GENERALIZATION/SPECIALIZATION AND ROLE IN OBJECT ORIENTED CONCEPTUAL MODELING

To appaer in a forthcoming issue of Data and Knowledge Engineering

Monique Snoeck
Guido Dedene

Katholieke Universiteit Leuven
Dept. of Applied Economic Sciences
Naamsestraat 69, 3000 Leuven, Belgium
Tel. 32 16 32.68.83
Fax. 32 16 32.67.32
Email: {Monique.Snoeck, Guido.Dedene@econ.kuleuven.ac.be

ABSTRACT

The "IS A"-relationship and the mechanism of inheritance are powerful concepts that help to reduce complexity of models and redundancy in specifications. However, in the area of conceptual modeling, it seems that current Object Oriented Analysis methods put most emphasis on the structural aspects of the "IS A"-relationship while inheritance and sharing of behaviour are often not or ill-defined.

This paper investigates how attribute sharing, behaviour sharing and subset hierarchies can be combined into a sound "IS A"-modelling concept that guarantees universal substitutability. Decision criteria on the use of generalization/specialization are discussed and a formal taxonomy of processes corresponding to the generalization/specialization hierarchy is presented.

Keywords: Generalization/Specialization, Inheritance, Roles, Object oriented modeling methods, process algebra, formal specifications.

1. INTRODUCTION

The "IS A"-relationship and the mechanism of inheritance are powerful concepts that help to reduce the complexity of models and the redundancy in specifications. Moreover they allow for increased reuse of specifications and code. However, as pointed out by various authors (e.g. [1, 14]), the reasons why and the situations in which modellers and programmers use these concepts are often questionable. In order to improve the use of generalization/specialization and inheritance a sound "IS A"-modelling relationship should be defined. This relationship must also be easy to map into OOPL inheritance.

Generalization/specialization is an abstraction principle that allows to define classes as a refinement of other classes. The more general class is also called supertype or parent class and the refined class is then called a

subtype or child class. A complete Object Oriented Analysis (OOA) method should cover both static and dynamic aspects of objects as well as the way objects interact with each other with equal emphasis. These aspects are generally supported by different techniques. The generalization/specialization hierarchy can be considered as part of the static model while behaviour descriptions are part of the dynamic model. In a good OOA-method, the relationship between static, dynamic and interaction schemas should be made explicit and checked for consistency. However, the question of how the behaviours of generalization and specialization types relate to each other is rarely addressed by OOA-methods. If we assume that the technique of Finite State Machines is used for behaviour modelling (as is done in [6, 8, 19, 21, 22] and many other OOA-methods), then these are examples of relevant questions:

- Does a specialization inherit the statemachine of its parent ?
- Can it refine this statemachine by adding, removing or redefining states, transitions or events ?
- Can it restrict the behaviour of its parent or extend it or both ?
- Are the events of the specialization specializations of the events of the parent ?
- Can a specialization override properties of its parent ?

Many methods do not answer these questions in a very precise or formal way. For example, in OOSA [21, 22] the life-cycle of a subtype corresponds to a part of the life-cycle of its supertype. This definition violates the broadly accepted notion of inheritance where subtypes inherit data *and behaviour* of their supertype.

When using structured techniques for behaviour specifications, modelling difficulties appear often as structure clashes amongst events [13]. This kind of clashes occurs when the set of relevant events for an object type can be split in several subsets such that there are no sequence restrictions between events of different subsets. In [13] an example is given of Ruritanian soldiers pursuing simultaneously a training career and a promotion career, which are completely independent of each other. Structure clashes can easily be solved by the use of role types. Role types are very close to specialization types in this sense that they model partial temporal behaviour aspects of object types.

The role concept is also described by Pernici [18] as a way to model temporal aspects of objects. Using roles to describe temporal properties of objects is completely compatible with the role concept as used in JSD [13]. It is also compatible with the notion of roles in OOD where a role denotes the selection of a set of behaviours that are well-defined at one point in time [3, p.518]. The role concepts as used in OSA [8], OMT [19] and OOSA [22] is totally different from the role concept of Pernici [18]. In OSA [8] it is in fact the concept of existence defined subclass; in OMT roles are defined as "one end of an association" [19, p.34] and in OOSA roles are one category of "things" in the problem domain that can possibly be identified as object types ([22], p.13).

Even when -at first sight- the use of the concepts are compatible, the definitions of the generalization/specialization hierarchy and the concepts of roles are often very different or even contradictory. Sometimes the generalization/specialization and role hierarchy is defined as a superset/subset hierarchy [8], but most of the time the generalization and specialization type have disjoint instance sets. Some authors (e.g. [9]) argue that a specialisation has a permanent character as opposite to roles, but other authors allow objects to move between classes in a generalization/specialization hierarchy (e.g. [10]). Specializations are always allowed to add features to those inherited from the parent class. Some methods allow to override inherited features [19], other methods not [6].

It is clear that the concepts of Generalization/Specialization and Role would benefit from precise definitions. This paper formalizes the notion of the "IS A"-hierarchy and role at the conceptual level. In general most OOA-methods use the "IS A" concept without formally defining its semantics. As a result, specifications can be

ambiguous and interpreted differently by different analysts. Some authors have formalized the "IS A" notion (e.g. [24]), but to our knowledge none of them has explicitly dealt with the notion of inheritance of behaviour in their formalization. This is surprising because in the Object Oriented paradigm a class definition encapsulates data and behaviour and consequently behaviour is always inherited as well. In addition, an ill defined inheritance relationship at analysis level will probably cause problems with polymorphism and dynamic binding.

This paper uses Finite State Machines (FSMs) as technique for behaviour modeling, just as most OOA-methods. By formalizing the concept of Finite State Machine by means of Process Algebra, we are able to explicitly deal with the notion of inheritance of behaviour and formulate rules that ensure consistency between inheritance of static aspects and inheritance of dynamic aspects.

Before we give a formalization of inheritance, we first investigate how the "IS A"-relationship can be used in a consistent manner. In the following section it is argued that in the area of *conceptual modeling* the generalization/specialization and role concepts must be treated very carefully. A number of situations can be modelled with as well as without these concepts. More specifically, a number of examples will illustrate how in some cases the generalization/specialization or role construct can be replaced by more simple constructs while retaining the same semantic content. The next section will then propose a number of criteria that determine in which situations the generalization/specialization and role construct are to be used. Finally, the last section precisely defines the semantics of the role and generalization/specialization concepts in a mathematical way and presents a process taxonomy corresponding to the generalization/specialization hierarchy. From this taxonomy consistency requirements for the behaviour definitions of supertype and subtype are derived. In order to be able to refine the idea of inheritance of behaviour in a formal way, the basic definitions of the process algebra of M.E.R.O.DE. are used [7, 23].

2. On the use of Generalization/Specialization hierarchies

The following notations will be used to denote a generalization/specialization hierarchy:

Notation.

Let G, S^1, ..., S^n be object types from a conceptual schema.
If G is a supertype of S^1, ..., S^n this is written as:
$G < S^1, ..., S^n$
and graphically shown as

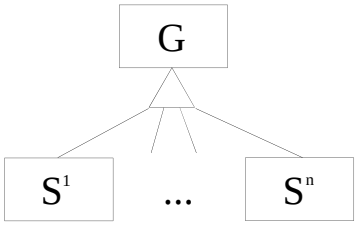

figure 1: graphical representation of a generalization/specialization hierarchy

Specializations are supposed to be disjoint (no object can be in two specialization classes at one point in time) and optional (the generalization class is not necessarily empty), unless otherwise specified.

For the ER-schema, the notations of [5, 23] are used. Weak relationship types are indicated by means of a double line on the side of the weak entity type and mandatory relationship types are indicated by means of a black dot on the side of the entity type that requires mandatory participation.

This section investigates the different ways in which a base class can be partitioned in subclasses and whether it is appropriate to identify these subclasses as new object types. Three types of subclasses are discussed: attribute defined, existence defined and state defined subclasses [10].

2.1. Attribute defined subclass

Example 1.

Let BOOK be an object type defined within the context of a library. This object type has an attribute named 'colour' with domain (red, blue, gray, ...). We can define subclasses of BOOK based on the value objects take for the colour attribute:

RED-BOOK = BOOK where BOOK.COLOUR = 'red'
BLUE-BOOK = BOOK where BOOK.COLOUR = 'blue'
...

Although it is hard to imagine that it might be useful to define the set of red books as a specialization type of BOOK, there are examples where attribute defined subclasses might seem useful.

Example 2.

In Belgium a different V.A.T. percentage has to be paid for cars depending on the price of the car. Cars of 1.500.000 BEF or more are considered to be of a de luxe model and are subject to the de luxe V.A.T. of 33%. Standard cars of 1.499.999 BEF or less are subject to a V.A.T. of 20.5%

The attribute price thus defines a total (the generalization class is always empty) and disjoint specialization and the event 'invoice' will have a different content for each specialization.

```
standard_model.invoice =
    Begin
    ...
    invoice_amount = price * 1.205;
    ...
    end;

de_luxe_model.invoice =
    Begin
    ...
    invoice_amount = price * 1.33;
    ...
    end;
```

The same effect can be obtained if a single class CAR is defined with the following method for invoice:

```
car.invoice =
    Begin
    ...
    if price < 1500000
    then invoice_amount is price * 1.205
    else invoice_amount = price * 1.33;
    ...
    end;
```

This example also shows that the concept of generalization/specialization is the dual of choice as suggested by [4].

2.2. Existence defined subclass

A subclass S of a base class G can be defined as consisting of all members of G that are involved in a relation with an other object. This can be done for any optional relationship type. Two examples illustrate this concept.

Example 3.

Every department has exactly one manager. Some employees are the manager of one department. This is illustrated in figure 2

figure 2: Some employees manage a department.

As the relationship type HEAD_OF is optional for employees, some employees will not participate in a relation of this type (see figure 3). The set of employees can thus be split in two disjoint subsets: employees that are a manager and employees that are not, as shown in figure 4. As a result, the relationship type HEAD_OF is now mandatory for MANAGER.

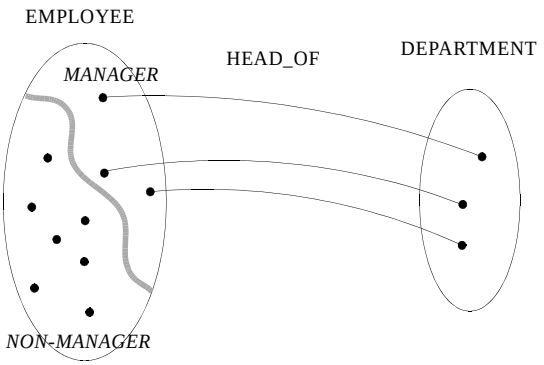

figure 3: occurrences of EMPLOYEE, DEPARTMENT and HEAD-OF

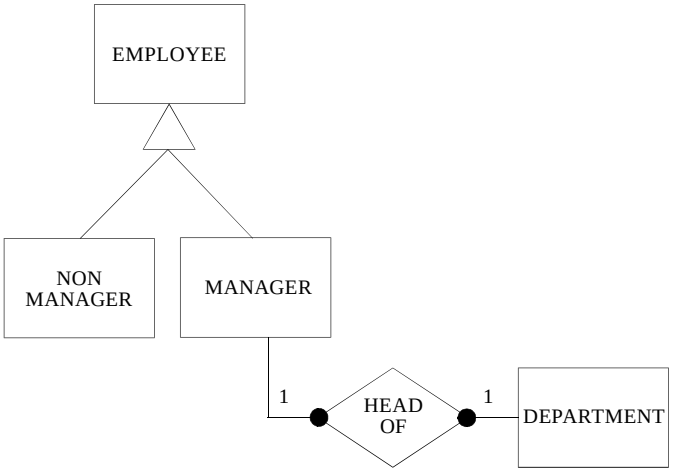

figure 4: managers and non-managers

It can be argued that the mere fact of participating in a relationship is not a sufficient reason to define a generalization/specialization hierarchy. Hence, in honour of the principle that a specification should be kept as simple as possible, the solution with the optional relationship type is to be preferred in our opinion. The more that both solutions have an identical semantic content. The constraint that only managers can head a department can be modelled by means of sequence restrictions, by requiring that the event 'assign_to_department' be preceded by an event 'promote_to_manager'. In addition, being a manager may in fact be a temporal property of employees: usually an employee is promoted to manager only after a few years and possibly a manager can be demoted to rank of employee.

The second example is taken from the CRIS-case [17].

Example 4.

The CRIS-case describes the organization of an IFIP Working Conference. This example shows part of the schema for this case-study. Figure 5 shows a schema without generalization/specialization and figure 6 shows a solution with generalization/specialization as proposed by [9].

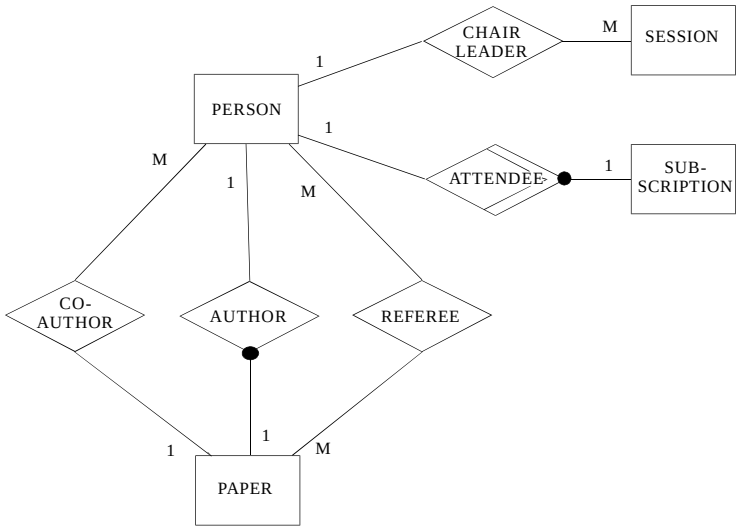

figure 5: ER-schema for the CRIS-case

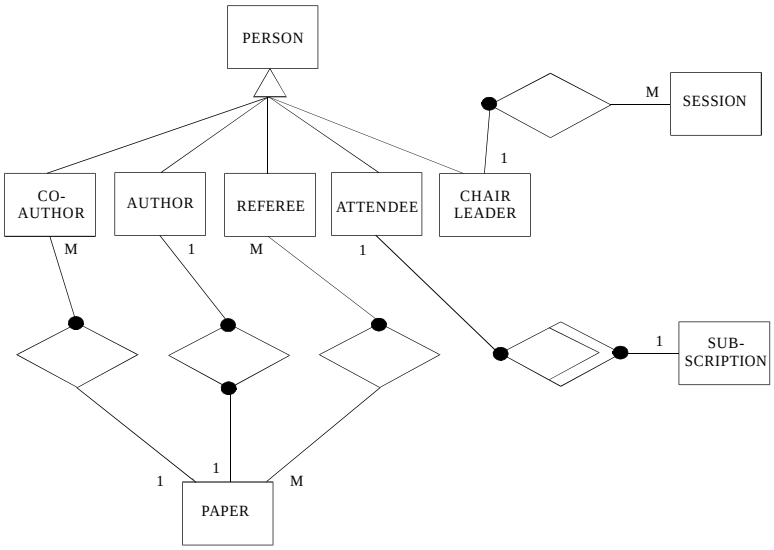

figure 6: Specializations for PERSON in the CRIS-case

Note that because of the use of generalization/specialization cardinality restrictions change from optional to mandatory.

In [9] ATTENDEE, REFEREE, AUTHOR, CO-AUTHOR, ... are defined as *roles* of PERSON, motivated by the fact that being an attendee, referee, author, ... and so on are temporal rather than permanent aspects of a person. In our opinion, the use of the role-concept is only needed if behaviour restrictions are to complicated to be modelled in a single lifecycle for person.

2.3. State defined subclass

This kind of subclass definitions is equivalent to the 'user controllable subclass' mentioned in [10]. In this case a subclass S is defined as all members of the base class G that are in a particular (set of) state(s). The first example is taken from [10].

Example 5.

The class of SHIPS can be divided into two disjoint subclasses BANNED_SHIPS and SAFE_SHIPS. Originally every ship is a member of SAFE_SHIPS. If it is banned from territorial waters it becomes a member of the class BANNED_SHIP. If authorities decide to rescind the ban, the ship is moved to the set of SAFE_SHIPS. The fact that ships are *moving* between subclasses indicates that the fact of being banned is a temporal property of a ship and can thus be modelled by means of behaviour descriptions. Figure 7 shows a finite state machine that describes the life cycle of a ship. The class of BANNED_SHIPS can then be defined as the set of ships which are in state 2, while SAFE_SHIPS are those that are in state 1. This same situation can also be modelled as an attribute defined subclass if the attribute 'banned' with domain ('yes','no') is added to the class definition of SHIPS.

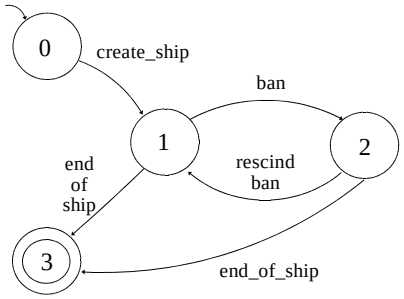

figure 7: Banning a ship and rescinding the ban.

The second example is taken from the CRIS-case [17].

Example 6.

Authors can respond to a call for papers by submitting a paper. The submitted papers are classified and then refereed by two persons. The outcome of the refereeing process is rejection or acceptation of the paper. Accepted papers will be published in the proceedings of the conference. This sequence of events is shown in the finite state machine in figure 8 that models the lifecycle of a paper. In [20] the classes SUBMITTED_PAPER and ACCEPTED_PAPER are defined. These classes correspond to the subclasses of PAPER consisting of all papers which state is in the set {1, 2,..., 9} or {7, 8} respectively.

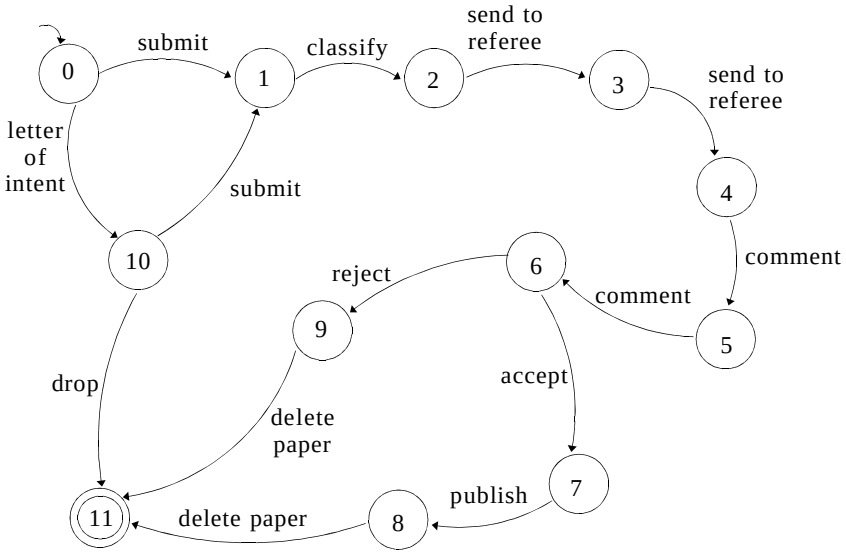

figure 8: lifecycle for a paper

In [9] SUBMITTED_PAPER is modelled as a subtype because being considered as a permanent aspect of a paper while ACCEPTED_PAPER is modelled as a role type because being accepted is considered to be a temporal aspect.

These examples show that defining subsets of classes can be done in various ways. As a result the same Universe of Discourse can be represented by different schemas. The following section presents a number of guide-lines that help the analyst to decide upon the concepts to use to model subsets.

3. GUIDE-LINES FOR USING THE ROLE CONCEPT AND THE GENERALIZATION/SPECIALIZATION CONCEPT

In order to obtain a complete set of criteria to decide upon the use of the generalization/specialization or role concept, we start from the components of an object type definition. The static aspects of object types contain attribute definitions. A subclass can have additional attributes and it can have more stringent or additional constraints on inherited attributes. The dynamic aspects of an object type are described by means of methods and sequence restrictions. Methods are the reaction of an object to certain events. A subclass can respond to more events than its parent, it can overwrite inherited methods and it can overwrite the sequence constraints on inherited methods. As general guide-lines we propose that attribute, existence or state subclasses are not modelled as specialization types or role types of a base type *unless* at least one of the following conditions is satisfied:

C1) Does subclass A have *additional attributes* compared to base class B ?
C2) Do objects from subclass A respond differently to events than objects from base class B ? That is, does subclass A have *other methods* than class B for the same events ?
C3) Does subclass A have *different attribute constraints* than class B ?
C4) Does subclass A respond to *more events* than class B ?
C5) Does subclass A have other *sequence constraints* than class B ?

In order to choose between the use of the role concept or the generalization/specialization concept, the following question must be answered:

C6) Do objects belong *permanently or temporary* to subclass A ?

Depending an the answers to the these six questions one of the following actions must be taken:

- model A as a specialisation type of B
- model A as a role type of B
- A is the same type as B

The three actions are exhaustive and exclusive in this sense that we cannot model a type A as being a specialisation type of B and a role type of B at the same time. The decision rules thus should lead to exactly one of the three conclusions.

Originally, the following decision rules were proposed:

rule 1: Roles are used to model temporal aspects while the generalization/specialization hierarchy is used to model permanent aspects.
rule 2: If A has other methods than B for the same event types, then A is a specialization type of B.
rule 3: If A has other attribute constraints than B, then A is a specialization type of B.
rule 4: If A only has additional event types and/or other sequence constraints, A can be modelled as a role type of B.

The rules have been investigated by means of PROLOGA [25]. This decision table workbench reveals unnecessary conditions, incomplete specifications and contradictions. The final decision table with guide-lines for the use of the role concept and the generalization/specialization concept is shown in figure 9. The conditions are listed in the upper left quadrant; the actions are listed in the lower left quadrant. The upper right quadrant shows the possible alternatives for the conditions of the corresponding row. For the five first conditions a 'Y'

stands for 'Yes', and a 'N' stands for 'No'. For the sixth conditions (What is the character of the subtype ?), the 'P' stands for 'Permanent' and the 'T' for 'Temporal'.

C1. other attributes ?	-						
C2. other methods	Y		N				
C3. other constraints	-		Y		N		
C4. additional events	-		-		Y	N	
C5. sequence constraints	-		-		-	Y	N
C6. character	P	T	P	T	-	-	-
1. A is spec type of B	x	-	x	-	-	-	-
2. A is role type of B	-	-	-	-	x	x	-
3. A = B	-	-	-	-	-	-	x
4. Contradiction	-	x	-	x	-	-	-
	1	2	3	4	5	6	7

figure 9: Guide-lines for using the role concept and the generalization/specialization concept.

In turns out that in case at least one of the four conditions C2 to C5 is satisfied, the question of additional attributes is irrelevant. In case none of the four conditions is satisfied (column 7) the mere fact that A has other attributes than B is not a sufficient condition to model B as a specialization or role type. Indeed, if A has additional attributes, we can expect that there are also additional of different methods that can handle these additional attributes. If no such methods are present, we assume that the additional attributes belong to the base class and can possibly be assigned a null value.

Columns 2 and 4 denote real contradictions: the temporal aspect of A leads to the use of the role concept while the presence of other methods and/or other attribute constraints leads to the use of the generalization/specialization concept. In these situations it is recommended that the contradiction be resolved by modifying the conceptual schema. This can for example be done by explicitly modeling the role as an extra class as shown in the following example.

Example 7.

K.U.Leuven Real Estate Management :

> *Each building consists of a number of rooms. Each room serves a specific purpose: a didactical, research or logistic purpose. When a building is rearranged, rooms can receive another destination.*

Suppose we have other attributes, other events, other methods and other attribute constraints for each kind of room, then we would like to model CLASS_ROOM, RESEARCH_ROOM and OFFICE as specializations of ROOM. But as the destination is a temporal aspect of rooms, we have to model these as role types of ROOM. A possible solution is to model the destination of a room as a separate object type as shown in figure 10.

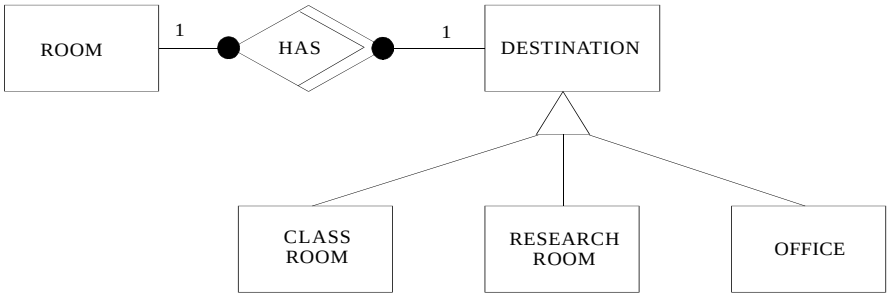

figure 10: Kinds of rooms

Note that a DESTINATION is existence dependent of ROOM with cardinality 1: a room can have only one destination at one point in time. But with this solution we have also the flexibility to allow rooms to have more than one destination at one point in time by changing the cardinality to Many.

4. A MATHEMATICAL DEFINITION FOR ROLES AND GENERALIZATION/SPECIALIZATION

The first paragraph defines the basic concepts of object type and event type. It then briefly defines the notions of object (occurrence) and event (occurrence). As conceptual schemas usually consist of more than one object type, it is necessary to describe how object types can be composed. First the basic operations of choice, sequence and iteration are defined. Next the concurrency operator on object types is described. The second and third paragraph formalize the concepts of role and generalization/specialization respectively. Finally, the fourth paragraph presents a process taxonomy that matches the generalization/specialization hierarchy and investigates what restrictions must be imposed on behaviour definitions in order to ensure consistency between the structural generalization/specialization hierarchy and the process schema.

All the definitions are part of the process algebra of M.E.R.O.DE. as presented in [7, 23]. Although in M.E.R.O.DE. communication between objects happens by means of common events, the definitions should be easy to extent to OOA-methods that use other communication mechanisms, provided that a formalism equivalent with regular expressions (e.g.. Finite State Machines or Harel Statecharts) is used for modeling the dynamic aspects. Object interaction by means of common event types can easily be extended to synchronous event triggering where the synchronizing events need not to have the same name. Indeed, an equivalence relation can be defined on the set of event types such that all event types in one equivalence class synchronize. It is then sufficient to replace event names by the name of the equivalence class to which the event belongs, to be in a situation of communication by means of common events.

4.1. Basic definitions

In M.E.R.O.DE. at the conceptual level, object types do not communicate with each other by means of message passing, but by means of common event types. For example assume an object type MEMBER, an object type BOOK and an event type *borrow*. In stead of specifying a message from BOOK to MEMBER (or inversely from

MEMBER to BOOK) that is triggered by a borrow event, both object types are provided with a method called 'borrow' and it is agreed that object types have to synchronize on common events. This way of communication is similar to communication as defined in CSP [11] and ACP [2]. Message passing is more similar to CCS [16].

If we assume a universe A of relevant event types in the Universe of Discourse, the set of services (methods) delivered by an object type is a subset of A. This subset is also called the alphabet of the object type. Object types are allowed to impose sequence restrictions on the event types in their alphabet by means of a regular expression, a Finite State Machine or a JSD diagram, which are all three mathematically equivalent formalisms. As a result, object types can be seen as tuples over $<\mathcal{P}(A), R^*(A)>$, where $R^*(A)$ is the set of all possible regular expressions over the universe of event types A [7, 23]. Formally:

Definition 1.

$R^*(A) = \{e \mid e$ is a regular expression over A$\}$ where e is a regular expression over A if and only if

(a) $e = 0$ or
(b) $e = 1$ or
(c) $\exists\, a \in A: e = a$ or
(d) $\exists\, e', e'' \in R^*(A)$ such that $e = e' + e''$ or $e = e'.e''$ or $e = (e')^*$

The symbols '1' and '0' stand for 'do-nothing' and 'deadlock' respectively.
In fact, iteration is a meta-syntactical operator as it can be defined by means of selection and sequence:

Definition 2.

$e^* = \sum_{i \in \mathbb{N}} e^i$ where $e^0 = 1$, $e^1 = e$ and $\forall\, n \in \mathbb{N}, n \geq 2: e^n = e.e^{n-1}$

In M.E.R.O.DE. the two basic operators '+' and '.' satisfy the following laws:

Let $e, e', e'' \in R(A)$, then

(1)	$e + 0 = e$	(6a)	$e.0 = 0$
(2)	$e + e = e$	(6b)	$0.e = 0$
(3)	$e + e' = e' + e$	(7)	$e.(e'.e'') = (e.e').e''$
(4)	$e + (e' + e'') = (e + e') + e''$	(8)	$e.(e' + e'') = e.e' + e.e''$
(5a)	$1.e = e$	(9)	$(e' + e'').e = e'.e + e''.e$
(5b)	$e.1 = e$		

A motivation and discussion of these laws can be found in [7, 23].

These definitions can now be used to define the concept of object type:

Definition 3.

Let $\alpha \subseteq A$, $e \in R^*(A)$, then $<\alpha,e>$ is called a tuple over $<\mathcal{P}(A), R^*(A)>$

An object type P is a tuple $<\alpha,e>$ over $<\mathcal{P}(A), R^*(A)>$ such that $e \neq 0$ and $\varphi(e) \subseteq \alpha$

where $\varphi : R^*(A)$ and $\varphi(0)$ is not defined

In addition it is required that α can be partitioned into three pairwise disjoint sets c(P), m(P) and d(P). c(P) and d(P) must not be empty.

Notation: if $P = <\alpha,e>$ then $S_A P = \alpha$, $S_R P = e$

A conceptual schema $\mathcal{M}$ is a set of object types such that $\bigcup_{P \in \mathcal{M}} S_A P = A$

It is worth to note that the alphabet of an object type P is dived into three disjoint sets of events c(P), m(P) and d(P) which denote the sets of creators, modifiers and destructors of P respectively. In addition it is required that there is at least one event type that creates occurrences of P and at least one event type that destroys occurrences of P.

Example 8.

Imagine a library where all available books can be searched for by means of an on-line catalogue. The definition of the object type BOOK could be as follows:

BOOK = <{acquire, classify, borrow, renew, return, lose, declassify, remove_book},
acquire.classify.(borrow.(renew)*.return)*. (borrow.(renew)*.lose + 1).declassify. remove_book},

This specification should be read as:

In the context of a library, the existence of a book starts with its acquisition. The book is then classified, this is, registered in the catalogue. It can then be borrowed and returned to the library many times consecutively. Loans can be renewed and the book can possibly be lost in stead of being returned to the library. Finally the book is removed from the catalogue (declassify) and the set of existing books (remove_book).

Object types can be composed by means of a number of operations. The basic operations are choice, sequence and iteration.

Definition 4.

Let $<\alpha,e>, <\alpha',e'> \in <\mathcal{P}(A), R^*(A)>$, then

$<\alpha,e> + <\alpha',e'> = <\alpha \cup \alpha', e + e'>$

$<\alpha,e> . <\alpha',e'> = <\alpha \cup \alpha', e . e'>$

$<\alpha,e>^* = \sum_{i \in \mathbb{N}} <\alpha,e>^i$ where $<\alpha,e>^i = <\alpha,e^i>$

$= <\alpha,e^*>$

The concurrency operator || expresses the fact that object types synchronize on common event types. Rather than giving an axiomatic definition, the parallel-operator is defined by means of sets of accepted sequences of event types. Every regular expression defines a Regular Language, this is a set of scenarios over A. A scenario (or sentence) over A is a finite sequence of event types from A, where '^' acts as the concatenation operator and 1 as the empty scenario. A* is the set of all possible scenarios over A.

Definition 5.

The regular language of a regular expression is a subset of A* defined by

$L(0) = \emptyset$

$L(1) = \{1\}$

$\forall a \in A : L(a) = \{a\}$

$\forall e, e' \in R^*(A)$: $L(e + e') = L(e) \cup L(e')$, $L(e.e') = L(e).L(e')$, $L(e^*) = L(e)^*$

where $L(e).L(e') = \{ s\hat{}t \mid s \in L(e) \text{ and } t \in L(e')\}$

and $L(e)^* = \{1\} \cup L(e) \cup L(e).L(e) \cup L(e).L(e).L(e) \cup L(e).L(e).L(e).L(e) \cup ...$

The language of an object type is the language of its regular expression: $L(<\alpha,e>) = L(e)$

In order to compare the scenarios of two different object types relative to the common events, a projection operator \B, with $B \subseteq A$, is defined as follows:

Definition 6.

Let $B \subseteq A$. Then

$1\backslash B = 1$

$\forall a \in A : (a\backslash B = 1 \Leftrightarrow a \notin B)$ and $(a\backslash B = a \Leftrightarrow a \in B)$

$\forall s, t \in A^* : (s\hat{}t)\backslash B = s\backslash B \hat{} t\backslash B$

Example 9.

Suppose the library example is extended with an object type LOAN that is defined as follows:

LOAN = <{borrow, renew, return, lose}, borrow.(renew)*.(return + lose)>

According to the definition given in example 8 the following is a possible scenario for the object type book:

acquire^classify^borrow^renew^return^borrow^return^declassify^remove_book

Looking at this scenario of BOOK from the point of view of LOAN results in the following:

1^1^borrow^renew^return^borrow^return^1^1 = borrow^renew^return^borrow^return

which is a scenario that is perfectly acceptable from the point of view of LOAN.

When two object types run concurrently, only those scenarios are valid where both object types agree on the sequence of common event types:

Definition 7.

Let $P, Q \in <\mathcal{P}(A), R^*(A)>$

$P \parallel Q = <S_A P \cup S_A Q, e''>$ with $e'' \in R^*(A)$ such that

$L(e'') = \{ s \in (S_A P \cup S_A Q)^* \mid s\backslash S_A P \in L(P) \text{ and } s\backslash S_A Q \in L(Q)\}$

The regular expression e'' always exists as is proved by the theory on Finite State Machines [12, theorem 8-7, p. 252]. This ||-operator can be used to calculate the behaviour defined by composite conceptual schemas [7, 23] as illustrated in the next example.

Example 10.

Suppose we have the following definition for the object types BOOK and LOAN:

BOOK = <{acquire, catalogue, borrow, renew, return, sell, lose},
acquire.catalogue.(borrow + renew + return)*.(sell + lose)>

LOAN = <{borrow, renew, return, lose},borrow.(renew)*.(return + lose)>

The behaviour of a book that can be on loan zero, one or more times consecutively is:

S_R(BOOK || (LOAN)*)
= S_R(BOOK || <{borrow, renew, return, lose},(borrow.(renew)*.(return + lose))*>)
= acquire.catalogue.[borrow.(renew)*.return]*.[sell + borrow.(renew)*.lose]

Objects are occurrences of object types. Every object has a unique identity, which for simplicity is assumed to be a natural number. As there can be a countable infinite number of objects, we simply assume that the set of objects is the set of natural numbers IN. Objects can be classified into object types, and each object is of exactly one type. This is expressed by the classification function type(.) and its inverse function, called oids[1](.), which collects all objects of a same type, and which must be disjoint, because an object can be of at most one type:

Definition 8.

Let $\mathcal{M}$ be a conceptual schema. Then
type : IN oids(.) : $\mathcal{M}$

Events are supposed to be atomic and to have no duration and can therefore uniquely be characterized by the point of time at which they take place. As we assume that time is discrete and linear, the set of time values could be represented by IN. In order to avoid confusion with the set of objects, the set of events is called T and events are written as underlined numbers. Each event from T has a type and a list of participating objects, called the objectlist (short: "olist"). This objectlist is subject to some restrictions: it must contain exactly one object for each object type having the corresponding event type in its alphabet.

Definition 9.

Let $\mathcal{M}$ be a set of object types and IN be the set of objects.
A *typed objectlist* is a set of pairs P:p where each pair P:p must satisfy the restriction type(p)=P.
The *set of typed objectlists* is denoted <$\mathcal{M}$:IN>.

type : T olist : T «M :IN> : t « olist(t) such that " P Î M : type(t) Î SAP Ì $! p Î oids(P) : P:p Î olist(t)»

Example 11.

Suppose we have an object of type BOOK with oid 100 and an object of type MEMBER with oid 200 (that is, type(100) = BOOK, type(200) = MEMBER). The fact that member 200 borrows book 100 at time 1023 can be denoted by the event 1023 with
type(1023) = borrow
olist(1023) = {BOOK: 100, MEMBER: 200}

4.2. Role types

From a mathematical point of view, using role types is the same as allowing the use of the parallel-operator for specifying sequence restrictions. Role types are not considered to be independent object types having own occurrences. Rather, a role type allows to specify multiple sequence restrictions that apply in parallel to the same object type. And as parallel composition of regular expressions results in a regular expression, all definitions can remain unchanged.

The following notations and schema definitions are introduced.

Definition 10.

---

[1]. oids is a shortcut for "object identifications".

Let $\mathcal{M}$ be a conceptual schema.
Let $P \in \mathcal{M}$ and $R_1, ..., R_n \in \langle \mathcal{P}(A), R^*(A) \rangle$
If $R_1, ..., R_n$ are role types for P, this is denoted as: $P \rho R_1, ..., P \rho R_n$

Role types must satisfy the following restriction:
$\forall R_i, P \rho R_i : S_A R_i \subseteq m(P)$

The composite behaviour of P, denoted as $\rho(P, R_1, ..., R_n)$ is then by default $P \parallel R_1^* \parallel ... \parallel R_n^*$, unless otherwise specified.

By requiring that object creation and destruction be modelled separately from roles, it is ensured that the base object type always exists before a role is created and that the base object can not end before the lifecycle of all its roles have ended.

Example 12.
The following example is adapted from [13]:
Base class: MOVIE_STAR with
S_RMOVIE_STAR = create_star.(marry + divorce + hire + fire)*.delete_star

Roles: MOVIE_STAR ρ STAR-PRIVATE, STAR-PROFESSION
S_RSTAR-PRIVATE = (marry.divorce)*.(marry + 1)
S_RSTAR-PROFESSION = (hire.fire)*.(hire + 1)

Composite behaviour is specified as:
ρ(MOVIE_STAR, STAR-PRIVATE, STAR-PROFESSION) = MOVIE_STAR || STAR-PRIVATE || STAR-PROFESSION

4.3. Generalization/Specialization types

The following definition formalizes the structural aspects of generalization/specialization:

Definition 11.
$<^+$ is a hierarchical partial order on $\mathcal{M}$
$\Leftrightarrow$ (1) $\forall G \in \mathcal{M}: G <^+ G$
(2) $\forall G, S \in \mathcal{M}: G <^+ S$ and $S <^+ G \Rightarrow G = S$
(3) $\forall G, S, T \in \mathcal{M}: G <^+ S$ and $S <^+ T \Rightarrow G <^+ T$
(4) $\forall G, G', S \in \mathcal{M}: G <^+ S$ and $G' <^+ S \Rightarrow G = G'$

$<$ is the cover of $<^+$.
$<^+$ can be recovered by taking the reflexive transitive closure of $<$

Let $G, S \in \mathcal{M}$. If $G < S$ then G is a generalization of S and S is a specialization of G

$\gamma(P)$ is the set of *parent types* of a type P:
$\gamma(P) = \{G \in \mathcal{M} \mid G <^+ P\}$

$oids^+$ is a function that takes the deep extent of an object type and is defined as:

$$oids^+(P) = \bigcup_{P \in \gamma(S)} oids(S)$$

Example 13

Imagine a library where different kinds of items can be consulted. Suppose we have the object types ITEM, BOOK, JOURNAL, MULTI-MEDIA-ITEM, VIDEOTAPE, CD-ROM with the following hierarchy:

ITEM < BOOK, ITEM < JOURNAL, ITEM < MUTLI-MEDIA-ITEM,

MUTLI-MEDIA-ITEM < CD-ROM, MUTLI-MEDIA-ITEM < VIDEOTAPE

Then for example

ITEM $<^+$ VIDEOTAPE

γ(VIDEOTAPE) = {VIDEOTAPE, ITEM, MUTLI-MEDIA-ITEM}

$oids^+$(MUTLI-MEDIA-ITEM) = oids(MUTLI-MEDIA-ITEM) $\cup$ oids(CD-ROM) $\cup$ oids(VIDEOTAPE)

Unlike methods where the specialization type is a subset of the generalization type [8], we maintain the requirement that the function oids(.) must be a disjoint function, even in the presence of generalization/specialization. As a result, the generalization type and specialization type have disjoint extensions and overlapping specializations are not allowed:

$\forall$ G, S $\in \mathcal{M}$: G < S $\Rightarrow$ oids(G) $\cap$ oids(S) = ý

$\forall$ G, S^i, $S^j \in \mathcal{M}$, G < S^i, G < S^j : oids(S^i) $\cap$ oids(S^j) = ý

However, the superset/subset hierarchy can be recovered by working with the deep extent of object types:

Property 1.

$\forall$ G, S $\in \mathcal{M}$: G < S $\Rightarrow$ $oids^+(S) \subseteq oids^+(G)$

Proof

$\forall$ P $\in \mathcal{M}$: S $<^+$ P and G < S $\Rightarrow$ G $<^+$ P. This implies that S $\in \gamma(P) \Rightarrow$ G $\in \gamma(P)$. From the definition of $oids^+$(.) it follows that $oids^+(S) \subseteq oids^+(G)$. ■

This demonstrates that proposals for the use of the generalization/specialization hierarchy that are apparently contradictory as explained in the introduction, can be reconciled when using the proper formalization.

With these definitions it is also possible to introduce the notion of a *total* IS-A relationship (or *abstract* generalization type) by requiring that oids(G) = ý.

In order to allow a smooth transition from conceptual modelling to OOPL's, universal substitutability must be ensured. The principle of substitutivity says that an object of the generalization type is allowed to be replaced by an object of one of its specialization types in all circumstances. In particular, when an object p can be taken from the extension of a type P (p $\in$ oids(P)), p can now be taken from the *deep* extent of this type P (p $\in$ $oids^+$(P)). And when an object p is required to be of type P (type(p) = P), p is now allowed to be of a subtype of P (P $<^+$ type(p)). Definition 9 can thus be reformulated as follows:

Definition 9 revisited.

Let $\mathcal{M}$ be a set of object types and IN be the set of objects.

A *typed objectlist* is a set of pairs P:p where each pair P:p must satisfy the restriction P $<^+$ type(p).

olist : T ↔M:IN> : τ ↔ ολιστ(τ) συχη τηατ ≈∀ P ∈ $\mathcal{M}$: type(t) ∈ S_AP ⇒ ∃! p ∈ oids$^+$(P) : P:p ∈ olist(t)

Inheritance and substitutivity also have consequences for the specification of sequence restrictions. In the first place every specialization inherits all the event types of the generalization type. In order to discern between own event types and inherited event types the operator α(.) is defined on object types: S_A denotes the set of proper event types, while α(.) denotes the set of all inherited event types plus the own event types.

Definition 12.

$$\alpha(P) = \bigcup_{Q <^+ P} S_A Q$$

Note that each object can have its own methods for each event type in its alphabet. Thus if an object type BOOK inherits the event create_item from the object type ITEM then BOOK.create_item can have a different content than ITEM.create_item (in order to deal with additional attributes, for example). An object type can impose sequence restrictions on all its inherited and own event types and thus the definition of an object type must be modified as follows:

Definition 3 revisited.

An object type P is a tuple <α,e> over <$\mathcal{P}$(A), R*(A)> such that e ≠ 0 and φ(e) = α(P).
It must be possible to partition α(P) into three pairwise disjoint sets c(P), m(P) and d(P).
By definition ∀ G ∈ $\mathcal{M}$ such that G <$^+$ P : c(G) ⊆ c(P), m(G) ⊆ m(P) and d(G) ⊆ d(P)
c(P) and d(P) must not be empty.

The definition of the parallel-operator must be adapted in the same way:

Definition 7 revisited.

Let P, Q ∈ <$\mathcal{P}$(A), R*(A)>
P || Q = <S_AP ∪ S_AQ, e"> with e" ∈ R*(A) such that
L(e") = { s ∈ (α(P) ∪ α(Q))* | s\α(P) ∈ L(P) and s\α(Q) ∈ L(Q)}

4.4. Consistency checking between static and dynamic schemas.

The generalization/specialization hierarchy can be considered as part of the static schema while the sequence restrictions are part of the dynamic schemas [7, 23]. In this section the relation between the static and dynamic schema is investigated and requirements for consistency are formulated.

A subtype S^i inherits the sequence restrictions of its supertype G. In most methods, the subtype is allowed to restrict or redefine the behaviour of its parent. A mathematical analysis reveals that restriction together with the principle of substitutivity requires that a subtype must leave the sequence restrictions imposed on the inherited event types unchanged.

In order to be able to compare the sequence restrictions of a generalization and a specialization object type, the following order on regular expressions is defined:

Definition 13.

$\forall$ e, e' $\in$ R*(A) define $\quad e \leq e' \Leftrightarrow e + e' = e'$

Property 2.

$\leq$ is a partial order on R*(A)

Proof

$\forall$ e,e',e" $\in$ R*(A) :

1. $e \leq e$ because $e + e = e$
2. $e \leq e'$ and $e' \leq e \Rightarrow e = e'$
   because $\quad e \leq e' \Rightarrow e + e' = e'$ and $e' \leq e \Rightarrow e' + e = e$
   Thus $\quad e = e' + e = e + e' = e'$
3. $e \leq e'$ and $e' \leq e" \Rightarrow e \leq e"$
   because $\quad e \leq e' \Rightarrow e + e' = e'$ and $e' \leq e" \Rightarrow e' + e" = e"$
   Thus $\quad e + e" = e + (e' + e") = (e + e') + e" = e' + e" = e"$ ■

The regular expression of an object type determines a set of scenarios (sequences of events) that are accepted by this object type. Intuitively, $e \leq e'$ means that the set of scenarios defined by e' includes the set of scenarios defined by e. In general, a specialization type has additional event types beside the inherited event types ($\alpha(G) \subset \alpha(S)$). In order to analyze the restrictions imposed by a specialization type on the *inherited* event types only, we need a projection operator that drops irrelevant event types from the behaviour description:

Definition 14.

Let $B \subseteq A$, $a \in A$, e, e' $\in$ R*(A). Then define

$1\backslash B = 1$

$(a\backslash B = a \Leftrightarrow a \in B)$ and $(a\backslash B = 1 \Leftrightarrow a \notin B)$

$(e + e')\backslash B = e\backslash B + e'\backslash B$, $(e.e')\backslash B = e\backslash B \,.\, e'\backslash B$ and $((e)^*)\backslash B = (e\backslash B)^*$

This partial order on regular expressions can be used to define a partial order (or taxonomy) on processes:

Definition 15.

Let P, Q $\in$ <$\mathcal{P}$(A), R*(A)> then define

$$P \leq' Q \Leftrightarrow \alpha(P) \subseteq \alpha(Q) \text{ and } S_R P \leq (S_R Q)\backslash\alpha(P)$$

Property 3

$\leq'$ is a partial order on object types.

Proof. This follows from the fact that $\leq$ is a partial order on regular expressions and the fact that $\subseteq$ is a partial order on sets.■

Substitutivity requires that an operation valid for a supertype always be valid for a subtype in all situations. In other words, a subtype must have loser preconditions and stronger postconditions than its supertype [15]. For sequence restriction, this means that a subtype must accepts all scenarios of its supertype and thus $S_R G \leq S_R S \backslash\alpha(G)$. As $\alpha(G) \subseteq \alpha(S)$ for any G and S with G < S, requiring substitutivity is the same as requiring that G $\leq'$ S for every G and S with G < S. As a result the partial order $\leq'$ on object types can be considered as the formal pendant of the generalization/specialization hierarchy.

In most OOPL's inherited behaviour can be redefined by extending it or restricting it. If restriction means that the subtype can define additional sequence constraints on the inherited event types, the subtype will accept less scenarios than those defined by the parent type, thus $S_R S \backslash \alpha(G) \leq S_R G$. As $\leq$ is a partial order on regular expressions, combining restriction with universal substitutability results in the requirement that $S_R S \backslash \alpha(G) = S_R G$, which means that a subtype must leave the sequence restrictions on inherited event types unchanged. The following example illustrates how the partial order $\leq$ can be used to analyze specifications in a formal way.

Example 14.

Imagine a library where different kinds of items can be consulted: single issues of journals, volumes of journals, books, CD-roms, ... and so on. All available items can be searched for by means of an on-line catalogue. Issues of journals can not be lend out; books and volumes of journals can. Only loans of books can be renewed. To prevent loss, CD-roms are kept in a separate place and must be lend out at the loan desk. They must not leave the library. Equipment is provided to view CD-roms and to print information. Printing is charged when the CD-rom is returned.

A possible library model could contain the following object type definitions:

ITEM = <{create_item, classify, borrow, return, declassify, remove_item},
create_item.classify.(borrow.return)*.declassify.remove_item>

BOOK = <{renew, lose},
create_item.classify.(borrow.(renew)*.return)*.(borrow.(renew)*.lose + declassify).remove_item>

VOLUME = <{lose}, create_item.classify.(borrow.return)*.(borrow.lose + 1).declassify.remove_item>

ISSUE = <{}, create_item.classify.declassify.remove_item>

CD-ROM = <{print}, create_item.classify.(borrow.(print)*.return)*.declassify.remove_item>

BOOK, ISSUE, VOLUME, CD-ROM are specialization types of ITEM:
ITEM < BOOK, ITEM < ISSUE, ITEM < VOLUME, ITEM < CD-ROM

Comparing the sequence restrictions of generalization and specialization types results in the following:

S_RBOOK \ α(ITEM)
= create_item.classify.(borrow.(1)*.return)*.(borrow.(1)*.1 + 1).declassify.remove_item>
= create_item.classify.(borrow.return)*.(borrow + 1).declassify.remove_item>

Which is loser than S_RITEM itself: S_RITEM $\leq$ S_RBOOK\$\alpha$(ITEM). The same result applies to VOLUME: S_RITEM $\leq$ S_RVOLUME\$\alpha$(ITEM). This means that substitutivity is ensured, but that BOOK and VOLUME are *not restricting* but *broadening* the behaviour of ITEM.

On the other hand, ISSUE is *restricting* the inherited behaviour:

S_RISSUE \ α(ITEM)
= create_item.classify.declassify.remove_item

And thus S_RISSUE\α(ITEM) ≤ S_RITEM. In this case substitutivity is *not* guarantied. This problem can be resolved by removing the event types 'borrow', 'return' and 'declassify' from the specification of the object type ITEM. Possibly a supplementary level can be added in the generalization/specialization hierarchy:

ITEM < LOAN_ITEM, ITEM < ISSUE, ITEM < CD-ROM
LOAN_ITEM < BOOK, LOAN_ITEM < VOLUME

with

ITEM = <{create_item, classify, declassify, remove_item},
create_item.classify.declassify,remove_item>

ISSUE = <{}, create_item.classify.declassify.remove_item>

CD-ROM = <{borrow, return, print},
create_item.classify.(borrow.(print)*.return)*.declassify.remove_item>

LOAN_ITEM = <{borrow, return, lose},
create_item.classify.(borrow.return)*.(borrow.lose + 1).declassify. remove_item>

BOOK = <{renew},
create_item.classify.(borrow.(renew)*.return)*.(borrow.(renew)*.lose + 1).declassify.remove_item>

VOLUME = <{}, create_item.classify.(borrow.return)*.(borrow.lose + 1).declassify.remove_item

These specifications combine the requirements of substitutivity and restriction. Indeed, the inherited sequence restrictions remain unchanged:
S_RLOAN_ITEM\α(ITEM) = S_RITEM,
S_RISSUE\α(ITEM) = S_RITEM,
S_RCD-ROM\α(ITEM) = S_RITEM
S_RBOOK\α(LOAN_ITEM) = S_RLOAN_ITEM,
S_RVOLUME\α(LOAN_ITEM) = S_RLOAN_ITEM,

5. RELATION TO PREVIOUS, CURRENT AND OTHER WORK

A conceptual schema can be considered as a set of object type definitions. An information system implemented according to this schema will be composed of occurrences of these object types that are running concurrently and synchronise were specified so. One of the requirements that guided the development of a Process Algebra for M.E.R.O.DE. is that we should be able to compute the overall behaviour of an implemented system from individual object type behaviour descriptions. In [7] it was presented how such a computation can be done for conceptual schemas that do not use the concepts of Generalisation/Specialisation and Role. In order to complete the results presented in [7], we must define the behaviour of a system composed of object types with specialisations and roles. The global behaviour of an object type is written as θ(P) and is defined as follows for the case of specialisation and roles:

Definition 16.
Let P, S_1, ..., S_n, Q, R_1, ..., R_n ∈ $\mathcal{M}$ such that

$P \rho R_1, ..., P \rho R_n$ and $\forall$ i: $P < S_i$ then

$\theta P = [\rho(P, \theta R, ..., \theta R_n) + \theta S_1 + ... + \theta S_n]$

where $\rho(P, \theta R, ..., \theta R_n)$ is the composite behaviour of P as defined in definition 10.

Indeed, as a result of the substitutivity principle, any place where an object of the generalisation type is required, it can be replaced by an object of one of the specialization types. The expected behaviour is thus the behaviour of the generalisation or the behaviour of one of the specialisations.

When viewed from the static point of view, the formalisation of the "IS A" hierarchy and the concept of role as presented in this paper, results in more restrictive concepts with less freedom for the modelers and maybe less expressiveness than the concept of generalization/specialization as formalized by e.g. [24]. This is due to the explicit attention that is paid to inheritance of behaviour and the requirement that it must be possible to derive global system behaviour from individual object type descriptions.

The consistency requirements formulated in this paper, and especially the requirements formulated in section 4.4, can easily be computed by means of the classical algorithms for FSMs. Today a CASE-tool is under development that will incorporate all the consistency rules presented in this paper and will allow for automatic consistency checking. With the availability of such a CASE-tool, designers can formulate their specifications by means of the classical techniques of ER and FSMs. The only thing they have to care about is to make the right choice between Generalisation/Specialisation and Role according to the decision table presented in section 3. The case-tools will then automatically do all consistency checking and, in the case of roles, calculate the composite life cycle. The absence of a CASE-tool does however not mean that the designer needs knowledge about the formal specifications underlying the techniques of ER and FSMs. The $\leq$' order (definition 15) is easy to check with pen and paper. Only the calculation of composite life cycles for object types with roles, can sometimes be tedious. Moreover, it is easy to implement the ideas proposed in this paper on top of current OO CASE-tools, provided they have a rigourous definition for object behaviour.

6. Conclusion

In this paper a coherent and formal definition has been developed for the conceptual IS-A relationship. It integrates the notions of structure sharing, behaviour sharing, strict inheritance and universal substitutability. Using this relationship at the conceptual level will help for a better use of the inheritance mechanism at the programming level.

The formal basis for the definition by means of process algebra shows how apparently contradictory modelling formalisms can be reconciled and how mathematical analysis can effectively help to investigate the consequences of particular modelling options.

The formalization presented here is based on the use of ER-modeling as technique for modeling static aspects and the use of Finite State machines (or any equivalent formalism) for modeling dynamic aspects. In principle, it is applicable to any method that uses these techniques.

Further extensions of this work include the handling of multiple inheritance and the inheritance aspects of attributes in general. The first one seems straightforward with an appropriate definition of syntax and semantics. The latter requires an algebraic approach to abstract data types.

REFERENCES

[1] James M. Armstrong, Richard E. Mitchell, Uses and abuses of inheritance, *Software Engineering Journal*, (January 1994), 19-26

[2] J.C.M. Baeten, *Procesalgebra* (Kluwer programmatuurkunde, 1986).

[3] Booch, G. *Object Oriented Analysis and Design with Applications*. Second Edition, (Benjamin/Cummings, Redwood City, CA, 1994).

[4] M.L. Brodie, D. Ridjanovic, On the Design and Specification of Database Transactions, in M.L Brodie, J. Mylopoulos, J.W. Smith (eds.), *On conceptual Modeling*, (Springer Verlag New York, 1984) 277-306

[5] P.P. Chen, *The Entity Relationship Approach to logical Database Design*, QED information sciences Wellesley (Mass.),1977

[6] Peter Coad, Edward Yourdon, *Object-Oriented analysis* (Prentice Hall, Englewood Cliffs, New Jersey, 1991)

[7] Dedene G., Snoeck M., Formal deadlock elimination in an object oriented conceptual schema, *Data and Knowledge Engineering*, Vol. 15 (1995) 1-30

[8] D.W. Embley, B.D. Kurtz, S.N. Woodfield, *Object-Oriented Systems Analysis: A Model-Driven Approach* (Yourdon Press, Prentice Hall, Englewood Cliffs, 1992)

[9] L.J.B. Essink, W.J. Erhart, Object Modelling and System Dynamics in the Conceptualization Stages of Information Systems Development, in F. Van Assche, B. Moulin, C. Rolland (eds.) Object Oriented Approach in Information Systems, *Proceedings of the IFIP TC8/WG8.1 Working Conference on the Object Oriented Approach in Information Systems*, Quebec City, Canada, 28-31 October, 1991 (North Holland, 1991) 89-116

[10] M. Hammer, D. McLeod, Database Description with SDM: A Semantic Database Model, *ACM Transactions on Database Systems*, Vol. 6, No. 3 (September 1981) 351-386

[11] C. A. R. Hoare, *Communicating Sequential Processes* (Prentice-Hall International, Series in Computer Science, 1985).

[12] John E. Hopcroft, Jeffrey D. Ullman, *Formal languages and their relation to automata* (Addison Wesley Publishing Company, 1969)

[13] M. A. Jackson, J. R. Cameron, *System Development*, (Prentice Hall Englewood Cliffs (N.J.), 1983)

[14] Ivar Jacobson et al. *Object-Oriented Software Engineering: A Use Case Driven Approach*, (Addison-Wesley Publishing Company, Mass., 1994)

[15] B. Meyer, *Object Oriented Software Construction*, (Prentice Hall International, 1988)

[16] R. Milner, *A calculus of communicating systems* (Springer Berlin, Lecture Notes in Computer Science, 1980).

[17] T.W. Olle, Comparative Review of Information Systems Design Methodologies - stage 1: Taking Stock, in T.W. Olle, H.G. Sol, A.A. Verrijn-Stuart (eds.), Information Systems Design Methodologies: a comparative review, *Proceedings of the IFIP WG8.1 Working Conference on Comparative Review of Information Systems Design Methodologies*, Noorderwijkhout, The Netherlands 10-14 May, 1982 (North Holland, 1983) 1-14

[18] B. Pernici, Objects with Roles, *Proceedings of the Conference on Office Information Systems*, SIGOIS Bulletin 11, (2/3),(1990) 205-215

[19] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, *Object Oriented Modeling and Design* (Prentice Hall International, Englewood Cliffs, New Jersey, 1991)

[20] C. Sernadas, P. Resende, P. Gouveia, A. Sernadas, In-the-large object-oriented design of Information Systems, in F. Van Assche, B. Moulin, C. Rolland (eds.) Object Oriented Approach in Information Systems, *Proceedings of the IFIP TC8/WG8.1 Working Conference on the Object Oriented Approach in Information Systems*, Quebec City, Canada, 28-31 October, 1991 (North Holland, 1991) 209-232

[21] S. Shlaer, S.J. Mellor, *Object-Oriented Systems Analysis: Modeling the World in Data* (Prentice Hall, Englewood Cliffs, New Jersey, 1988)

[22] S. Shlaer, S.J. Mellor, *Object Lifecycles: Modeling the World in States* (Prentice Hall, Englewood Cliffs, New Jersey, 1992).

[23] Snoeck Monique, *On a Process Algebra Approach to the construction and analysis of M.E.R.O.DE.-based conceptual models*, Phd. Dissertation (Katholieke Universiteit Leuven, Faculty of Science & Department of Computer Science, May 1995)

[24] ter Hofstede A.H.M., *Information Modeling in Data Intensive Domains*, Phd. Dissertation (Katholieke Universiteit Nijmegen, September 1993)

[25] Vanthienen Jan, Knowledge Acquisition and Validation Using a Decision Table Engineering Workbench, Proc. of the World Congress on Expert Systems, Dec. 16-19, Orlando (Florida) (Pergamon Press, 1991) 1861-1868