COLOR-$\mathcal{X}$: Linguistically-based Event Modeling: A General Approach to Dynamic Modeling

J.F.M. Burg* and R.P. van de Riet

Department of Computer Science
Vrije Universiteit
Amsterdam, The Netherlands
{jfmburg,vdriet}@cs.vu.nl

Abstract. This paper introduces a way of modeling the dynamic aspects of an Information and Communication System in which all the occurring events are listed and ordered in time. These graphical Event Models are based on formal (logical) specifications. Event Models are very close to the specifications in the informal requirements document, which describes the Universe of Discourse. By means of the underlying formal specifications Natural Language sentences are generated automatically, in order to give some feedback to the designer and user. By combining this feedback feature and the power of the logical foundation, the Event Models can be verified and validated. We will also present an algorithm and its implementation to generate State Transition Diagrams from Event Models automatically. This is especially useful in our environment in which programming code-generation is the key objective.

1 Introduction

The name of our current project, **COLOR-X**, is an acronym for the **CO**nceptual **L**inguistically based **O**bject oriented **R**epresentation Language for **I**nformation and **C**ommunication **S**ystems (**ICS** abbreviated to **X**). In the COLOR-X project we are using the logical conceptual modeling technique CPL (Conceptual Prototyping Language) [8], which is linguistically based, as a formal foundation for graphical modeling techniques. This approach is chosen to facilitate the process of conceptual modeling and which leads to more consistent and complete models that are linguistically correct. COLOR-X is the first phase of a larger project which has as objective the generation of object-oriented programming code from a natural language based modeling technique, which brings, as a side-effect, the conceptual models closer to programming code. In addition, by using a modeling technique based on linguistic notions, we are narrowing the gap between requirements documents, written in natural language, and conceptual models as well. The COLOR-X project is divided into several parts, analog to existing conceptual modeling methods, like OMT [20]). This paper contains the dynamic part,

* 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

whereas [4] describes the COLOR-X Static Object Model (CSOM), in which the static aspects of the Universe of Discourse (UoD) (i.e. objects, classes and the relations, like generalization and aggregation, between them) are contained. The graphical CSOMs are linguistically-based, and logically founded by underlying CPL-specifications. The CSOM-model contains the overall structure of the UoD for the programming code generator.

COLOR-X is part of the LIKE-project (Linguistic Instruments in Knowledge Engineering) which is a consortium of researchers of three disciplines: Linguistics, Business Administrators and Computer Science. The LIKE-project is focusing research around the theme: how linguistic instruments can be used profitably in the area of Knowledge Engineering, e.g. to build Information and Communication Systems (ICSs).

One of the main reasons to use linguistic knowledge is to make the use of words appearing in the models consistent, and thus making the models as a whole more meaningful. Earlier projects conducted in our group have shown the profitability of this approach, [3], [2], [4] and [5]. Another reason to use linguistic knowledge in modeling techniques is to give more expressive power to them. For example, it is now possible to express which events *should* and which *could* occur in a certain UoD. An additional nice feature of a linguistically based modeling technique is that it is relatively easy to generate natural language sentences from it, in order to give some feedback to the system designers and to the end-users as well, see also [7]. This feedback consists of generated sentences during the modeling phase, in order to check if the model is consistent with the requirements and on the other hand this feedback consists of explanation facilities, like [11]. The first kind of feedback is already incorporated into COLOR-X.

Now that we know *why* to use linguistic knowledge, we need to know *how* to use it. We will use a lexicon as a source containing this knowledge. Such a lexicon contains information about taxonomies, verb frames, synonym sets, etcetera. We are using (an extension of) WordNet [17], which is the result of an ongoing research program at Princeton University in the representation of lexical information.

The remainder of this paper is organized as follows: First we will give an overview of and remarks about the traditional way of dynamic modeling. After that we will offer an alternative by introducing COLOR-X Event Models (CEMs). The generality of this approach will be shown in section 5 and section 4.1 by generating State Transition Diagrams and formal CPL Specifications out of CEMs. We will conclude this paper by giving some conclusions and by listing work and research that is still to do.

2 Dynamic Modeling

The purpose of Dynamic Modeling is to show the time-dependent behaviour of the system as a whole or a particular part of the system. In general, there are three ways of modeling this information:

1. Dynamic and Deontic Logic, [8], interested in the states between actions

2. Process Algebra, [22] and [13], concentrating on the actions themselves
3. Petri Nets, which are useful in environments where simulation plays an important role [12], but which will not be discussed in this paper.

A popular example of a process algebra-based modeling technique is the graphical State Transition Diagramming (STD) technique, [20]. The use of STDs, however, causes some problems:

- It is not clear whether you should model one STD per object, one STD for the system as a whole or a mixture of these two approaches. Because of the lack of consensus concerning this point, it is very hard to parse or interpret STDs in computerized tools.
- The words used as transitions- and state-labels are not constrained by rules. The models would be more comprehensible if the *kind of words* used for actions, events and states would be pre-defined (like controlled verbs, non-controlled verbs and nouns, respectively). Another rule could constrain the *form of the words*, like infinitive verbs and singular nouns. Both kind of rules would facilitate the interpretation of the models and thus the generation of programming code out of them, but it is very hard to identify them and to establish some agreement about them.
- By adding different modalities (like necessary and possible) to actions and events, the resulting STD models would have more expressive power. In traditional STD techniques, just one modality (factual) is used.
- A state is not only defined by the attribute values of some object or (sub-) system, but also by common sense knowledge which is hard to capture in attributes (e.g. *"a person is ill"*).

An example of a dynamic and deontic logic-based modeling technique is the Conceptual Prototyping Language (CPL) [8]. The main problem with this approach is the awkward formal syntax, and the difficult underlying linguistic theory. To overcome these problems, we will propose Event Models in the next chapter. These models have CPL-specifications as an underlying formal representation. To understand these models properly we will first give a short introduction to CPL.

2.1 An Introduction to CPL

The Conceptual Prototyping Language (CPL) has been developed as a specification language as close as possible to natural language, by basing it on *Functional Grammar* [9], but formal enough to specify the requirements of an ICS in a precise and unambiguous way. The formal semantics, as defined in [8], is based on predicate, modal, deontic and temporal logic. Each CPL construct is translated into some combination of these logics. The general form of a CPL specification language is as follows:

Mode : Tense : Predication $T_1 \cdots T_n$ (**id:** $\cdots$) (**sit:** $\cdots$)

Mode	= **FACTUAL|MUST|NEC|PERMIT**
Tense	= **ACTION|DONE|PROSP|PERF|PRET**
Predication	= a relation between n terms $T_1 \cdots T_n$,
T_i	= a term denotes a (set, with cardinality c, of) object(s). Each object occurs in a specific role.
id	= identification of the objects
sit	= situation in which this CPL specification is supposed to hold

For example, the following specification says that
When a company has sold a car to a customer, it has to send a bill to this customer within a week.:

MUST: ACTION:
send(**ag**=C in company) (**pat**=bill) (**dest**=C2 in customer) (**temp**=T2 in time)
(**id:** T2 = T1 + 1*week)
(**sit: PERF:** sell(**ag**=C in company) (**pat**=car) (**dest**=C2 in customer)
(**temp**=T1 in time))

The meaning of the used modalities (**MUST** means 'should'), tenses (**ACTION** means 'present tense' and **PERF** means 'perfect tense') and semantic functions (**ag, go, pat, dest** are the agent, goal, patient and destination of the event and **temp** is the time at which the event takes place) can be found in [8].

3 COLOR-X Event Modeling

The COLOR-X Event Model (CEM) is merely a trace of the events that could and should be performed in the Universe of Discourse (UoD). This way of modeling the dynamic aspects of the UoD links up very well with the way these aspects are described in the requirements document. There is however no automatic acquisition of conceptual models out of these natural language sentences provided yet, as [1] and [19] propose. The following example will illustrate this correspondence:

Requirements Document: *A user can borrow a book from a library. If a user has borrowed a book he has to return it within three weeks, before he is allowed to borrow a book again.*

COLOR-X Event Model (CEM): Figure 1 shows the corresponding CEM. It is fairly easy to read and corresponds very closely to the natural language sentences.

Informally, a box represents an event that could, should or has to take place (depending on the modality), a straight arrow represents the actual occurrence of that event and a 'lightning'-arrow represents the fact that the specific event did not take place.

A strong point of CEMs is the possibility to express the modality of the sentences. The occurrences of the words *'can'* and *'is allowed to'* in the requirements document trigger a **PERMIT**-box. The **MUST**-box is caused by the words *'has to'*. As will be shown further on in section 3.1 when we will treat the

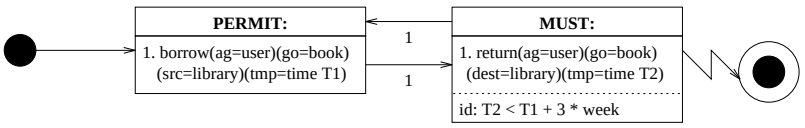

Fig. 1. Example of a COLOR-X Event Model

syntax and semantics of CEMs, a **MUST**-event requires two outgoing arrows to succeeding events: the obligatory event has taken place (as it should be) or the obligation is violated. Because of the fact that in our simple example there is no event specified that has to be done when the book is not returned within three weeks, the outgoing ('lightning'-) arrow ends in an end-node.

3.1 Syntax and Semantics of CEMs

The graphical notations of CEMS can be found in Figure 2. An event box, Figure 2(a), consists of a modality, one ore more event descriptions and zero or more constraint descriptions. An event description consists of a verb denoting an event, which is either an action (an event controlled by some agent) or a (not controlled) process, and one or more terms. The (CPL-) syntax of these terms is: *[<cardinality>] role = [variable in] noun* [2]. The components of a term were already mentioned in section 2.1. An example clarifies this abstract formulation:

one user borrows four books $\Leftrightarrow$ borrow(<1> ag = user) (<4> go = book)

This formal event representation expresses exactly what the modeler wants, which is not always true when using ambiguous natural language sentences. Another advantage of this approach is that it is now possible to use automatic tools to support the modeling process. It is always possible to generate natural language sentences automatically out of the CPL constructs.

A constraint description constrains the value of one or more terms (through the use of variables) absolutely (*age > 21*) or relatively (*age father > age son*). The syntax used to express these constraints is the same as the one used in CPL: (**id:** $V_1 > 21$) and (**id:** $V_1 > V_2$).

Besides the event-nodes there are two special nodes that denote start and final points (Figure 2(b) and (c), respectively).

Because of the fact that there are three modalities to use (permit, necessary and must), there are three different kinds of event-boxes (Figure 2(d) - (f)). In this way a certain degree of completeness is accomplished. When a **MUST**-box is used there are always two succeeding events to be specified. After finishing the model the remark "the model does not specify what has to be done when event X has not taken place" will not occur! One has always to specify a relative or absolute *expiration-time*, which may be infinite, when using a **MUST**-box, in order to verify whether the obligation has been violated or not. The event-boxes

[2] everything between square brackets ([..]) is optional

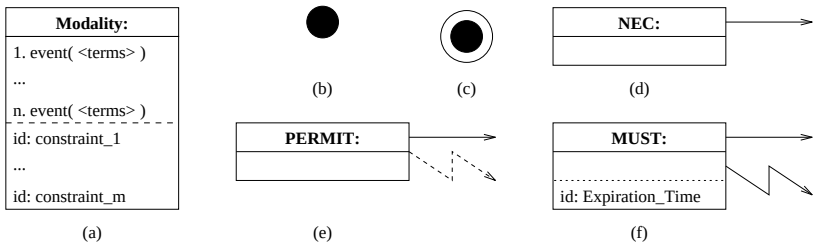

Fig. 2. CEM Notation, (a) general event, (b) start node, (c) final node, (d) necessary, (e) permitted, (f) obligatory

are connected with arrows which denote the fact that one or more events are performed (depicted by a straight arrow with one or more event-numbers) or are not performed at all (depicted by a lightning-arrow).

Creating CEMs: Almost every current conceptual modeling method contains some step in which the events occurring in the UoD are listed, see for example OMTs *event traces* [20]. CEMs do not only contain this kind of information, but also formalize it. The process of creating CEMs is supported with a lexicon. Although the initial step, listing the events and ordering them in time in an informal way, should be done manually by the modeler, the creation of the CEM itself is embedded, and thus supported, by a CASE-environment. The availability of standard building blocks, the reusable event specifications from a lexicon and the complementary information, like *antonym-events* that will be treated later on, generated out of the lexicon, will help the modeler very much in creating correct and complete CEMs.

4 Correct and Complete CEMs

In this section we will give an overview of the advantages yielded by our approach in which a lexicon plays an important role. After modeling a certain UoD, regardless the method used, there remain always two questions:

Correctness, i.e. Is this model right? Is the model constructed according to the syntax and semantics of the method used? By offering standard building blocks, see Figure 2, the resulting model could not be offending the graphical syntax rules. By checking if there exists exactly one start and final state, and that every arrow goes from one block into another, we can verify if the model is syntactically correct. The *kind* and *form* of the words used in the model are constrained by the use of a lexicon. The following information is retrieved from the lexicon in order to get the kind and the form of the words right, respectively.

1. An *event* is identified by a verb, and one or more nouns, that play certain roles. We retrieve the *verb frame* corresponding to the verb from the lexicon and check if the entered role-playing nouns fit into this frame. For example:

Verb and Nouns: borrow, user and book
Verb Frame: somebody borrows something [3]
Match: user *is a* somebody [3] , book *is a* something [3]

2. When entering nouns in the plural form, it is very easy to obtain the singular form from the lexicon, in order to further standardize the model.

Completeness, i.e. Is this the right model? Does the model contain all the information from the requirements document? To verify if the model corresponds to the text from the requirements document it is very helpful to generate a verbalized form of the model, see also [3], [18] and [7]. This is made possible in CEMs because the underlying CPL specification can be verbalized. Another heuristic to verify if a CEM contains all the information from the requirements document is to generate the *antonym-events*, which can be found in the lexicon, of all the events occurring in the CEM and to check if they appear in this CEM already. In the library-example, section 6, the *free*-event was generated as antonym of the *block*-event and added to the CEM. The antonym-event of *borrow* (i.e. *return*), however, is already appearing in the model. The next two sections will show the CPL- and Natural Language generators.

4.1 Generating CPL Specifications

The generation of CPL-specifications from CEMs is fairly easy, because CPL is used as a foundation for CEM. We will review all the concepts used in CEMs and give their CPL counterparts, as our demo-tool CEM2CPL generates:

1. CEMs start- and end-node and their in- and outgoing arrows have no CPL equivalent
2. **CEM:** general event box with modality *Modality*, events $event_1 \cdots event_l$ and corresponding terms $term_{i_1} \cdots term_{i_{p_i}}, 1 \leq i \leq l$, constraints $c_1 \cdots c_k$ and outgoing arrows $a_1 \cdots a_h, 1 \leq h \leq l$
   CPL: for each arrow a_j, $1 \leq j \leq h$, with label $n\& \cdots \& m$, $1 \leq n, m \leq l$:
   Modality: $event_n($ $term_n \cdots term_{n_{p_n}}$ $)$
   and $\cdots$ **and**
   Modality: $event_m($ $term_m \cdots term_{m_{p_m}}$ $)$
   All the CPL-blocks belonging to a certain arrow are OR-ed together (disjunctive normal form).
   All the constraints $c_1 \cdots c_k$ are AND-ed together (disjunctions between constraints should be expressed as one constraint).
   (**id:** c_1) **and** $\cdots$ **and** (**id:** c_k)
   If a 'lightning'-arrow is appearing in the model, the negation of the event(s) will appear in the CPL-specification.
3. **CEM:** a certain event box EB_i with all its predecessors ($EB_1 \cdots EB_{i-1}$, without their modalities and tenses). Each $EB_j, 1 \leq j \leq i$ contains events $E_{j_1} \cdots E_{j_l}$

[3] Retrieved from WordNet

CPL: Each $EB_j, 1 \leq j \leq i$, is translated into a CPL-disjunction CPL_j according to step 2. Combining each CPL_j will result in:
CPL_i
(**sit: DONE:** CPL_{i-1})
(**sit: PERF:** CPL_{i-2}) $\cdots$ (**sit: PERF:** CPL_1)
For $i = 2$ the CPL specification looks like: CPL_2 (**sit: DONE:** CPL_1)

There are four reasons why we would like to generate CPL-specifications:

1. It is possible to generate Natural Language (NL) sentences out of CPL specifications. Because CPL exists for several years now, we have got some CPL-parsing and NL-generation tools already (section 4.2).
2. Because CPL is logically founded, see [8], it is possible to formally derive new specifications out of existing ones and to check if the specifications are correct. We will not treat the logical foundation of CPL in this paper.
3. CPL supplies formal semantics for the dynamic, as well as the static, aspects of a UoD and its related database, which restricts the behaviour of the *generated* computer programs exactly to the behaviour modeled.
4. By using CPL as the underlying specification language for all kinds of COLOR-X models, we have a uniform way of representing different kinds of information. This uniform format facilitates the integration of the different views on a UoD and makes updates and queries on the integrated information more manageable.

4.2 Generating Natural Language

Our Prolog-translator CPL2NL translates any form of CPL-specifications into correct Natural Language sentences. In this translation process the lexicon plays a very important role, because it contains (information about) verb derivations, plural and singular form of nouns, numerals, adjectives, determiners, etc. We will list some aspects of the CPL specifications that have their impact on the generated sentences. First, the modality determines the auxiliary verb of the sentence as follows: **NEC**, **MUST** and **PERMIT** trigger *obliged to*, *should* and *permitted to* respectively. Secondly, the cardinality of the subject (agent or zero) of the relationship determines the singular or plural form of the related verb. The identification of the objects is added as a subordinate clause, starting with *where*.... Finally, the satellites of the CPL specification are translated into adjuncts of place or time.

There are three basic forms of CPL-specifications: [4]

1. *Unconditional:*
   PERMIT:ACTION: borrow(ag=user)(<+>go=book)(<1>src=library)

   `[an,user,is,permitted to,borrow,one or more,books,from,a,library]`

[4] The consonant sound of *user* is not noticed because we do not use a phonetical analyzer. Therefore, the article *an* is generated instead of *a*.

2. *Conditional:*
   MUST:PROSP: return(ag=user)(<+>go=book)(<1>dest=library)
   (**sit: PERF:** borrow(ag=user)(<+>go=book)(<1>src=library))

   `[if,an,user,borrowed,one or more,books,from,a,library,`
   `then,an,user,will have to,return,one or more,books,to,a,library]`

3. *Identified:*
   PERF: borrow(ag=user)(go=book)(tmp=V1 in time)(id: V1 = yesterday)

   `[an,user,borrowed,a,book,at,a,time,V1,where,V1,is,yesterday]`

5 Generating STDs

There are mainly two reasons to generate State Transitions Diagrams (STDs) out of COLOR-X Event Models:

1. STDs have become a standard (to a certain degree) in modeling the dynamic aspects of an ICS. Although we have had some difficulties and problems using STDs, see section 2, we have shown that our CEMs contain also the information normally found in STDs by generating STDs out of CEMs. The reverse process is only possible if the STD is not violating the STD-rules that are mentioned below.
2. A lot of research in the field of programming code generation from STDs is already done, and most of the existing CASE-tools support such a generation facility, see for example [21]. We can gain from this knowledge and experience by generating STDs as intermediate results.

The generated STDs satisfy the following rules:

- Every STD belongs to exactly *one* active object occurring in the UoD. Active objects are nouns that play the *agent*-role in one or more CEM-events.
- A state is represented by a box, identified by a unique number. A verbal label can be added manually, but has no semantic meaning in the model. This decision is made because it is really hard to find meaningful labels for every state, and to maintain a certain degree of conformity among the labels.
- A transition is represented by a uni-directional arrow labeled with a verbal phrase, describing the event that causes the state-transition. Constraints can be attached to a transition as an optional component of it.
- There exist two special states: the *start*-state and the *final*-state, that are connected with the first 'real' state (i.e. the state before the first event) and the last state, respectively, by empty transitions. These states correspond to the creation and destructions, respectively, of the object.

The next section will describe the algorithm that generates STDs for each active object out of CEMs.

5.1 Algorithm

The following steps describe roughly the STD-generation process, which has as an input a COLOR-X Event Model, consults a lexicon, and has as an output STDs for each active object of the CEM:

1. create the start-node (N_s), the first node (N_1) and an empty transition ($T_\in$) between them
2. for each event box (EB_i) with events $E_{i_1} \cdots E_{i_l}$: create for each outgoing arrow (with label $n\& \cdots \& m$, $1 \leq n, m \leq l$) a state transition to a new or an existing node (depending on whether the succeeding event box (EB_j) is already traversed ($j < i$) or not ($j > i$)). The label attached to this transition is a conjunction of the *verbalized* event descriptions of each E_{i_k}, $1 \leq k \leq l$. The verbal phrase describing event E_{i_k} are adjusted as follows: it is stripped from its modality and tense, and
   - if the object, described by the STD, is the agent of the CEM-event, the CEM-event is copied into the transition-label
   - otherwise, the sentence is transformed into a new one in which the object is the linguistic subject of it. To achieve this the perspective antonym of the verb describing the event is retrieved from the lexicon. This new sentence becomes the transition-label. E.g. the transition label in the library-STD corresponding to the CEM-event *"a user borrows a book from a library"* will become *"a library lends a book to a user"*.

   If the CEM-arrow is a 'lightning'-arrow the negation of the verbal phrase is attached to the STD-transition.
   The (optional) constraint descriptions of event box EB_i is attached to the constraint part of the transition.
3. create the final node (N_f), and an empty transition ($T_\in$) from the last state (N_n) to it

Implementation We have implemented the algorithm as described in the previous section. The resulting Prolog-translator CEM2STD reads in a CEM and generates for each active object occurring in the CEM a corresponding STD-description. These STD-descriptions are translatable into internal representations of several tools. One of these tools is the CASE-tool *Software through Pictures (StP)*, another one is of course the code-generator of our overall project.

5.2 Related Work

In [15] the inference of state machines out of OMT trace diagrams [20] is described. The main difference between their approach and ours is the fact that we use a formalized input, whereas their OMT trace diagrams are informal. The advantages we gain out of this difference are the natural language generation facility, the possibility to use other kinds of (logical) inferences (also using the static information, which has the same logical foundation [4]) and the syntactical and semantical verification of the models. Another difference is that we

are generating STDs as an intermediate result to generate programming code, and [15] are incorporating their method into an environment which supports the conceptual modeling process using the OMT methodology.

6 Example

To visualize the techniques, algorithms and tools described in this paper, we will present an example. This example consists of the simplified library book circulation system.

Library: Requirements Document: *The library gives passes to persons that want to become users of the library. If a person does not want to be a user any more, he returns his pass. A user can borrow a book for three weeks. At the end of the allowed lending period, the user should return the book. If a user does not return a book action is taken, by sending him a reminder. If one week after the reminder is sent there is no message from the user, he must pay a fine of Dfl 70 and is blocked for borrowing any more books until the book is returned and the fine is paid.*

Library: COLOR-X Event Model: Figure 3 shows the COLOR-X Event Model corresponding to the requirements stated in the previous chapter, which is *syntactically correct*, i.e. the model is right, and it is semantically correct, i.e. the right model, according to the rules, stated in section 4.

Library: CPL and NL: The CEM2CPL tool has generated all the CPL-specifications, which we will not show here. Some corresponding NL-sentences (generated by CPL2NL) are listed below:

```
1.  [a,library,is,permitted to,give,a,pass,to,an,user]
2.  [if,a,library,gave,a,pass,to,an,user,
     then,an,user,is,permitted to,borrow,a,book,from,a,library]
4.  [if,a,library,blocked,an,user,
     and,an,user,did not return,a,book,to,a,library,
     and,a,library,sent,a,reminder,to,an,user,
     then,an,user,should,return,a,book,to,a,library,at,time,T4,
     and,an,user,should,pay,a,fine,F,to,a,library,at,time,T4,
     where,F,is,70,and,where,T4,is,infinite]
```

Library: State Transition Diagram: We haven chosen to let the CEM2STD-tool generate a State Transition Diagram for the active object *Library*. This has led to sentences at the transition-labels in which the library is the subject, see Figure 3 (e.g. *borrow* has become *lend*).

7 Conclusions and Further Research

This paper shows an approach to dynamic modeling (COLOR-X Event Modeling, CEM), the result of which is very close to the original natural language sentences that describe the Universe of Discourse. By facilitating the modeling

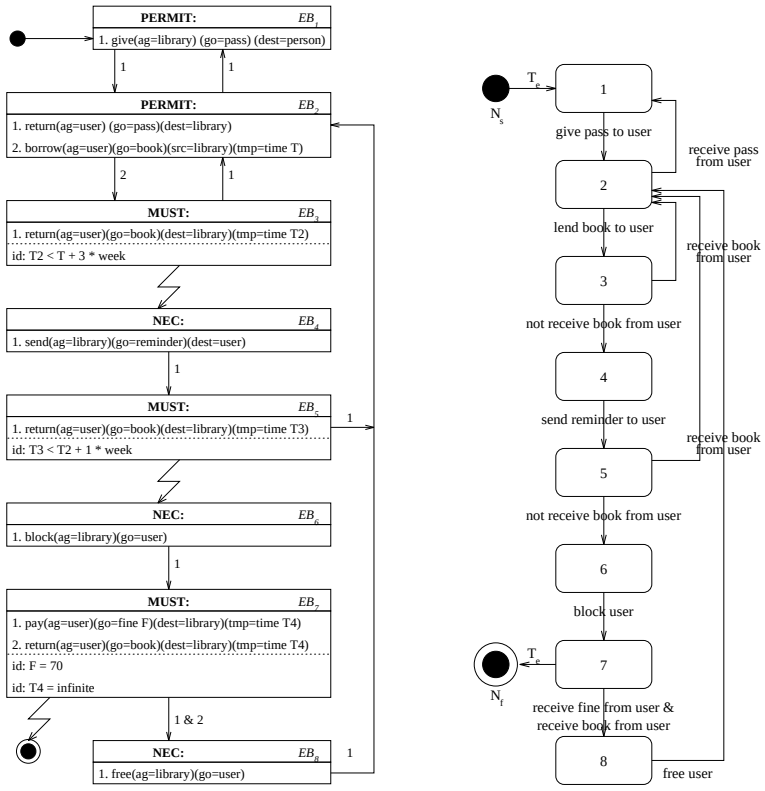

Fig. 3. The CEM and library-STD of a Library Book Circulation System

process itself, by means of a lexicon and offering standard building blocks, the resulting models will tend to be correct and complete. By generating natural language sentences out of CEMs the correspondence with the requirements document can be verified. A nice feature of CEMs is the fact that for each object a State Transition Diagram can be generated, which gives in turn very useful information (object states, state transitions and causes for those transitions) for a programming code generator. A similar project which focuses on Jacksons Entity Structure Diagrams [14] is currently carried out. We are also comparing CEMs with process graphs [10]. All these steps are meant to narrowing the gap between problem specification and implementation.

The resulting dynamic event model is one way of viewing the Universe of Discourse. To get a complete view of the UoD, we have already defined the COLOR-X Static Object Model (CSOM), [4], that describes the static aspects of the UoD in a linguistically-based graphical way, that links up closely with the object model of the OMT-method, [20]. We have also defined a logical foundation of CSOM by giving a translation to CPL-facts. The CSOM-model contains the overall structure of the UoD for the programming code generator.

After finishing the COLOR-X project we will gain advantages in the fields of programming code generation, reusable models (also by using a lexicon, which can be regarded as a repository of reusable relationships and objects) and software and feedback facilities.

With respect to the CEMs the following aspects are still researched:

- The addition of more semantic information to the arrows. For now, we have just one kind of arrow: the conditional one (if $event_i$ has taken place then $event_j$ could/should/has to take place). Other kinds of (rhetorical) relationships could include causal, resulting and concurrent relations [16], [11].
- The relations between the dynamic CEM-models and the static CSOM-models. The events from CEM will have their impact on the relations and objects of CSOM. The kind of impact will also be described by means of rhetorical relations.
- We are still analyzing some aspects of State Transition Diagram- Techniques, like triggered operations and nested diagrams, that are not expressable in CEM-models, yet.

We are still working on the kind of information a lexicon should contain to be useful in the construction process of an Information and Communication System. We have carried out some previous projects, in which a lexicon was used in a data-dictionary environment [6], to interpret ER-diagrams [3], and a general feasibility study to use linguistic knowledge during the conceptual modeling process [2].

The tools described in this paper are still preliminary demos, although fully functional. We are (re-) implementing these tools in a more efficient way into a coherent environment at the moment. This should lead to a CASE-environment in which a Lexicon Management System plays an important role. The overall idea is to support the modeling process, according to the COLOR-X method, with linguistic knowledge and tools in order to generate correct programming code easily.

References

1. W.J. Black. Acquisition of conceptual data models from natural language descriptions. In *Proceedings of the 2nd Conference of the European Chapter of the ACL*, Copenhagen, 1987.
2. P. Buitelaar and R.P. van de Riet. A feasibility study in linguistically motivated object-oriented conceptual design of information systems. Technical Report IR-293, Vrije Universiteit, Amsterdam, 1992.
3. P. Buitelaar and R.P. van de Riet. The use of a lexicon to interpret er diagrams: a like project. In *Proceedings of the ER Conference*, Karlsruhe, 1992.
4. J.F.M. Burg and R.P. van de Riet. Color-x: Object modeling profits from linguistics. Technical Report IR-365, Vrije Universiteit, Amsterdam, 1994.
5. J.F.M. Burg and R.P. van de Riet. Color-x: Object modeling profits from linguistics. 1995. To appear in the Proceedings of the KB&KS'95, the Second International Conference on Building and Sharing of Very Large-Scale Knowledge Bases, Enschede, The Netherlands.

6. J.F.M. Burg, R.P. van de Riet, and S.C. Chang. A data-dictionary as a lexicon: An applicication of linguistics in information systems. In B.Bhargava, T.Finin, and Y.Yesha, editors, *Proceedings of the 2nd International Conference on Information and Knowledge Management*, pages 114–123, 1993.
7. H. Dalianis. A method for validating a conceptual model by natural language discourse generation. *Proceedings of the 4th International Conference on Advanced Information Systems Engineering*, 1992.
8. F.P.M. Dignum. *A Language for Modelling Knowledge Bases. Based on Linguistics, Founded in Logic.* PhD thesis, Vrije Universiteit, Amsterdam, 1989.
9. S.C. Dik. *The Theory of Functional Grammar. Part I: The Structure of the Clause.* Floris Publications, Dordrecht, 1989.
10. R.B. Feenstra and R.J. Wieringa. Lcm 3.0: A language for describing conceptual models – syntax definition. Technical Report IR-344, Vrije Universiteit, Amsterdam, 1993.
11. J.A. Gulla. *Deep Explanation Generation in Conceptual Modeling Environments.* PhD thesis, University of Trondheim, Trondheim, 1993.
12. K.M van Hee, L.J. Somers, and M. Voorhoeve. Executable specifications for distributed information systems. In E.D. Falkenberg and P. Lindgreen, editors, *Information System Concepts: An In-depth Analysis*, pages 139–156. North-Holland/IFIP, Amsterdam, 1989.
13. A.H.M. ter Hofstede. *Information Modelling in Data Intensive Domains.* PhD thesis, Katholieke Universiteit Nijmegen, Nijmegen, 1993.
14. M Jackson. *System Development.* Prentice-Hall, 1983.
15. K. Koskimies and E. Makinen. Inferring state machines from trace diagrams. Technical Report A-1993-3, University of Tampere, 1993.
16. W.C. Mann and S.A. Thompson. Rhetorical structure theory: Description and construction of text structures. In G. Kempen, editor, *Natural Language Generation: New Results in Artificial Intelligence, Psychology and Linguistics*, pages 85–95. Martinus Nijhoff Publishers, 1987.
17. G.A. Miller, R. Beckwith, C. Fellbaum, D. Gross, K. Miller, and R. Tengi. Five papers on wordnet. Technical report, Cognitive Science Laboratory, Princeton University, 1993.
18. G.M. Nijssen and T.A. Halpin. *Conceptual Schema and Relational Database Design : A Fact Oriented Approach.* Prentice Hall, 1989.
19. C. Rolland and C. Proix. A natural language approach for requirements engineering. In P. Loucopoulos, editor, *Proceedings of the 4th International Conference on Advanced Information Systems Engineering.* Springer-Verlag, Manchester, 1992.
20. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice-Hall International, Inc., Englewood Cliffs, New Yersey, 1991.
21. A.I. Wasserman and P.A. Pirchner. A graphical extensible integrated environment for software development. In P. Henderson, editor, *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pages 131–142. ACM, ACM Press, March 1986.
22. R.J. Wieringa. *Algebraic Foundations for Dynamic Conceptual Models.* PhD thesis, Vrije Universiteit, Amsterdam, 1990.

This article was processed using the LaTeX macro package with LLNCS style