sample fact table that includes one column for each role of the fact type.

The ORM model in Figure 6 includes three object types (Movie, Person and Sex) and five fact types: Movie is banned; Movie is based on Movie; Movie was directed by Person; Movie was reviewed by Person; Person is of Sex. Inverse readings are supplied for two associations: Person directed Movie; Person reviewed Movie. One role is named (“director”). Simple identification schemes may be abbreviated in parentheses. For example, Movie(Nr) abbreviates the injective (1:1 into) association Movie has MovieNr. For simplicity, we assume that persons in this domain may be identified by name. In this example, all fact types are unary or binary. We could add Movie was released in Country in Year as a ternary fact type.



**Figure 6** An ORM model including an ORM schema and sample fact populations.

ORM classifies business rules into constraints and derivation rules. The ORM model in Figure 6 includes constraints but no derivations. The *value constraint*  $\{ 'M', 'F' \}$  indicates the possible sex codes. Arrow-tipped lines across one or more roles denote *uniqueness constraints*, indicating that instantiations of that role sequence must be unique. For example, the uniqueness constraint on the first role of Person is of Sex indicates that entries in the fact column for that role must be unique. The English version of ORM’s formal textual language verbalizes this constraint as: **each** Person is of **at most one** Sex.

A solid dot (possibly circled) connected to a set of one or more roles denotes a *mandatory constraint* over that role set. For example, the mandatory dot connected to the first role of Person is of Sex indicates that **each** Person is of **some** Sex. The mandatory dot connected to the other two roles played by Person depicts an *inclusive-or constraint*: **each** Person directed **some** Movie **or** reviewed **some** Movie (possibly both).

The  $\subseteq$  symbol connected to the roles of the fact type Movie is based on Movie denotes the irreflexive *ring constraint*: **no** Movie is based on **itself**. The circled subset symbol “ $\subseteq$ ” connected by an arrow from the first role of Movie was reviewed by Person to the first role of Movie was directed by Person denotes a *subset constraint*, indicating that the population of the first role must always be a subset of the population of the second role. In English: **each** Movie **that** was reviewed by **some** Person **also** was directed by **some** Person.

A subset constraint is one kind of set-comparison constraint. In general a set-comparison constraint applies across sequences of compatible role sequences (of one or more roles). Other varieties of set-comparison constraints are exclusion and equality constraints. For example, the circled “X” in Figure 6 denotes an *exclusion constraint* between the role-pairs that comprise the direction and review predicates. In English: **no** Movie was directed by **and** reviewed by **the same** Person.

We now turn to the library application. For convenience, we divide the ORM schema into four subject areas: membership, loan, book, and book shipment. Figure 7 shows the main aspects of the membership subschema. The reference mode “Id” for Person indicates that each person has a *value-based identifier*, called PersonId, used in human communication. Each person also has a name, not necessarily unique. A library year is a calendar year, at some time during which the library was in operation. The *reference mode* “CE” denotes “Common Era”, indicating calendar years are based on the Gregorian calendar.

The association Membership covers Year is *objectified* as the entity type AnnualMembership. Its association with FeeType indicates whether or not a given member has been granted a reduced membership fee for a given year. If desired, a derived fact type may be added to infer the fee paid for a given annual membership, based on the fee type and the membership fee of that type for the given year. For simplicity we assume that a member pays the full annual fee regardless of when he/she began or renewed the annual membership. In practice, it would be more usual to apply a pro-rata fee or extend the membership to a year after the date paid.

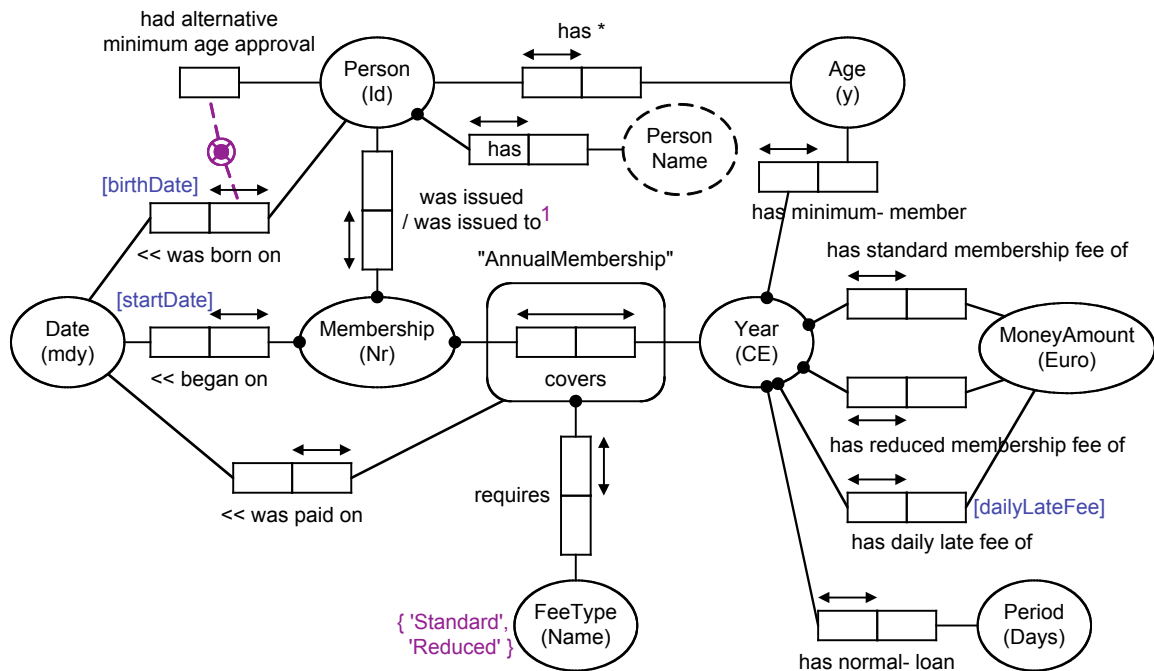


Figure 7 ORM subschema for library membership.

By default, predicates are read left to right and top to bottom. A reversed reading direction is indicated by a back arrow “<<”. The first role of the fact type Person was born on Date is optional. This means it is optional whether we record a person’s birth date (even though in the real world each person has a birth date). An ORM model reflects the *universe of discourse* (i.e. those aspects of the application world that we wish to discuss, and the rules that we wish to enforce), so the model need not agree in every respect with the real world. In this aspect, ORM differs from DEMO, where birthdate is mandatory simply because each person in the real world has a birthdate.

The life-buoy symbol (combination of inclusive-or and exclusion symbols) denotes an *exclusive-or constraint*: **each** Person was born on a Date **or** had alternative minimum age approval, **but not both**. Here the unary fact type caters for the case where a person does not supply his/her birthdate, (e.g. he/she may not wish to divulge it, or might not know it) but can have the minimum age requirement approved by authorized library staff (e.g. visual inspection of a person who is obviously old).

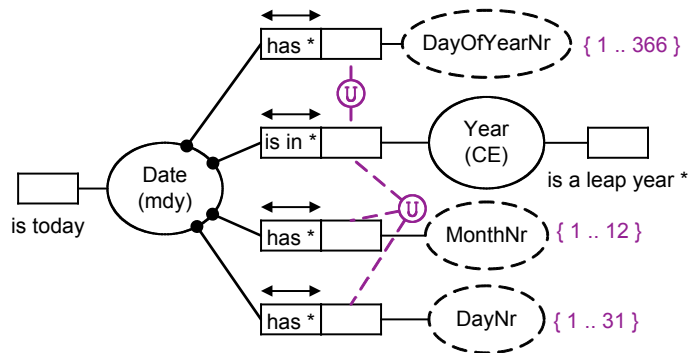
Notice the use of *hyphens* in the fact types Year has minimum- member Age and Year has normal- loan Period. This causes the hyphenated and any subsequent words before the following term for the object type to be bound to that term for verbalization purposes. For example, the uniqueness constraint on the first of these fact types verbalizes as “**each** Year has **at most one** minimum member Age” instead of “**each** Year has minimum member **at most one** Age”.

As discussed later, *role names* displayed in square brackets are used to provide function names for derivation rules that make use of attribute-style notation. The second role of predicates with the reading “has” are assumed to have name of the second object type, with the first letter in lower-case, unless an explicit role name overrides this. For example, the second role of Person has PersonName is named “personName”. For binary predicates with a reading comprised of “has” followed by a hyphenated phrase, the second role has a default name obtained by prepending the hyphenated phrase to the right-hand object type term. For example, the second role of the fact type Year has normal- loan Period is “normalLoanPeriod”.

The superscript “<sup>1</sup>” on the fact type Membership was issued to Person indicates the existence of a *textual constraint* on this fact type. The asterisk “\*” on the fact type Person has Age indicates that this fact type is *derived*. In a complete ORM model, all constraints that cannot be expressed in graphical notation as well as all *derivation rules* (to indicate how derived fact types are derived from other fact types) should be specified in a formal, textual language. For example, the derivation rule for Person has Age may be specified in attribute-style as shown below. Here, dayOfYearNr denotes the sequential position of the day in its year (e.g. 2003 September 14 has dayOfYearNr 257).

```
Person.age = today.year – Person.birthdate.year if today.dayOfYearNr >= Person.birthdate.dayOfYearNr
             else = today.year – Person.birthdate.year + 1
```

This formulation makes use of various operations (e.g. date subtraction) and functions (e.g. year) that are predefined for Date. Figure 8 summarizes some of the main underlying semantics from an ORM perspective. Each circled “u” depicts an *external uniqueness constraint*, indicating that each Year, DayOfYearNr combination and each Year, MonthNr, DayNr combination refers to only one Date. While the mdy (month-day-year) format for dates is used for communication purposes, internally dates may be implemented otherwise (e.g. as Julian dates). Fundamentally, ORM uses relational-style, over which an attribute-style may be defined. The nullary function “today” is defined as the result of the query !Date is today (using “!” to prepend each desired projection). The role names “dayOfYearNr”, “year”, “monthNr”, “dayNr” on the right-hand roles of the derived predicates may be used as function names in attribute-style rules.



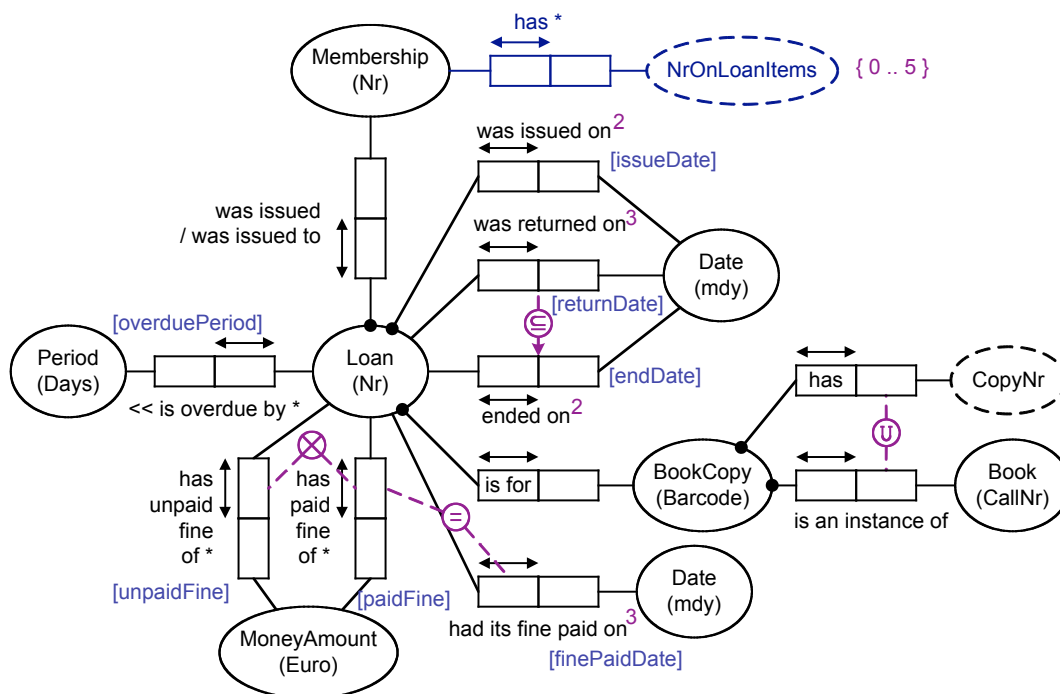
**Figure 8** Some predefined semantics underlying Date.

As a small extension to the current age rule shown earlier, a derivation rule may also be specified for the derived fact type Person on Date had Age. Using this fact type, the function “age of ... on ...” may now be specified over the parameter list (Person, Date). The textual constraint indicated by the subscript “1” in Figure 7 may now be specified as follows.

```
Membership.person.[age of Person on Membership.startDate] >= Membership.startDate.year.minimumMemberAge
```

If a person’s birthdate is not recorded, the age function returns null, and the whole expression evaluates to unknown. As in SQL, the constraint is violated if and only if it evaluates to false.

Figure 9 shows an ORM subschema for the main details about library loans. The circled “u” depicts an *external uniqueness constraint*, indicating that a particular copy (physical instance) of a book can be identified by combining the call number for the book with the copy number. As well as this composite identification scheme, a book copy also has a simple identification scheme (its barcode). The circled “=” depicts an *equality constraint* (a loan has a paid fine if and only if it had its fine paid on some date).



- <sup>2</sup> For each Loan, endDate >= issueDate
- <sup>3</sup> For each Loan, finePaidDate >= returnDate
- \* Membership.nrOnLoanItems = count each Loan that was issued to Membership and was returned on no Date
- \* Loan is overdue by Period iff Loan ended on no Date and Period = today - Loan.issueDate - Loan.issueDate.year.normalLoanPeriod and Period > 0
- \* Loan.unpaidFine = Loan.overduePeriod.nrUnits \* Loan.issueDate.year.dailyLateFee
- \* Loan.paidFine = (Loan.returnDate - Loan.issueDate - Loan.returnDate.year.normalLoanPeriod).nrUnits \* Loan.returnDate.year.dailyLateFee

**Figure 9** ORM subschema for library loans.

Each loan is for exactly one book copy. The subset constraint between the loan-return and loan-end associations declares that each loan that was returned on a date also ended on the same date. The superscripts “<sup>2</sup>” and “<sup>3</sup>” on fact types indicate that a textual constraint applies to them. In this case, the textual constraints are listed below the diagram. For each derived fact type (asterisked), a formal derivation rule declares how instances of the fact type may be derived from other facts. This example includes four derivation rules displayed below the diagram. ORM rules and queries (Bloesch and Halpin, 1997) may be formally specified in relational style and/or attribute-style (using role names and/or defined functions). The first derivation rule is expressed in relational style, the second rule in a combination of relational and attribute styles, and the last two rules in attribute style. The derivation rule for unpaid fines determines the fine currently accrued for an overdue loan—this amount may vary over time. The derivation rule for paid fines enables the system to compute the fine amount actually paid. The predefined nrUnits function converts a unit-based amount (e.g. 3 days) into a pure number (e.g. 3). This function may apply to any expression that returns a unit-based type, and enhances semantic stability by protecting rules against changes to choice of units.

Usually, constraints on derived fact types are themselves derivable. However, further constraints can be explicitly added to them (e.g. the value constraint on NrOnLoanItems). This provides a convenient and powerful way to declare various business rules that are awkward to express on base fact types.

Figure 10 shows basic subschemas for the book and shipment areas. If a book has an International Standard Book Number (ISBN), the library records this as well. These subschemas are straightforward, so should need no further explanation.

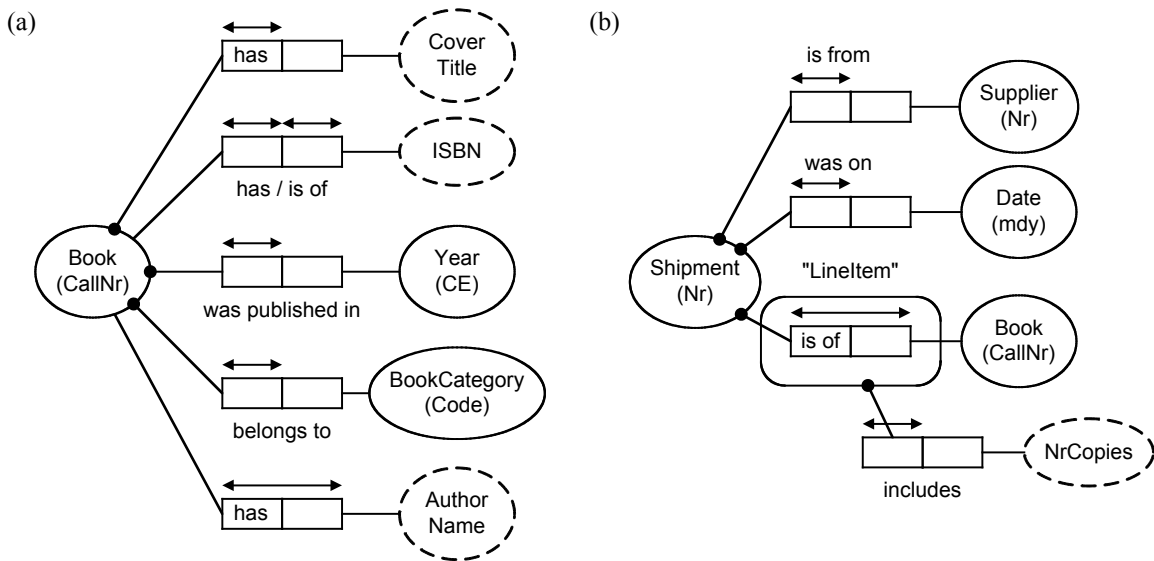


Figure 10 ORM subschemas for details about Books and Shipments.

#### 4 Possible benefits of ORM for DEMO

As explained in section 2, the DEMO approach uses a state model to declare the “essential” fact types and rules pertaining to the real world objects in the application domain. A state model is specified using an object-fact diagram supplemented by an object property table. Figure 5 shows the state model for the library application. Collectively, Figure 7, Figure 9 and Figure 10 provide an ORM model for the library application. A comparison between these two models reveals some important differences.

An ORM model is intended to capture all the fact types that are of interest in the application domain, as well as all static business rules (constraints and derivation rules that apply to each individual state of the information system) that need to be enforced. ORM models are also formal, so that they can be automatically transformed into implementation models. For these reasons, ORM models tend to be more complete and precise than corresponding DEMO state models.

The first major addition provided by ORM models is their *inclusion of at least one identification scheme for each entity type*. For example, in Figure 7 we see that each loan is identified by a loan number, and each book copy is identified by a barcode. In addition, we see that each book copy can be identified by combining the call number of its book with a copy number. Any reference scheme that is to be used in the application is considered relevant. Apart from being needed for the operation of the information system, such identification schemes enable the modeler to use real examples when populating fact types for validation purposes (as shown in Figure 6). This makes it much easier to decide whether the model accurately reflects the application domain. As DEMO considers the choice of any identification scheme as non-essential, this kind of information is ignored.

The second major difference is that ORM models typically *capture more constraints*. For example, the DEMO-SM ignores any dependency between the unary fact types PE05 (BookCopy has been returned) and PE04 (Loan has ended to exist) because this is captured in the OM (and consequently in the PM). To enforce the dependency, the ORM model includes a subset constraint between the loan-return and loan-end fact types to ensure that each returned loan is classified as ended. In general, ORM’s constraint language is more powerful (e.g. see Halpin, 2002b \\* MERGEFORMAT ).

A third addition provided by ORM models is that *all temporal aspects are declared explicitly*. For example, consider the DEMO unary fact type PE04: Loan has started to exist. Like any other DEMO fact type,

this has an implicit time stamp. In ORM, this is explicitly modeled using the fact type: Loan was issued on Date. This goes beyond the DEMO representation by including the *granularity* of the time stamp—in this case, day, rather than, for example, minute or second. This granularity choice is uncovered by inspection of sample requirements or by discussion with the domain expert. One of the design heuristics in ORM is to consider each fact type, and ask whether it needs to be treated in a snapshot or historical way. For example, consider the fact type: BookCopy has CopyNr. Although in the real world, instances of this fact type come into being at a given time (e.g. when assigned by the librarian), the recording of timestamp information for this fact type is not of interest to the users of the library application (as confirmed in interview sessions). Hence the ORM model excludes any temporal information about this fact type. In contrast, DEMO's ontological approach includes time stamps for all production events.

A fourth difference is that ORM provides *formal derivation rules* for relevant derived fact types. This makes it possible to automatically generate application code to enforce the rules. For example, consider the four derived properties listed at the bottom of the DEMO state model in Figure 5. Although precise, they are not expressed in a formal language, so are not executable. Although a derivation rule for computing a person's age is included in the ORM model, this applies only to those members who supply their birthdates. The library decided not to require all applicants to provide their birth date (this is left optional), allowing other ways to establish age (e.g. by visual inspection of the applicant). In contrast, the DEMO model assumes that birthdates are always known, and its derivation rule is based on this assumption.

Unlike a person's age, the determination of fines for overdue loans is always considered to be of interest to the system, as is the recording that such fines were paid. Unlike the DEMO model's single derivation rule for incurred fines, the ORM model includes two derivation rules, one to allow the computation at any instant for unpaid fines, and one to record fines that were actually paid (see Figure 9). The ORM model captures explicitly all decisions about what history to record in the information system.

In addition to enabling the formal capture of more information than DEMO state models, ORM provides *modeling procedures* and *formal transformation theorems* to assist modelers to create conceptual models and map them to implementation code. Details on ORM's conceptual schema design procedure and transformation theorems may be found in (Halpin, 2001a) \\* MERGEFORMAT . Of particular interest in this regard is ORM's use of data use cases (samples of required information) to seed the model. For example, concrete instances of data required from an as-is or to-be library system can be extremely helpful for specifying an initial model. But this practice requires the use of value-based identification schemes (at least tentative ones) for the entities involved, an aspect ignored by DEMO.

For the above reasons, ORM appears to provide a useful supplement to DEMO, offering ways to flesh out state models to complete, executable data models, and providing further procedures to help in the modeling process itself.

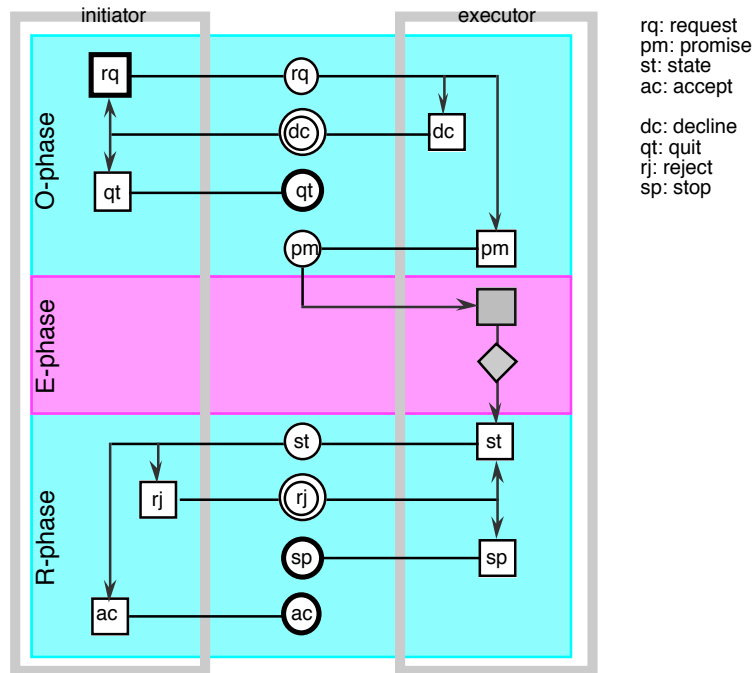
## 5 Possible benefits of DEMO for ORM

ORM is a method for information modeling, in particular for developing conceptual database schemas. Although ORM can be used to model manual and/or automated information systems, it is especially useful for specifying an executable schema for a fully automated information system (AIS). Because of its data-oriented focus, ORM covers only part of the scope of a business system (BS). This section investigates what DEMO can add to ORM in this respect.

The first addition provided by DEMO is the distinction between a BS and an AIS, which DEMO treats as an automated realization of the I-system discussed earlier (see Figure 3). This I-system supports the B-system, which represents the abstracted essence of the organization. The kinds of support are purely informational: collecting, providing, recalling and computing knowledge about business acts and their results. The AIS and the BS can each be modeled as a discrete dynamic system (or discrete event system) (Hee, van, Houben, and Dietz, 1989), but of a different category—a BS is a social system, whereas an AIS is a rational system.

An AIS is a software system, so the only support it can offer is to provide information to the BS that is modeled in the AIS. Only in the BS may original facts be created (which can then be entered in the AIS). For example, the replenishment orders generated by an automated stock control system are just (computed) output information as far as the AIS is concerned. At the I-system level they are not business orders. Only by virtue of the declaration by the B-system do these information items count as replenishment orders.

The second contribution offered by DEMO to the design of an IS, is a full account of the possible actions to be supported. The operating principle of a BS is the ability of human beings (in their role of social individual) to enter into and comply with commitments and agreements. This was called coordination in section 2. The standard pattern of C-acts and resulting C-facts of a transaction is shown in Figure 11. An open or white box represents a C-act type and an open or white disk represents a C-fact type. A gray box represents a P-act type and a gray diamond a P-fact type.



**Figure 11** The standard pattern of a transaction.

The initial C-act is drawn with a bold line, as is every terminal C-fact. The gray colored frames denoted by "initiator" and "executor" represent the responsibilities of the two partaking actor roles. The steps in the transaction process are ideal candidates for the functions (use cases) of the AIS. These are the atomic components of business processes, there is nothing more to support. Using ORM, one (only) has to decide which actions will be supported and how, and which will not.

As a third contribution to ORM, DEMO distinguishes between the dynamics of the BS and AIS. Every C-fact may serve as an agendum (singular of agenda) to be dealt with by an actor. Typically, an actor disposes of a set of agendums, or agenda. In dealing with an agendum, one or more new agendums may be generated. This constitutes the dynamics of a BS. The dynamics of the AIS is basically *asynchronous* with respect to the dynamics of the BS. They coincide only when products of the AIS are declared to count as acts in the BS (like we have seen for an automated stock control system). In all other cases, the C-acts must be made known to the AIS.

Consider for example the borrowing of a library book. This transaction of type T04 starts with a member request. The resulting C-fact "requested" is entered in the AIS. At the same time, a new instance of Loan is created by the AIS, including the facts that existentially depend on it ("the membership of L is M" and "the book copy of L is C"). Ideally, the recorded time stamp of the C-fact is the real time (valid time) at which it was created in the BS. It is however common practice to take the time of entering into the AIS (transaction time) as the time stamp. This usually causes no problems since the order in which the steps of Figure 11 are entered in the AIS is easily controlled by the AIS. For example, a promise fact is rejected by the AIS if there is no corresponding request fact. If the transaction succeeds, the terminal state is the C-fact "ac" (accepted). At the time of establishing this fact, the production fact becomes existent. From that time on, the loan really exists in the BS. This definition is easily implemented in the AIS: as soon as the accepted fact is entered, the loan exists (there is only the unavoidable time delay with the BS).

## 6 Conclusion

This paper outlined the essential features of the DEMO and ORM approaches to conceptual modeling, then explored various potential benefits of synthesizing both methods to achieve a more complete and productive approach to business and information system modeling. As both methods treat fact types as fundamental, it seemed judicious to their fact models as a basis for integration. With this in mind, a basic library application was modeled in both DEMO and ORM, and then commonalities and differences between these models were examined.

As regards the benefits of supplementing DEMO with ORM, it seems clear that ORM offers several advantages for fleshing out DEMO state models into more comprehensive, formal data models that can be automatically transformed into application code. In particular, ORM models can extend DEMO state models by providing identification schemes, additional constraints, explicit and granular coverage of relevant temporal aspects, and formal derivation rules, as well as focusing on those features of actual interest to the automated information system. In addition, various ORM modeling procedures may provide additional assistance in the task of constructing models.

On the other hand, using DEMO in conjunction with ORM provides a more comprehensive modeling approach that goes beyond ORM's data-oriented perspective. In particular, DEMO provides a clean integration of static and dynamic aspects of business modeling, offering high level, implementation-independent ways of modeling the essential business processes in terms of the communication acts being performed by the business actors. Because communication acts may modeled in terms of propositions (facts) and associated illocutionary forces, a clean integration with ORM's fact-based approach becomes feasible.

While our initial findings indicate positive benefits for synthesizing the DEMO and ORM approaches, a number of research problems require further analysis. In particular, the role of identification schemes in modeling needs further study. ORM mandates the use of such reference schemes early in the modeling process, while DEMO deliberately avoids them. The pragmatic consequences of this difference needs closer examination, as does the decision process involved in specifying automation boundaries to scope those aspects of the business that are to be implemented in an automated information system.

## References

- Bakema, G., Zwart, J. & van der Lek, H. (1994). Fully Communication Oriented NIAM. In: G. M. Nijssen & J. Sharp (Eds.), *NIAM-ISDM 1994 Conf. Working Papers* (pp. L1-35). Albuquerque, NM..
- Bloesch, A. C. & Halpin, T. A. (1997). Conceptual Queries using ConQuer-II. *Proc. 16th International Conference on Conceptual Modeling ER'97* (pp. 113-126). Los Angeles: Springer LNCS 1331.
- Chen, P. P. (1976). The entity-relationship model—towards a unified view of data. *ACM Transactions on Database Systems*, 1(1), 9-36.
- De Troyer, O. & Meersman, R. (1995). A Logic Framework for a Semantics of Object Oriented Data Modeling. *OOER'95, Proc. 14<sup>th</sup> International ER Conference* (pp. 238-249). Gold Coast, Australia: Springer LNCS 1021..
- Dietz, J. L. G. (1994). Modeling Business Processes for the Purpose of Redesign. *Proc. IFIP TC8 Open Conference on BPR*. Amsterdam: North-Holland.
- Dietz, J. L. G. (1999). Understanding and Modeling Business Processes with DEMO. *Proc. 18th International Conference on Conceptual Modeling ER'99*. Paris: Springer LNCS.
- Dietz, J. L. G (2003a). The Atoms, Molecules and Fibers of Organizations. *Data & Knowledge Engineering*.
- Dietz, J. L. G. (2003b). Generic recurrent patterns in business processes. In: Aalst, W. van der, Hofstede, A. ter, & Weske, M. (Eds.), *Business Process Management*, LNCS 2678. Springer-Verlag.
- Goldkuhl, G. (1996). Generic business frameworks and action modelling. In Dignum, F., J. Dietz, E. Verharen, & H. Weigand (Eds.), *Communication Modeling - The Language/Action Perspective, Proc. First International Workshop on Communication Modeling*, Electronic Workshops in Computing Springer. [Online] Available: <http://www.springer.co.uk/ewic/workshops/CM96/>.
- Halpin, T. A. (1998a). ORM/NIAM Object-Role Modeling. In: P. Bernus, K. Mertins, & G. Schmidt (Eds.), *Handbook on Information Systems Architectures* (pp. 81-101). Berlin: Springer-Verlag. .

- Halpin, T. A. (1998b). Conceptual Queries. *Database Newsletter*, 26(2). Boston: Database Research Group, Inc.
- Halpin, T. A. (2001a). *Information Modeling and Relational Databases*. San Francisco: Morgan Kaufmann.
- Halpin, T. A. (2001b). Supplementing UML with concepts from ORM. In: K. Siau & T. A. Halpin (Eds.), *Unified Modeling Language: Systems Analysis, Design, and Development Issues*. Hershey: Idea Publishing.
- Halpin, T. A. (2002a). Metaschemas for ER, ORM and UML: A Comparison, *Journal of Database Management*, 4-13. Hershey: Idea Group Publishing.
- Halpin, T. A. (2002b). Join Constraints. In: T. Halpin, J. Krogstie, & K. Siau (Eds.), *Proc. Seventh CAiSE/IFIP-WG8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design* (pp. 121-131). Toronto, Canada.
- Halpin, T. A. (2003). Verbalizing Business Rules: Part 1. *Business Rules Journal*, 4(4). [Online]. Available: <http://www.BRCommunity.com/a2003/b138.html>.
- Halpin, T. A. Evans, K., Hallock, P., & MacLean, B. (2003). *Database Modeling with Microsoft Visio for Enterprise Architects*. San Francisco: Morgan Kaufmann.
- Hee, K. M. van, Houben, G-J., & Dietz, J. L. G. (1989). Modelling of discrete dynamic systems; framework and examples. *Information Systems*, vol 14.
- ter Hofstede, A. H. M., Proper, H. A. & Weide, th. P. van der (1993). Formal definition of a conceptual language for the description and manipulation of information models. *Information Systems*, 18(7), 489-523.
- Medina-Mora, R., Winograd, T., Flores, R. & Flores, F. (1992). The Action Workflow Approach to Workflow Management Technology. In: J. Turner, R. Kraut (Eds.), *Proc. 4th Conference on Computer Supported Cooperative Work*. New York: ACM Press.
- OMG UML RTF (2003). *Unified Modeling Language (UML), Version 2.0*. [Online]. Available: [www.omg.org/uml](http://www.omg.org/uml).
- Spyns, P., Meersman, R., & Jarrar, M. (2002). Data modeling versus Ontology engineering. *ACM SIGMOD Record*, 31(4), 12-17.
- Van Reijswoud, V.E, & Dietz, J. L. G. (1999). *The DEMO Modeling Handbook*. [Online]. Available: <http://www.demo.nl>.
- Van Reijswoud, V. E., Mulder, J. B. F. & Dietz, J. L. G. (1999). Speech Act Based Business Process and Information Modeling with DEMO. *Information Systems Journal*.
- Wintraecken, J. (1990). *The NIAM Information Analysis Method: Theory and Practice*, Deventer, Netherlands: Kluwer.

## APPENDIX: Basic Description of the Library Case

The library described hereafter is one of the branches of the public library of Delftown. In the building in which it is located, is a desk for lending books (the out-desk) and a desk for returning books (the in-desk). The in-desk is occupied by Louise and the out-desk by Tim and Kevin on turn. There is a third desk, called the information desk, which is occupied by Lisa. The books that may be borrowed are put on shelves, sorted on the category of the title. Every (copy of a) book is identified by a bar code.

At the information desk one can get information such as opening hours, loan rules, and membership fees, and of course about the books. There is a binder on Lisa's desk which contains the complete library catalog, sorted in several ways (on author, on category and on title). One can freely browse through the binder to find the book one is looking for. Next to that, one can ask Lisa about the books in the catalog.

The information desk also serves as the registration desk. Anyone who wants to be registered as member of the library has to apply with Lisa. She writes the data needed on a registration form. These forms are collected daily by someone from the central office. Within a few days, the new member receives a letter welcoming him/her as new member and informing him/her about the library rules. The letter also contains the fee to be paid, and the message that the membership card can be collected at the branch office. By default, this fee is the standard annual fee as determined by the library board. Exceptions may be made for people without means. In that case, Lisa applies in writing to the library board for the reduced fee. Of course, she has to wait for the board's decision, which she also gets in writing, before the membership can

be registered. One gets the membership card after cash payment of the fee. The membership card has a bar code on it representing the membership number.

If one wants to borrow a book, one has to take (a copy of) the book from the shelves and take it to the out-desk. Tim or Kevin will then scan the bar code on the membership card, as well as the bar code on the book. These data are automatically entered into the library information system (LIS). The book is now considered to be lent to the member. No more than 5 books may be lent simultaneously to the same member.

When one returns a book, one goes to the in-desk and hands the book to Louise. She scans the book code, which is automatically entered into LIS. On the screen of her computer, she sees whether the loan period is exceeded or not. If it is, she also sees the fine that has to be paid. The person who returns the book has to pay the fine right away and in cash. After payment, Louise marks the book in her computer as returned. If the loan period is not exceeded, she only enters that the book has been returned. Returned books are piled on a table next to Louise. About every hour Lisa collects the pile and puts the books back on the shelves

Every month, the librarian (Maria) decides which titles should be added and how many copies per title have to be ordered. She does so on the basis of the announcements of new books she knows of (by means of flyers of publishers but also by surfing on the web).

At the start of a new calendar year, Lisa sends out invoices to all current members for the annual membership fee. Fees have to be paid in cash at the branch office. If applicable, she also sends renewal requests for the reduced fee to the library boards.