*Journal of Universal Computer Science, vol. 2, no. 9 (1996), 623-636*
*submitted: 19/9/96, accepted: 19/9/96, appeared: 28/9/96* 

# Linguistic Tools for Modelling Alter Egos in Cyberspace: Who is responsible?[1]

R.P. van de Riet and J.F.M. Burg
Department of Computer Science
Vrije Universiteit
de Boelelaan 1081a
Amsterdam, The Netherlands
Tel: +31 20 4447757
Fax: +31 20 4447653
{vdriet,jfmburg}@cs.vu.nl

**Abstract:** Alter Egos represent people in Cyberspace. An Alter Ego is a kind of intelligent agent who is active in performing actions in behalf of the person it represents. How these Alter Egos can be modelled and constructed is discussed. In this context the question whether Alter Egos can be held responsible is studied. The tools we use for the modelling process are using Linguistic knowledge and are logically founded.
**Categories:** Security and Privacy, Intelligent Agents, Building Knowledge Bases, Linguistic Tools, Modelling

## 1 Introduction

Modelling in Cyberspace can be done from two different perspectives: one is modelling the information which is available, the other one is modelling the people living in it. With this we mean that people will be (are) represented by active agents, objects residing in computer systems. They represent real people when real people leave doing their business to them, as already happens with ordering goods and automatic payments. We call these objects *Alter Egos*. The approach presented here is a pure technical one, in contrast with for example Erickson [6] who describes the World-Wide Web (WWW) as a social hypertext, in which the WWW-nodes are becoming (social) representations of people.

We are studying how Alter Egos can be modelled when they are being used for home banking and aspects of privacy and security are involved. Also we came upon the question of what is meant when we say that these objects are *responsible* while developing certain modelling tools. These tools are being used to model complex behaviour of people and computer systems, communicating with each other. Dealing with people involves also dealing with responsible people. That means they can be held responsible for their deeds; they can be punished when they do not do what they promised. So communicating with people means that one always has to take into account that these people do not do what they promised (or committed themselves to), on the other hand computer systems always do what they are programmed to do (we do not talk about failures of

[1] This paper is an extension of "Linguistic Tools for Modelling Alter Egos in Cyberspace: Who is responsible?", R.P. van de Riet and J.F.M. Burg, *WebNet '96: The First World Conference of the Web Society*, San Francisco, CA, October 1996

these systems, but assume that they function correctly, according to the specifications).

Naturally, then the question comes up when people are represented by Alter Egos whether these Alter Egos can be held responsible, and what this means.

Let us see what Webster has on the word *responsible*:

**re.spon.si.ble**:

1a: liable to be called upon to answer as the primary cause, motive, or agent
1b: liable to legal review or in case of fault to penalties
2a: able to answer for one's conduct and obligations : TRUSTWORTHY
2b: being a free moral agent
3 : involving responsibility or accountability
4 : politically answerable; esp : required to submit to the electorate if defeated by the legislatur

Three things we distill from this definition: *responsibility* means:

- free to act or not to act according to the promises
- able to explain one's deeds
- in case of fault: penalties.

In the next sections we will first introduce the notion of Alter Ego, then the modelling of behaviour will be treated, finally we will come back to the above definition of *responsibility* and see what it means for Alter Egos.

## 2 An Alter Ego for a Cybernaut

Cyberspace is considered to be the space in which people and information is connected through computer networks. Nowadays we see the effect of Internet; in the near future the new information superhighway will provide the facilities. In this paper the people in Cyberspace will be called *cybernauts*. One can look at them from two different perspectives: as active persons communicating with each other and with information systems (e.g. through the WWW) and as objects who are themselves being dealt with, e.g. by governmental agencies or by Banks. The idea is that the Social Security Number (SSN) will be replaced by an object. Probably recorded in object-oriented databases in the same way as people nowadays are represented in ordinary databases using SSN.

In [11] we have introduced the notion of Alter Ego which combines both aspects: an object as subject, acting in behalf of its owner as a so-called intelligent agent, and as an object representing its owner.

The issue of responsibility is relevant for both aspects mentioned. However, we will be interested only in modelling the behaviour of an Alter Ego.

In Western Society citizens are represented by their SSN. It is not very far fetched to assume that in the near future these citizens will be represented by an object (in the object-oriented (OO) sense), where the SSN is replaced by an object identifier (or is the object identifier itself).

By using OO methodology it is possible to construct logically quite sophisticated objects representing these cybernauts in Cyberspace.

Consider first the possibility that a central agency, like the Civil Administration (CA) keeps track of all the personal properties of a cybernaut, such as: name, address, birthdate and marital status.

Consider next that a cybernaut can inscribe in a University U as student and has additional properties such as field, and year.

Let us further suppose that the cybernaut may also be a customer of some bank B, having an account with a number and an amount.

Logically we can now declare the following types, which is done using the syntax of Mokum (see [4]):

```
type personT is_a thing
    has_a name: nameT
    has_a address: addressT
    has_a birthdate: dateT
    has_a marital_status: msT.

type studentT is_a personT
    has_a field: fieldT
    has_a year: int.

type customerT is_a personT
    has_a account: accountT.

type accountT is_a thing
    has_a number: nrT
    has_a amount: int.
```

All identifiers ending with "T" denote (user-defined) types, either defined here, such as personT, or defined elsewhere, such as nameT, which e.g. could have been defined as follows:

```
type nameT is_a thing
    has_a first_name: string
    has_a last_name: string.
```

To take into account that a cybernaut also has some private information, such as a calendar and hobbies, we also have:

```
type personal_assistantT is_a personT
    has_a calendar: calendarT
    has_a hobbies: collection_of hobbyT.
```

Evidently, we assume here that each cybernaut has a personal assistant object at her disposal, which may actually reside in a home-based PC or a personal digital assistant (PDA) a smart card or even a wrist watch.

Objects have (one or more) types, which can be structured in the form of a *is_a* hierarchy, as shown above. As usual the *is_a* hierarchy means that when a type t1 is defined as *is_a* type t2 then it inherits all the attributes of t2. So, in the above example, a student has attributes:

field, year and
name, address, birthdate, marital_status.

A cybernaut is always a person, so there must be an instance of personT, representing this cybernaut. If she is also a student at U or a customer of B there must also be instances of the types: studentT and customerT. These objects together form one object called the Alter Ego.

Statically the structure of an object is quite simple. It is their dynamic behaviour which makes them very powerful and fit for their service as Alter Ego.

To each type a so-called script can be added in which the behaviour can be defined. In principle each object is a finite state automaton, which reacts on signals coming from elsewhere (usually messages sent by other objects to this object) in a predefined way. It is the way these scripts are defined that is studied in this paper.

Consider now an Alter Ego which is an object of type personT and of type studentT. We shall say that in this case there are two sub-objects: a person object and a student object; the Alter Ego object is the single object consisting of both sub-objects.

Logically these objects belong to each other, through the *is_a* relationship. Physically they are located in quite different worlds, with their own protection rules.

In Figure 1 these worlds are shown. There are fire-proof walls between these worlds to protect private information. We will assume that private information is kept in attributes declared 'private' in Mokum terminology [10, 11].

We will say a few words here about problems connected to the protection problems. In a more general setting these are dealt with in [13, 8], and in the more specific Mokum environment in [10, 11, 12]. We simply assume that the following rules from Mokum are implemented:

- the epistemic rule which says that private attributes of some type can be read and changed within the script of a subtype; the same holds for so-called keepers of a collection of objects of some type;
- the ontological rule says that only the object itself or the keeper of a collection in which that object lies can read and change the attributes of that object.

Introducing the notion of collection and a so-called keeper of a collection, it is possible to define rules for maintaining integrity and security in a very natural way, almost as is done in actual practice, where a person is held responsible for keeping some rules. In our case an object can take the role of a keeper of some rules for a collection of (other) objects. Such objects can be the hobbies of some cybernaut (as we saw in the example type definition above of 'personal_assistantT'), but they can also be a collection of customer objects being kept by a bank manager:

```
type employeeT is_a personT
    has_a rank: rankT
    has_a salary: int.

type bank_managerT is_a employeeT
    has_a customers: collection_of customerT.

type bank_ast_mgrT is_a bank_managerT
    has_a limit: int.
```

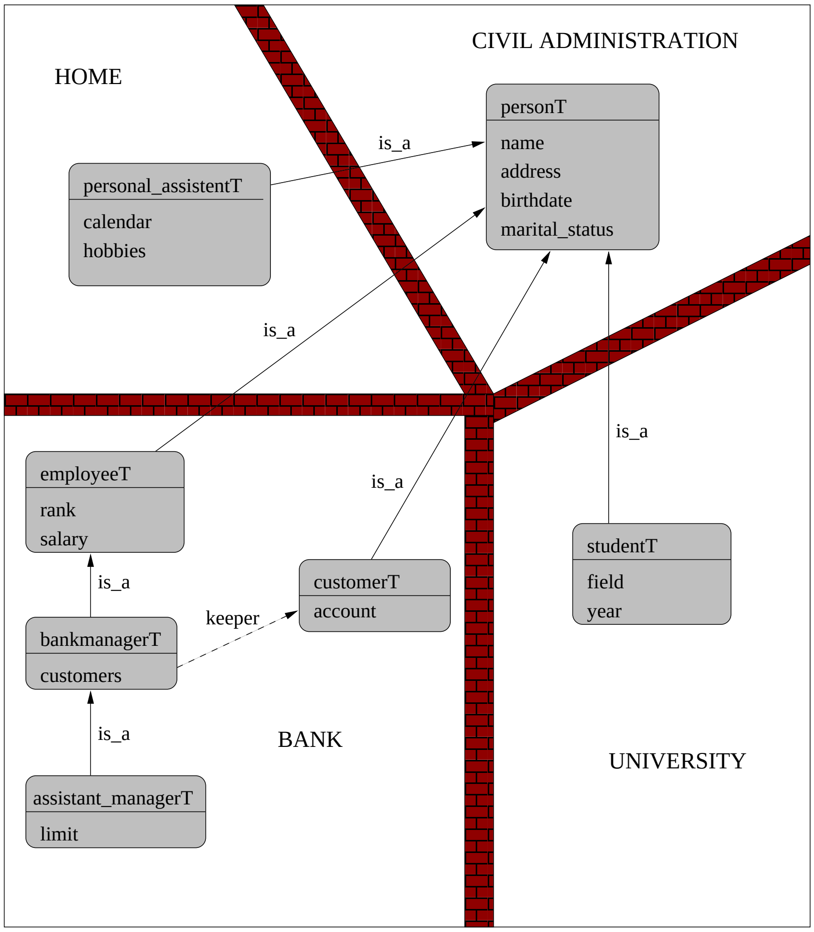


**Figure 1:** Physical worlds of objects

We have thus defined within a bank organization how a bank manager can have an assistant who is responsible for a collection of customers but whose responsibility does not go beyond a certain limit (evidently, the Barings bank did not restrict Nick Leeson to such a limit, which they now painfully regret).

Above we used a few times the word *responsible* in a very natural way. The word means simply what it says: a certain task which has to be carried out, such as dealing with financial accounts of cybernauts.

## 3 Modelling Cyberspace: COLOR-$\mathcal{X}$

### 3.1 COLOR-$\mathcal{X}$ Event Model

Consider the situation of the University library lending a book to a student. We are working on a tool, called COLOR-X [3, 2, 1], by means of which this situation can be described

- very concisely, by means of a diagram,
- very succinctly using linguistic tools, such as WordNet, and
- very helpfully, because Mokum scripts for the objects involved can be generated almost automatically.

The COLOR-X diagram is shown in Figure 2. What this diagram describes is the behaviour of the library system and a human being: the user. After obtaining a pass the user is permitted to borrow a book, which she then has to return within three weeks. Later on we will change the example and put a personal assistant in between. The boxes in the diagram denote actions with some modality:

- **PERMIT** to indicate that the actions are allowed;
- **NEC** to indicate that the actions are necessarily to be performed by the library system. In case the actions are not performed, the library system has a serious problem: it is in an inconsistent state (something comparable with the situation of a personnel database having a record of some person with a negative age).
- **MUST** to indicate that the actions have to be done, but there is a possibility that they are not done, within the prescribed amount of time; in that case there is a special arrow, the lightning arrow, which indicates what action is performed by the library system. We are dealing here with certain actions, such as returning the borrowed book, which the user should do, but she may decide not to do them, or simply she can be in the situation not being able to do them. We shall say that the library system has to deal with a violation in such a case.

We see here that MUST is closely related to the word *responsible* as MUST takes into account promises, not fulfilling promises and *punishment*.

It is evident that being in an inconsistent state is much more serious than dealing with a violation. In the first case the system cannot be trusted anymore and serious measures have to be taken immediately. In the latter case it is a matter of duely waiting, because, according to the diagram, eventually, there comes a moment that the violation will be resolved. Either by the action of the user to return the book, or by the action of the library system to block the user, after several warnings. The nice thing about the COLOR-X diagram is that indeed such a situation will occur, because there are no MUST boxes without a lightning arrow leading to some other box.

### 3.2 The Use of Linguistic Knowledge

The formulae written in the boxes are so-called CPL formulae; the syntax is very close to Functional Grammar (see [5]). Indeed, one can see what kind of things

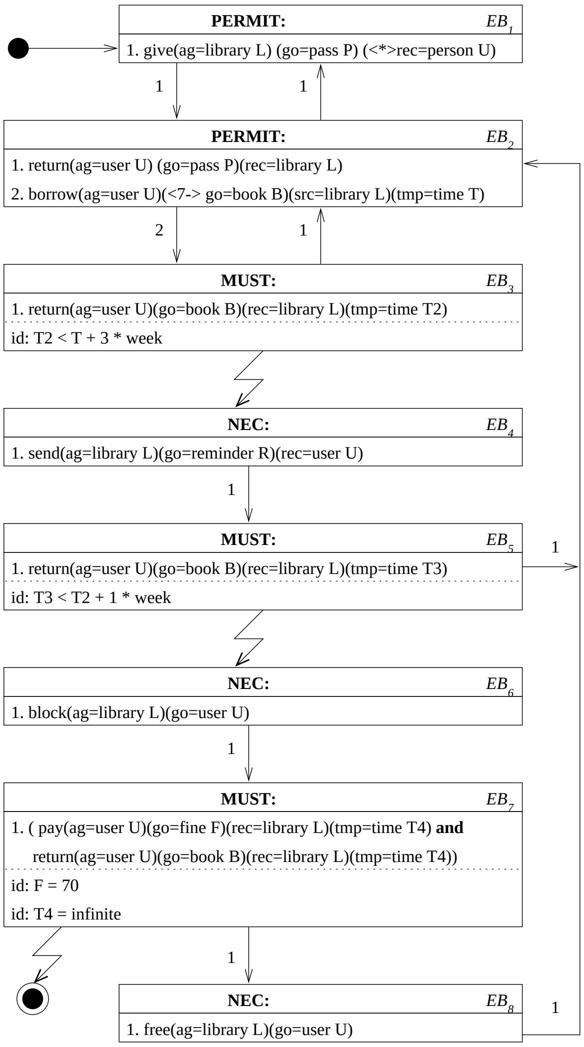


**Figure 2:** The COLOR-X Event Model for the Library System

(objects) may occur as roles/parameters within the formulae: in the borrowing action, the user is the active agent, the library is the source and the book is the goal. In this way one can see which roles are being played by which objects.

In the actual COLOR-X tool heavy use is being made of the WordNet [7] system, which is a lexicon consisting of the meaning (in a computer-friendly form) of some 90.000 words/concepts. Using this lexicon it is for example impossible to define an action in which the book is the actor of the borrow action. Also is it possible to generate finite state automata for the objects involved in our activities: library and user. The result is shown in Figure 3. Note that what is *"to borrow"* for the user is *"to lend"* for the library. Such knowledge is taken from WordNet. Also, *"to give"* a pass for the library is *"to get"* a pass for the user.

In COLOR-X it is also possible to generate natural language sentences describing the model, as is illustrated in Figure 4.

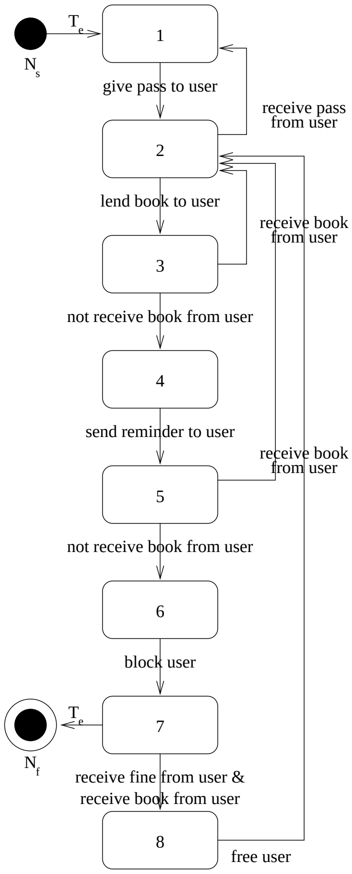


**Figure 3:** The State Transition Diagram for the Library System

### 3.3 The Use of Logics

The modality NEC is defined using modal logic, which is the logic dealing with necessity, while the modalities PERMIT and MUST are defined using deontic logic, which is the logic dealing with obligations. All three modalities can be combined with logical representations of actions by using dynamic logic. Actually, all formulae written in CPL can be translated into formulae in Logic (using predicate-, modal-, dynamic-, temporal- and deontic logic) so that questions of consistency, equivalency, redundancy and even correctness can be dealt with. To

```
[if,a,library,gave,a,pass,to,a,person,then,
the,user,is permitted to,return,the,pass,to,the,library,or,
the,user,is permitted to,borrow,a,book,from,the,library]
[if,an,user,borrowed,one or more,books,from,a,library,at,time,T,
then,an,user,will have to,return,one or more,books,to,the,library,
at,time,T2, where,T2,is less or equal,T + 3 * week]
[if,an,user,borrowed,a,book,from,a,library,at,time,T,and,
an,user,did not,return,the,book,to,the,library,at,time,T2,where,
T2,is smaller than,T + 3*week,then,
the,library,is obliged to,send,a,reminder,to,the,user]
```

**Figure 4:** Generated Natural Language for the Library System

give a few examples how this formalism can be used, let us first take the CPL sentence in the first box of Figure 2:

**PERMIT: ACTION:** give
$(<1>$ ag=library L$)(<1>$ go=pass P$)(<*>$ rec=person U$)$

This has the following deontic logic (operator $\mathcal{P}$) counterpart:

$$\exists L'(card(L') = 1 \wedge \forall X \in L'(library(X)))\wedge$$
$$\exists P'(card(P') = 1 \wedge \forall Y \in P'(pass(Y)))\wedge$$
$$\exists U'(\forall Z(person(Z) \leftrightarrow Z \in U'))\wedge$$
$$\mathcal{P}(give(X, Y, Z)))$$

The CPL sentences from the second and third boxes:

**MUST: ACTION:** return
$(<*>$ ag=user U$)(<1>$ go=book B$)$ $(<1>$ rec=library L$)$
$(<1>$ tmp=time T2$)$
(**id:** T2 $<$ T $+$ 3*week)
(**sit: DONE:** borrow
$(<*>$ ag=user U$)(<7->$ go=book B$)$ $(<1>$ src=library L$)$
$(<1>$ tmp=time T1$)$)

can be translated into:

$$\exists U'(\forall V(user(V) \leftrightarrow V \in U'))\wedge$$
$$\exists B'(card(B') \leq 7 \wedge \forall W \in B'(book(W)))\wedge$$
$$\exists L'(card(L') = 1 \wedge \forall X \in L'(library(X)))\wedge$$
$$\exists T1'(card(T1') = 1 \wedge \forall Y \in T1'(time(Y)))\wedge$$
$$\exists T2'(card(T2') = 1 \wedge \forall Z \in T2'(time(Z))) \wedge Z \leq Y + 3 * week\wedge$$
$$[< Y, borrow(V, W, X) >]\ (< Z, \mathcal{O}(return(V, W, X) >)$$

The temporal logic frame $< \ Time, Predicate \ >$ expresses the time (or time interval) in which the predicate is holding, whereas the dynamic logic construct $[\alpha]\beta$ expresses the fact that *after* the execution of action $\alpha$, action $\beta$ is triggered. The obligation of returning the book after the borrow-action is expressed by the deontic operator $\mathcal{O}$.

Raising a violation is quite different from getting an inconsistency. Suppose one had the specification:

**NEC: ACTION:** return
$(<*>$ ag=user U$)(<1>$ go=book B$)$ $(<1>$ rec=library L$)$
$(<1>$ tmp=time T2$)$

(**id:** T2 $<$ T + 3*week)
(**sit: DONE:** borrow
($< * >$ ag=user U)($< 7- >$ go=book B) ($< 1 >$ src=library L)
($< 1 >$ tmp=time T1)

This is translated into almost the same logical equivalent as stated above, but only the deontic operator $\mathcal{O}$ has been replaced by the modal operator $\Box$, which expresses necessity.

Suppose we find at some time $T3 > T + 3 * week$ some book being borrowed by a user from the library, but not returned. This would conflict with the above constraint leading to an inconsistency. The whole system is in trouble. Whereas, on the other hand, if a violation is raised because the deontic rule (MUST) is violated, nothing serious is at hand, except that another action is now activated (the sending of a reminder).

To demonstrate the use of logics, we show that there is a seeming inconsistency between boxes 1 and 2 in Figure 2. Suppose John is a person and the action in box 1 is applied to John, then John has a pass and is allowed to borrow a book according to box 2. However, in this box John is supposed to be a user. A theorem prover would not have any difficulties tracing such a shortcoming. In fact, what our Lexicon is doing is adding precisely this kind of information: a person who has got a pass is automatically considered as a user.

### 3.4 Code Generation

Finally, it is possible to generate the script for the library system, acting as an object who is responsible for the bookkeeping. The skeleton of such a script and some additional $< pseudo - code >$ to improve the understanding of the script, is shown below[2]:

```
/***** Generated Mokum Specification *****/
/*****          (skeleton)           *****/
/*****                               *****/
/*****      COLOR-X Codegen'96       *****/
/*****                               *****/
/*****       by: J.F.M. Burg         *****/
/*****                               *****/

type libraryT
    has_a borrowings:collection_of borrowT
    has_a books:collection_of bookT
    has_a users:collection_of userT
script
state initial:
    at_trigger start_library:
        < initialize me >
        next(active).
state active:
    at_trigger receive_borrow:
        < - delete borrow.book from me.books
                   if borrow.user in me.users
```

[2] The attribute (`has_a`) and specialization (`is_a`) information is retrieved from the additional COLOR-X Static Object Model [3]. `me` refers the object itself.

```
              - insert borrow into me.borrowings
              - create trigger: send_reminder
                        with trigger.time = now + 3 * week
                             trigger.book = borrow.book
                             trigger.user = borrow.user
            >
            next(active).
        at_trigger receive_return:
            < - delete borrow from me.borrowings
                        where borrow.book = return.book
              - delete trigger: send_reminder
                        where trigger.book = return.book
              - insert return.book into me.books
            >
            next(active).
        at_trigger send_reminder:
            < - send reminder to trigger.user
                        about trigger.book
            >
            next(active).
    endscript.

    type bookT
        has_a title:string
        has_a author:string

    type borrowT
        has_a user:userT
        has_a book:bookT
        has_a date:dateT.
```

## 4 What is responsibility?

Let us repeat what we said in the introduction about: *responsibility*; it means:

a) free to act or not to act according to the promises
b) able to explain ones deeds
c) in case of fault: penalties.

Let us give a few examples where the word *responsible* is being used:

1. In the CORBA-document [9] functions are defined for which *the object adapters are responsible* (p. 41)
2. Another example is the one in the section above about Alter Egos where we dealt with objects being keepers of collections of other objects, for whose integrity they are held responsible.
3. In the section about modelling: there is a person who is responsible for returning the book, but very easily one can replace this person/cybernaut by the Alter Ego it represents. Can we say that the Alter Ego can be held responsible for returning the book?

Let us analyze the three situations and compare them with the definition of *responsible*. For examples 1 and 2 it is evident that item b) can be realised for the examples: indeed, it is possible to program the adapter, the keeper and the Alter Ego in such a way that they can explain why they did what they did.

Items a) and c) are however rather difficult to imagine that a piece of software such as an adapter, keeper or Alter Ego can comply to. Either they are programmed (by a human programmer, who probably is the real one who is responsible) to act according to the promise, or commitment, which has been made, or not, but in the latter case they are programmed badly. We assume that that is not the case. A middle position could be that on purpose once in a while the adapter is not performing the function properly; in that case the programmer has not done the work properly (and should be punished). Also what does it mean that a piece of software should be punished?

So, we come to the conclusion that principally software cannot be held responsible, so Alter Egos cannot be held responsible. That means that it is not necessary in the world of Cyberspace to take into account the MUST modality, when Alter Egos are being modelled.

We shall see that that is a wrong conclusion.

Let us suppose that we are to design the script of the personal assistant, that is the specialization of the Alter Ego in charge to give a warning to the cybernaut when the borrowed book must be returned. Let us call it PA.

The first kind of behaviour of the PA would be just acting as the mailbox for its owner. The library system, let us call it L, sends messages to the user about returning the borrowed book, by sending it to the PA. In this case the PA's responsibility is very low. Actually, there is no responsibility, and if there were any this can be compared to the responsibility of a mail box: it should be open for messages and it should warn its owner.

In the next example we change the borrowing-a-book into borrowing-a-digital-book (or video, or cd). Because the book is digital there is no sense in the rule of returning it within three weeks. We assume therefore that the user has to pay a fee within three weeks.

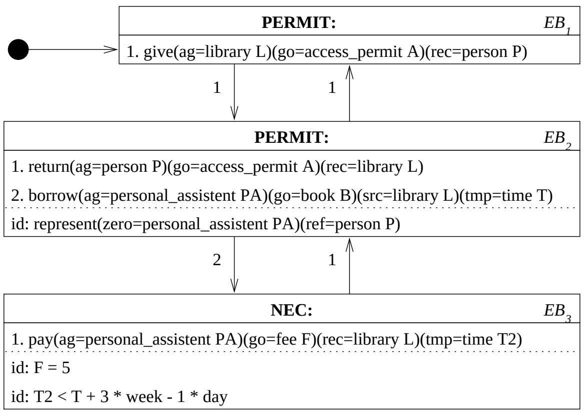


**Figure 5:** The Adapted COLOR-X Event Model for Borrowing Digital Books

In a diagram, very similar to Figure 2, this situation can be defined. The only change is that a fee has to be payed instead of to return the book. Also a similar rule can be chosen when the fee is not payed and a fine is to be payed. In the new situation there is more to be done by the PA. Upon the request of the student/cybernaut her PA asks the library for the particular book. Suppose it is available (even digital books can be non present!), suppose the student is entitled to borrow a book (remember that in the previous example she could have been blocked), then it gets the book from the library. The PA then just waits for three weeks (minus one day) and sends an order to the bank B, i.e. to its bank manager, to transfer the fee to L's account. Because this behaviour is fully defined we say that paying the fee is a NEC action. This situation is defined with the COLOR-X diagram shown in Figure 5. Because there is no real person anymore who can just choose to violate the commitment, there seems to be no reason to take such a situation into account.

However, the situation is more complex. It could very well be possible that the order for money transfer is not successfully carried out, for example because the account was not enough. So what then? The only thing which ultimately is left is that the PA goes to its owner and explains the difficulty. That means that in the end the owner herself has to take responsibility. The responsibility of the PA is only seemingly present: when unexpected problems occur it is the human being herself who is in charge and not the PA (Figure 6). The situation is very similar to the "Call the manager" rule in many organizations.

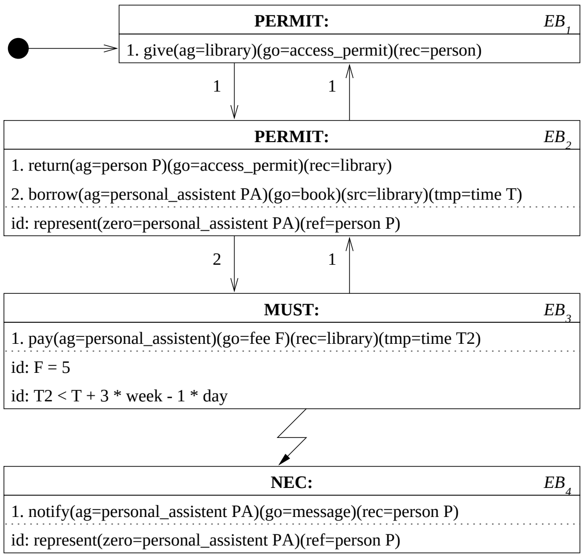


**Figure 6:** The Personal Assistant is still not Responsible

The final conclusion therefore is that although objects cannot take real responsibility we still have to seriously take into account that they do not do what has been promised.

## References


[1] J.F.M. Burg and R.P. van de Riet. COLOR-X Event Model: Integrated Specification of the Dynamics of Individual Objects. In M.P. Papazoglou, editor, *The Proceedings of the Fourteenth International Object-Oriented and Entity-Relationship Conference (OO-ER'95)*, Lecture Notes in Computer Science (1021), pages 146–157, Gold Coast, Australia, 1995. Springer-Verlag.

[2] J.F.M. Burg and R.P. van de Riet. COLOR-X: Linguistically-based Event Modeling: A General Approach to Dynamic Modeling. In J. Iivari, K. Lyytinen, and M. Rossi, editors, *The Proceedings of the Seventh International Conference on Advanced Information System Engineering (CAiSE'95)*, Lecture Notes in Computer Science (932), pages 26–39, Jyvaskyla, Finland, 1995. Springer-Verlag.

[3] J.F.M. Burg and R.P. van de Riet. COLOR-X: Object Modeling profits from Linguistics. In N.J.I. Mars, editor, *Towards Very Large Knowledge Bases: Knowledge Building & Knowledge Sharing (KB&KS'95)*, pages 204–214, Enschede, The Netherlands, 1995. IOS Press, Amsterdam.

[4] F. Dehne and R.P. van de Riet. A Guided Tour through Mokum 2.0. Technical Report IR-368, Vrije Universiteit, Amsterdam, 1994.

[5] S.C. Dik. *The Theory of Functional Grammar. Part I: The Structure of the Clause.* Floris Publications, Dordrecht, 1989.

[6] T. Erickson. The World-Wide Web as Social Hypertext. *Communications of the ACM*, 39(1):15–17, January 1996.

[7] G.A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11), November 1995.

[8] M. S. Olivier. Self-protecting Objects in a Secure Federated Database. In *Proceedings of the IFIP WG 11.3 9th working conference on Database Security*, pages 23–38, 1995.

[9] OMG. The Common Object Request Broker: Architecture and Specification. Technical Report 91.12.1, OMG, December 1991.

[10] R.P. van de Riet and J. Beukering. The Integration of Security and Integrity Constraints in Mokum. In J. Biskup, M. Morgenstern, and C.E. Landwehr, editors, *Database Security, VIII, Status and Prospects*, pages 223–246. IFIP, North-Holland, 1994.

[11] R.P. van de Riet, A. Junk, and E. Gudes. Security in Cyberspace: A Knowledge-Base Approach. Technical Report IR-398, Vrije Universiteit, Amsterdam, 1995.

[12] R.P.van de Riet and E. Gudes An Object-Oriented Database Architecture for providing High-level Security in Cyberspace to be presented at the IFIP WG11.3 Workshop on Security and Databases, Como, 1996.

[13] V. Varadharajan. Distributed Object System Security. In H.P. Eloff and S.H. von Solms, editors, *Information Security - the next Decade*, pages 305–321. Chapman & Hall, 1995.


## Acknowledgements

Co-author J.F.M. Burg has been supported by the Foundation for Computer Science in the Netherlands (SION) with financial support from the Dutch Organization for Scientific Research (NWO), project 612-123-309.