Modal Logic as a Design Notation

Areces, Carlos Felder, Miguel Hirsch, Dan Yankelevich, Daniel

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Argentina

e-mail: careces@dcs.warwick.ac.uk, felder@is.uba.ar, dhirsch@dc.uba.ar, dany@se.uba.ar

September 12, 1998

Abstract

A notation to describe software system designs is given together with the means to verify properties over them. Design graphs are considered as models of an extended modal logic. The procedure to derive the modal model associated with a design, the algorithm to check properties over a model, the method to define new relations and the method of model filtration are presented. These methods are introduced using the classic example of KWIC (Key Word in Context) and further tested on two different design systems to show their usefulness and generality. The logic proposed is called **KPI** (a poly-modal logic with inverse operators) and will be used as a property specification language verified through an algorithm of model checking. The methods proposed are effective and simple to implement.

1 Introduction

The successful development of software systems requires adequate tools. The task of software engineering environments is to provide tools that give the engineer a deeper understanding of the system under design. These tools have to be at the same time powerful, simple and as formal as necessary (or possible.)

There are different tools to be used in the different steps of the Software Development Cycle. In our work, we will devote special attention to the Design Step.

Design. After obtaining a specification of the problem under consideration (in a more or less formal language), the general characteristics defining the components of the system, the various inter-relationships among them, the properties or restrictions that the components should satisfy, etc., have to be stated. These general lines define the *Design* of the solution of the problem.

This work was partially supported by UBACyT EX-186 and ECC KIT#125 projects. Areces is under a grant from the British Council (#ARG0100049.)

Structured Design was the first technique reflecting and handling the complexity of modern systems. To this end, it provided a small set of tools for the semi-formal description of the problem. The use of standard tools like Data Flow Graphs, Structure Charts, etc. eliminated the *variety*, but they did not exclude the *ambiguity*, and the formal proof of properties was still impossible [GJM91]. In other words, the techniques of Structured Design established a standard language for the description of designs and a graphic language was selected as the best option, but it still lacked a *formal semantics* [Per87].

Many researchers contributed to the solution of various aspects of this problem. The work of [CMR92], for example, has as a goal to provide tools to obtain and manipulate simplified visions from the design aiming to a better understanding of the system. Other examples of projects in this line can be those of [ARP95] (specification of high level logic programs using operations on sets, through a graphic interface) [PP95] (a query algebra to reconstruct the design from the code of a system) and the more traditional approach of [MSV84] (given a high level specification using ADTs, a scheme of module composition and refinement is proposed, through an extended first order logic.)

However, in our opinion a jump towards a higher level of abstraction is needed. In addition to a technique that provides simplified visions of the system, we want to be able to write down properties in a formal language and reason over the design graphs abstractly. We think that this high level of abstraction is obtained when we consider designs as modal structures (Kripke models) and that the use of a modal language gives us the expressive power we need to formalize and verify properties of design.

The use of modal logics in verification is not an original idea in the area of Computer Science. Different logics have been proposed to deal with the verification of algorithms, specially concurrent algorithms (PTL, Propositional Temporal Logic [Pnu77]; TLA, Temporal Logic of Actions [Lam94]; CTL, Computational Tree Logic [CES86], etc.)

The original approach in this article is to carry this idea one step further and use descriptive logics (like modal logics) even during the Design stage. There have been already a few examples of modal logics as a tool for software design like [BT93, BT96] where a modal language is used to describe system configurations.

The formalisms for algorithm verification can be classified into two families: *proof theoretic* and *model checking*. The first approach is based on a syntactic characterization of the program while the techniques used by the second are mainly semantic. Model checking is the verification of the validity of a certain formula in a specific model using specific algorithms (the model checkers.) The latest advances in algorithm verification have been developed using semantic techniques. The main problem this area must confront is the combinatorial explosion in the number of states during the analysis of concurrent programs [HNSY94]. New techniques and optimized algorithms let now verify properties over systems of millions of states, and this upper limit is more than enough for our purposes: the size of a design will never exceed this magnitude.

Having in mind this predominance, we defined in Section 2 the logic system **KPI** that will be used as a property specification language. The design graphs are considered as models of the logic and the procedure to derive the modal model associated with a design, an algorithm of model checking, the method to define new relations and the method of model filtration are introduced, via a classic design example, in Section 3. Finally, these methods are applied to two different designs in Section 4 and we draw our conclusions in Section 5.

The methods proposed are effective and simple to implement. A prototype tool has been developed in SML-NJ covering all the concepts and functionalities described in the paper [AH96b]. Its description is out of the scope of this work.

2 Poly-modal Logic with Inverse Operators

Classically, modal logic is understood as the logic of the modal operators $\Box$ and $\Diamond$. However, for the purposes of our work, the expressive power of these logics is not enough and we will have to move to more expressive systems.

The first extension is motivated by the kind of models on which we base our work. Designers use many different relations to describe the interaction among components of a design. Thus, we need different modalities $[a]$, one for each relation $\multimap a \rightarrow$ in the design.

Another extension is needed given that when a module m_1 is related to a module m_2 it is always understood that m_2 is also related (but perhaps in a different way) to m_1. For example, if m_1 *uses* m_2 then m_2 *is used by* m_1. To cover this we will consider *inverse modalities* $[a]_i$ as is done in temporal logic. These modalities have as associated relation the converse of the relation associated to $[a]$.

We will present now the poly-modal logic with inverse operators **KPI** that adds both extensions. The basic notions from classical modal logic are assumed as known.

Definition 2.1 ($\mathbf{KPI}_L$) *Given L a finite set of labels, the system of poly-modal logic with inverse operators $\mathbf{KPI}_L$ is defined as follows:*

1. *If φ is a tautology or a substitution instance of a tautology then φ is in $\mathbf{KPI}_L$.*
2. *$\mathbf{KPI}_L$ has all substitution instances of the formulas*

   Ka. $[a](\varphi \rightarrow \psi) \rightarrow ([a]\varphi \rightarrow [a]\psi)$, *for any* $a \in L$.

   Ka$_i$. $[a]_i(\varphi \rightarrow \psi) \rightarrow ([a]_i\varphi \rightarrow [a]_i\psi)$, *for any* $a \in L$.

   It1. $\varphi \rightarrow [a]\langle a\rangle_i\varphi$, *for any* $a \in L$.

   It2. $\varphi \rightarrow [a]_i\langle a\rangle\varphi$, *for any* $a \in L$.

3. *$\mathbf{KPI}_L$ is closed under the following derivation rules*

   MP. *If φ, $\varphi \rightarrow \psi$ are in $\mathbf{KPI}_L$ then ψ is in $\mathbf{KPI}_L$. (Modus Ponens.)*

   Na. *If φ is in $\mathbf{KPI}_L$ then $[a]\varphi$, $[a]_i\varphi$ are in $\mathbf{KPI}_L$, for any $a \in L$. (Necessitation.)*

Definition 2.2 (Kripke Model) *A Kripke model for $\mathbf{KPI}_L$ is a tuple $\mathfrak{M} = \langle W, \{\multimap a \rightarrow \mid a \in L\}, v\rangle$ where W is a non empty set (of "worlds"), each $\multimap a \rightarrow$ is a binary relation on W and v is a modal valuations mapping propositional symbols to the sets of worlds where the symbols hold ($v : I\!P \mapsto \mathcal{P}(W)$.)*

Definition 2.3 (Validity in Possible Worlds) *Given a model $\mathfrak{M} = \langle W, \{\multimap a \rightarrow \mid a \in L\}, v\rangle$ and $m \in W$, the notion of validity in possible worlds is defined recursively as:*

1. *For $\top$, p_i (a propositional symbol) , $\neg$ and $\vee$ as usual.*

2. $\mathfrak{M} \models_m [a]\varphi$ *iff for all* $m' \in W$, $m -a\rightarrow m'$ *implies* $\mathfrak{M} \models_{m'} \varphi$.

3. $\mathfrak{M} \models_m [a]_i\varphi$ *iff for all* $m' \in W$, $m' -a\rightarrow m$ *implies* $\mathfrak{M} \models_{m'} \varphi$.

We say that a formula φ is valid in a model $\mathfrak{M}$ ($\mathfrak{M} \models \varphi$) iff φ is valid in all the worlds in the model. We say that a formula φ is valid in a class of models C ($\models_C \varphi$) iff it is valid in all the models in the class.

The result that claims that the system **KPI** is adequate (correct and complete) and decidable in the class of all the models follows from the results of adequateness and decidability of the minimal poly-modal logic and the minimal temporal logic plus preservation results.

Theorem 2.4 (Soundness, Completeness and Decidability) KPI *is adequate and decidable in the class of all models.*

3 Modal Logic as a Design Notation

In this section we will work on the example of the KWIC (Key Word in Context) to show how the new techniques are used. This problem was introduced in [Par72] and is a classical example in software engineering courses.

The indexing system KWIC accepts an ordered set of lines, where each line is an ordered set of words, and each word is an ordered set of characters. A circular shift can be applied to any line repeatedly removing the first word of the line and putting it at the end. The indexing system KWIC returns as result the alphabetically ordered set of all the circular shifts of all the lines. Various module decompositions have been proposed for this problem, [GS93]; in this section we will use a decomposition based on abstract data types (ADTs.)

The decomposition consists of a principal control component, and other five principal components that in turn are divided into submodules through which the data types will be accessed: Input (reads the lines from the input media), Characters (provides functions to manipulate characters in words), Circular_Shift (gives functions to work over any possible shift of the input lines), Alphabetic_Shift (Alphabetizer orders the lines and I-th returns the index of the circular shift in the i-th position in the order) and Output (outputs the set of circular shifts of the lines.) There are three basic relations between the components: *is_part_of*, *invokes* and *I/O*.

One of the most natural ways to describe a design is through a graph in which a symbol (generally a box or a rectangle) represents components, and lines connecting them represent relations, as in Figure 1.

3.1 Design Graph

We can give a formal definition of a design graph:

Definition 3.1 (Scheme of Design) *A general scheme of design is defined as a labeled directed graph $\langle N, E, LN, LE, R_{LN}, R_{LE}\rangle$ where, N, E are the sets of Nodes and Edges (with N finite and*

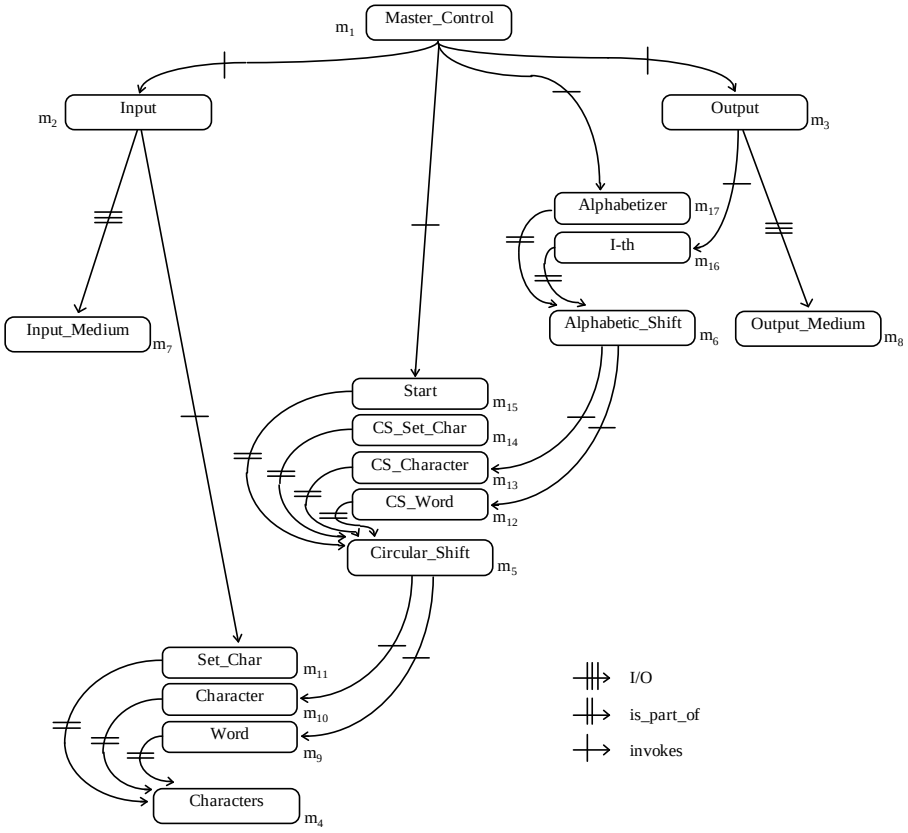

Figure 1: KWIC Design

non empty, and $E \subseteq N \times N$), LN and LE are sets of labels (disjoint and finite), R_{LN} is a total function in LN that assigns a label to each node ($R_{LN} : N \mapsto LN$), and R_{LE} is the relation that assigns at least a label to each edge ($R_{LE} \subseteq E \times LE, \Pi_1(R_{LE}) = E$.)

In other words: boxes are *nodes*, arrows are *edges*, box names are *node labels*, arrow names are *edge labels*, to give a name to a box is *to add a pair in R_{LN}* and to give a name to an arrow is *to add a pair in R_{LE}*.

We now begin to develop the different methods we will use in the verification of design properties.

3.2 From Design Graph to a Modal Model

Given that we will work with modal logics we have to transform the graph that represents a specific design into a model in the new formalism.

Definition 3.2 (Kripke Model of a Design Graph) *Given a design graph $G = \langle N, E, LN, LE, R_{LN}, R_{LE} \rangle$, the associated Kripke model is $\mathfrak{M} = \langle W, \{\text{-}a\!\rightarrow \mid a \in L\}, v \rangle$, where:*

1. $W = \{m_i \mid i \in N\}$.
2. $L = LE$.

3. $-a\rightarrow = \{(m_k, m_l) \in W' \times W \mid ((k,l),a) \in R_{LE}\}$.

4. *We take $\mathbb{P}$ (the set of propositional variables) equal to LN and set, for $p_i \in \mathbb{P}$, $m_j \in W$, $v(p_i, m_j) = 1$ if $(j, l_i) \in R_{LN}$, $v(p_i, m_j) = 0$ otherwise.*

Proposition 3.3 (Correctness of Translation) *If G is a design graph, $\mathfrak{M}$ obtained from G is a Kripke model.*

Example: Starting from the design graph obtained from Figure 1 we arrive at the model $\mathfrak{M} = \langle W, \{-/\rightarrow, -//\rightarrow, -///\rightarrow\}, v\rangle$ with:

$W = \{m_i \mid i \in \{1 \ldots 17\}\}$

$-/\rightarrow = \{ (m_1,m_2),(m_1,m_3),(m_1,m_{15}),(m_1,m_{17}),(m_2,m_{11}),(m_3,m_{16}),$
$(m_5,m_9),(m_5,m_{10}),(m_6,m_{12}),(m_6,m_{13}) \}$

$-//\rightarrow = \{ (m_9,m_4),(m_{10},m_4),(m_{11},m_4),(m_{12},m_5),(m_{13},m_5),(m_{14},m_5),$
$(m_{15},m_5),(m_{16},m_6),(m_{17},m_6) \}$

$-///\rightarrow = \{ (m_2,m_7),(m_3,m_8) \}$

$v(Master\_Control, m_1) = 1 \quad v(Input, m_2) = 1 \quad v(Output, m_3) = 1$
$v(Characters, m_4) = 1 \quad v(Circular\_Shift, m_5) = 1 \quad v(Alphabetic\_Shift, m_6) = 1$
$v(Input\_Medium, m_7) = 1 \quad v(Output\_Medium, m_8) = 1 \quad v(Word, m_9) = 1$
$v(Character, m_{10}) = 1 \quad v(Set\_Char, m_{11}) = 1 \quad v(CS\_Word, m_{12}) = 1$
$v(CS\_Character, m_{13}) = 1 \quad v(CS\_Set\_Char, m_{14}) = 1 \quad v(Start, m_{15}) = 1$
$v(I-th, m_{16}) = 1 \quad v(Alphabetizer, m_{17}) = 1$

$v(p_i, m_j) = 0$ for any other pair (p_i, m_j)

With $-/\rightarrow$ corresponding to *invokes*, $-//\rightarrow$ to *is_part_of* and $-///\rightarrow$ to *I/O*. □

3.3 Verifying Properties on the Model

In this section we give the algorithm to check validity of **KPI** formulas in a model. This algorithm is a modification of the algorithm in [CES86] to the operators in **KPI**.

Definition 3.4 (Model Checking Algorithm) *Given a formula φ of* **KPI** *and a finite model $\mathfrak{M} = \langle W, \{-a\rightarrow \mid a \in L\}, v\rangle$, the Model Checking Algorithm determines for any world m_j the sets C_j of subformulas of φ valid in m_j.*

Algorithm:

1. For any m_j
   $C_j = \emptyset$
2. For any $\psi \in Sub(\varphi)$ & $Size(\psi) = 1$
   For any $m_j \in W$
   If $(\psi = \top)$ then $C_j := C_j \cup \{\top\}$
   If $(\psi = p_i$ & $v(p_i, m_j) = 1)$ then $C_j := C_j \cup \{p_i\}$
3. For $n = 2$ to $Size(\varphi)$
   For any $\psi \in Sub(\varphi)$ & $Size(\psi) = n$
   For any $m_j \in W$
   If $(\psi = \neg\theta$ & $\theta \notin C_j)$ then $C_j := C_j \cup \{\neg\theta\}$
   If $(\psi = (\theta \vee \xi)$ & $(\theta \in C_j$ *or* $\xi \in C_j))$ then $C_j := C_j \cup \{(\theta \vee \xi)\}$
   If $(\psi = [a]\theta$ & $((\forall m_k)(m_j - a\rightarrow m_k$ *implies* $\theta \in C_k)))$ then $C_j := C_j \cup \{[a]\theta\}$
   // from where If $(\psi = \langle a\rangle\theta$ & $((\exists m_k)(m_j - a\rightarrow m_k$ & $\theta \in C_k)))$
   // then $C_j := C_j \cup \{\langle a\rangle\theta\}$
   If $(\psi = [a]_i\theta$ & $((\forall m_k)(m_k - a\rightarrow m_j$ *implies* $\theta \in C_k)))$ then $C_j := C_j \cup \{[a]_i\theta\}$

// from where If $(\psi = \langle a \rangle_i \theta$ & $((\exists m_k)(m_k -a\rightarrow m_j$ & $\theta \in C_k)))$
// then $C_j := C_j \cup \{\langle a \rangle_i \theta\}$

Suppose that we have to determine the validity of a formula φ in a model $\mathfrak{M}$. Our algorithm will proceed in stages working recursively in the size of the subformulas of φ. At the end of each stage, the validity of each sub-formula of size n will have been determined and the sub-formula can be interpreted as a new propositional symbol p_i. The algorithm builds for each world m_j, the set C_j of the subformulas of φ valid in that world. As φ is sub-formula of itself, when the algorithm stops, φ would be valid in m_j if and only if $\varphi \in C_j$.

Proposition 3.5 (Correctness of the Model Checking Algorithm) *Let* $\mathfrak{M} = \langle W, \{-a \rightarrow \mid a \in L\}, v\rangle$ *be a finite modal model and* φ *a formula of* **KPI**, *the model checking algorithm executed over* $\mathcal{M}$ *and* φ *stops, determining as output for any world* m_j *the set* C_j *of subformulas of* φ *valid in* m_j.

We can easily determine that the complexity of this algorithm is $O(t.\max(n, n.e))$ where t is the size of the formula φ, n is the number of worlds in $\mathfrak{M}$ and e is the number of edges in $\mathfrak{M}$. The complexity of our algorithm is less than that of the algorithm presented in [CES86] due to changes in the modal operators used and principally to the fact that it is not necessary to consider the transitive closure of the relations to determine the validity of the formulas. If some property is presupposed over the accessibility relations (transitivity, reflexivity, symmetry, etc.), the basic algorithm can be adapted to take into account these considerations.

Example: Starting again from the model corresponding to the design of the KWIC, we will verify the property "An ADT uses only services from others ADTs".

In our model we can characterize ADTs as those modules that defines functions: if a world corresponds to an ADT it must satisfy $\langle // \rangle_i \top$. To say that a module uses only services of ADTs is equivalent to say that any module invoked is part of an ADT, or formally $[/]\langle // \rangle \top$. Then we can state the above property as $\varphi = \langle // \rangle_i \top \rightarrow [/]\langle // \rangle \top$ and verify in the model that it is valid in any world. (Note how this property only involves model relations and not the modules themselves that can be abstracted through the symbol of tautology.)

Expanding abbreviations in φ, the algorithm proceeds over $\mathrm{Sub}([//]_i \neg\top \vee [/]\neg[//]\neg\top)$.

Figure 2 shows the relevant parts of the sets C_j obtained by the algorithm. □

3.4 Modifying the Model

The verification of a design is commonly associated with the proof of properties about the involved relations. Given the usual complexity of the designs, it is natural to try to obtain a more global view defining new relations over the original ones or paying attention only to certain subset of them.

In a modal language, accessibility relations are associated with the modal operators of the logic. The equivalent of creation or elimination of accessibility relations would be the definition of a new modal language with more or less modalities.

3.4.1 Model Expansion

From a strictly logical point of view, the idea of defining new modalities in the same logic is unusual. The new modalities we want to define are governed by the new relations introduced in the design which in turn have their behaviors determined by the basic relations that take part in

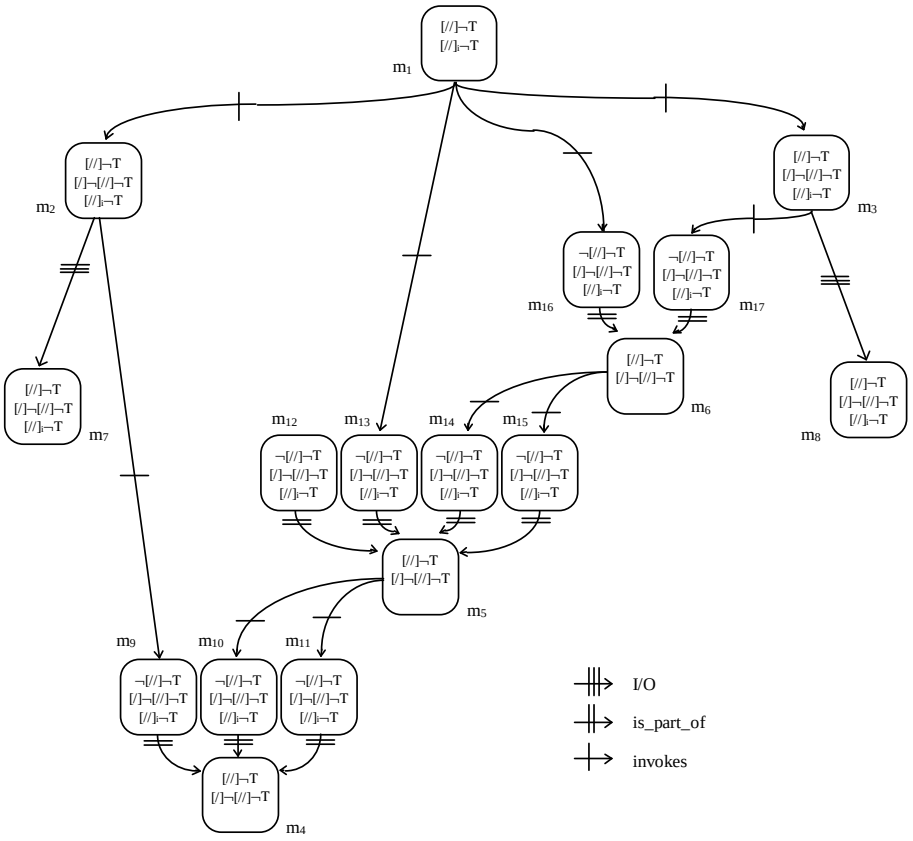

Figure 2: Result of the model checking algorithm for the formula $\langle // \rangle_i \top \rightarrow [/]\langle // \rangle \top$

the definition. In other words we are trying to introduce modalities as a kind of *shorthands* in the language with their associated relations appearing explicitly in the new model.

Our aim is similar to the ideas in Beth's Theory of Explicit Definition for first order logic [Bet53]. Beth's theory describes how new relational, functional and constant symbols can be added to a signature S of a first order language obtaining an extended signature S^+, *without adding expressive power to the logic.* The new symbols are defined as equivalent to formulas in the original language, and Beth proves that it is possible to give for any model $\mathfrak{M}$ corresponding to the original signature S, a model $\mathfrak{M}^+$ corresponding to the extended signature S^+ where formulas in the original language preserve their validity while formulas using new symbols respect the meaning assigned to these abbreviations via their definitions.

This result relies in two lemmas: the *Expansion* and *Definition Lemmas* [Bet53]. The expansion lemma states that the truth value of the formulas in $\mathcal{L}_S$ (the set of formulas over the signature S) will be preserved in the expansion, while the definition lemma and the invariance under substitution by equivalents of first order logic imply that any formula in $\mathcal{L}_{S^+} \backslash \mathcal{L}_S$ is equivalent to a formula in $\mathcal{L}_S$.

The importance of the definition lemma is shown principally in its *generality* (the sentences proposed in the set of definitions need only follow the specification given by Beth) and in the result of *uniqueness* of the extension (up to isomorphism.)

It seems that we can aspire to neither of these two qualities in a Theory of Definition for modal logic. On one hand, we cannot have uniqueness as it is well known that, because of the limited ex-

pressive power of modal languages, there are structurally different models that nevertheless satisfy the same modal theory. On the other hand, it will be hard to find a specification of what is a *modal definition*: while in first order logic any formula $\varphi(x_1, \ldots, x_n)$ defines uniquely an n-ary relation R in a model (the relation build from the tuples $(a_1, \ldots, a_n)$ such that $\varphi(x_1, \ldots, x_n)[a_1, \ldots, a_n]$ holds), it is not true that any modal formula $\varphi(p_1, \ldots, p_n)$ defines a modality M, or at least not a classical modality like those used in the logic **KPI**.

Anyway, for this work we need a much less general result than a complete Theory of Modal Definition. Given that our interest is to characterize syntactically the construction of new relations from basic relations, it is enough to give definitions for the operators we will use over relations (composition, union, etc.)

We begin then by showing a result similar to the Expansion Lemma over modal models.

Definition 3.6 (L^+-expansion) *Let $\mathfrak{M} = \langle W, \{-a\rightarrow \mid a \in L\}, v\rangle$ be a model, then the model $\mathfrak{M}^+ = \langle W^+, \{-a\rightarrow \mid a \in L^+\}, v^+\rangle$ is an L^+-expansion of $\mathfrak{M}$ iff*

1. *$W = W^+$.*
2. *$L \subset L^+$ and for any $a \in L$ it holds that $-a\rightarrow_{\mathfrak{M}} = -a\rightarrow_{\mathfrak{M}^+}$.*
3. *$v = v^+$.*

(i.e., $\mathfrak{M}^+$ is identical to $\mathfrak{M}$ except that it can define new accessibility relations.)

Lemma 3.7 (Modal Expansion Lemma) *Let $\mathfrak{M} = \langle W, \{-a\rightarrow \mid a \in L\}, v\rangle$ and $\mathfrak{M}^+ = \langle W^+, \{-a\rightarrow \mid a \in L^+\}, v^+\rangle$ be models, such that $\mathfrak{M}^+$ is an L^+-expansion of $\mathfrak{M}$, then for any formula $\varphi \in \mathcal{L}_L$ and for any $m \in W$, $\mathfrak{M} \models_m \varphi$ iff $\mathfrak{M}^+ \models_m \varphi$.*

This lemma proves that the formulas that do not use new symbols preserve their truth value from $\mathfrak{M}$ to $\mathfrak{M}^+$, but what happens with the formulas that use the new operators? To include the formulas in the extended language, we will use the result of invariance under substitution by equivalents that holds in our logic.

The following results let us characterize composition and union.

Proposition 3.8 *Let $\mathfrak{M} = \langle W, \{-a\rightarrow \mid a \in L\}, v\rangle$ be a model such that $b, c \in L$, then*

Composition, $-d\rightarrow = -b\rightarrow \circ -c\rightarrow$. *$\mathfrak{M}^+ = \langle W, \{-a\rightarrow \mid a \in L\} \cup \{-d\rightarrow \mid d \notin L$ and $m -d\rightarrow m'$ iff there exists m'' such that $m -b\rightarrow m''$ and $m'' -c\rightarrow m'\}, v\rangle$ is an $L \cup \{d\}$-expansion of $\mathfrak{M}$ which satisfies that for any $\varphi \in \mathcal{L}_{L\cup\{d\}}$, for any $m \in W$, $\mathfrak{M}^+ \models_m \langle d\rangle\varphi \leftrightarrow \langle b\rangle\langle c\rangle\varphi$.*

Union, $-d\rightarrow = -b\rightarrow \cup -c\rightarrow$. *$\mathfrak{M}^+ = \langle W, \{-a\rightarrow \mid a \in L\} \cup \{-d\rightarrow \mid d \notin L$ and $m - d\rightarrow m'$ iff $m -b\rightarrow m'$ or $m -c\rightarrow m'\}\rangle$ is an $L \cup \{d\}$-expansion of $\mathfrak{M}$ which satisfies that for any $\varphi \in \mathcal{L}_{L\cup\{d\}}$, for any $m \in W$, $\mathfrak{M}^+ \models_m \langle d\rangle\varphi \leftrightarrow (\langle b\rangle\varphi \vee \langle c\rangle\varphi)$.*

The characterization results on operations between relations are obviously limited by the expressive power of the logic. For example, as **KPI** has the operator of inverse accessibility, the inverse of a relation can be characterized. This operation could not be characterized in classical modal logic given the result that establish the higher expressive power of the language with inverse operators. A similar result will make it impossible to characterize in **KPI** the complement operation, as the logic with the inaccessibility operator (equivalent to accessibility over the complement of the relation) is more expressive than the logic with only accessibility operators.

Proposition 3.9 (Inverse, $-c\rightarrow = (-b\rightarrow)^{-1}$) *Let $\mathfrak{M} = \langle W, \{-a\rightarrow \mid a \in L\}, v\rangle$ be a model such that $b \in L$, then $\mathfrak{M}^+ = \langle W, \{-a\rightarrow \mid a \in L\} \cup \{-c\rightarrow \mid c \notin L$ and $m -c\rightarrow m'$ iff $m' -b\rightarrow m\}, v\rangle$ is an $L \cup \{c\}$-expansion of $\mathfrak{M}$ which satisfies that for any $\varphi \in \mathcal{L}_{L\cup\{c\}}$, for any $m \in W$, $\mathfrak{M}^+ \models_m \langle c\rangle\varphi \leftrightarrow \langle b\rangle_i\varphi$.*

Other operations over relations (as intersection and subtraction) cannot be fully characterized over the class of all models (again due to restrictions in the expressive power of the modal language), but they can be partially captured in the class of the models that correspond to designs.

Proposition 3.10 *Let $\mathfrak{M} = \langle W, \{-a\rightarrow \mid a \in L\}\rangle$ be a model that corresponds to a design graph such that $b, c \in L$ then*

Intersection, $-d\rightarrow = -b\rightarrow \cap -c\rightarrow$. *$\mathfrak{M}^+ = \langle W, \{-a\rightarrow \mid a \in L\} \cup \{-d\rightarrow \mid d \notin L$ and $m - d\rightarrow m'$ iff $m -b\rightarrow m'$ and $m -c\rightarrow m'\}\rangle$ is an $L \cup \{d\}$-expansion of $\mathfrak{M}$ which satisfies that for any $\varphi \in \mathcal{L}_{L\cup\{d\}}$, $p \in I\!P$, $\mathfrak{M}^+ \models_m \langle d\rangle(\varphi \wedge p) \leftrightarrow (\langle b\rangle(\varphi \wedge p) \wedge \langle c\rangle(\varphi \wedge p))$.*

Subtraction, $-d\rightarrow = -b\rightarrow \setminus -c\rightarrow$. *$\mathfrak{M}^+ = \langle W, \{-a\rightarrow \mid a \in L\} \cup \{-d\rightarrow \mid d \notin L$ and $m - d\rightarrow m'$ iff $m -b\rightarrow m'$ and no $m -c\rightarrow m'\}\rangle$ is an $L \cup \{d\}$-expansion of $\mathfrak{M}$ which satisfies that for any $\varphi \in \mathcal{L}_{L\cup\{d\}}$, $p \in I\!P$, $\mathfrak{M}^+ \models_m \langle d\rangle(\varphi \wedge p) \leftrightarrow (\langle b\rangle(\varphi \wedge p) \wedge \neg\langle c\rangle(\varphi \vee p))$.*

Note that in these two last cases the characterization is much more frail than in the cases above. First, we had to work only over the class of models corresponding to design graphs, but furthermore we could not obtain an equivalent formula for *all* formulas in the extended language. A formula $\langle c\rangle\psi$ with $\psi \neq (\varphi \wedge p)$ might not have an equivalent in the original language. Care must be taken with the formulas in the extended language if one wants to be sure that the meaning of the new modality is nothing more than a mere shorthand. Or in other words, the complete language $\mathcal{L}_{L\cup\{c\}}$ is in these cases more expressive than $\mathcal{L}_L$.

Example: As an example of the application of this method let us define from the basic relations *invokes* and *is_part_of* in the KWIC model, the relation *uses_ADT* as $uses\_ADT = invokes \circ is\_part\_of$. The new model $\mathfrak{M}^+$ is identical to that used in Section 3.2 with the addition of the new relation $\rightarrow$ defined as

$\rightarrow \;=\; \{\; (m_1, m_5), (m_1, m_6), (m_2, m_4), (m_3, m_6), (m_5, m_4), (m_6, m_5) \;\}$

With $\rightarrow$ corresponding to *uses_ADT*.

By Proposition 3.8, the new model $\mathfrak{M}^+$ use $\langle\ \rangle$ as shorthand of $\langle/\rangle\langle///\rangle$ providing furthermore the explicit relation corresponding to $\langle\ \rangle$ in the model. □

A more complex example of the application of this method can be found in [AH96a], where the hierarchy of the relation *uses* among modules in a CRC algorithm was formally verified.

3.4.2 Model Abstraction

Moving on the other way, we will try now to make the model simpler by eliminating details. We want to obtain less complex models that give us a clearer picture of the design. When we are trying to prove a given property, only some parts of the model are relevant and we would like to be able to get rid of the rest.

But anybody with some knowledge of modal logic will realize that this is the task for which *filtrations* were defined.

Even though the method of filtration is mainly used as a technique to prove decidability of a logic, its concrete result is to help in the detection of a simpler model that preserves the validity

of the formulas that take part in the process. From the point of view of Design, the method of filtration defines a new model that acts as an abstraction of the original model. This new model shows a simplified view, in which the non interesting details have been eliminated while new, more general and abstract concepts have been highlighted.

The formal definition of filtration is the following:

Definition 3.11 (Γ-Filtration) *Let $\mathfrak{M}_1 = \langle W_1, \{-a\rightarrow \mid a \in L_1\}, v_1\rangle$ and $\mathfrak{M}_2 = \langle W_2, \{-a\rightarrow \mid a \in L_2\}, v_2\rangle$ be two models, Γ a set of formulas of the language of $\mathfrak{M}_1$ ($L(\Gamma) \subseteq L_1$, with $L(\Gamma)$ the set of labels that appear in Γ.) Then $f : W_1 \mapsto W_2$ is Γ-filtration from $\mathfrak{M}_1$ onto $\mathfrak{M}_2$ if the following conditions are met:*

Sur. *f is surjective.*

Var. *For any propositional variable $p_i \in \Gamma$ and world $m_j \in W_1$, $v_1(p_i, m_j) = 1$ iff $v_2(p_i, f(m_j)) = 1$.*

Fil. *For any label a and formula φ such that $[a]\varphi \in \Gamma$ it holds that $\mathfrak{M}_1 \models_{m_j} [a]\varphi$ implies $\mathfrak{M}_1 \models_{m_k} \varphi$ for any pair of worlds m_j, m_k in W_1 such that $f(m_j) -a\rightarrow f(m_k)$ holds in $\mathfrak{M}_2$.*

Fil$_i$. *For any label a and formula φ such that $[a]_i\varphi \in \Gamma$ it holds that $\mathfrak{M}_1 \models_{m_j} [a]_i\varphi$ implies $\mathfrak{M}_1 \models_{m_k} \varphi$ for any pair of worlds m_j, m_k in W_1 such that $f(m_k) -a\rightarrow f(m_j)$ holds in $\mathfrak{M}_2$.*

The importance of the notion of Γ-filtration is given by the following result:

Theorem 3.12 (Preservation under Filtration) *Let Γ be a set of formulas closed by subformulas, and let f be a Γ-filtration from $\mathfrak{M}_1 = \langle W_1, \{-a\rightarrow \mid a \in L_1\}, v_1\rangle$ onto $\mathfrak{M}_2 = \langle W_2, \{-a\rightarrow \mid a \in L_2\}, v_2\rangle$. For any formula $\varphi \in \Gamma$, for any world $m_i \in W_1$, $\mathfrak{M}_1 \models_{m_i} \varphi$ iff $\mathfrak{M}_2 \models_{f(m_i)} \varphi$.*

Theorem 3.12 shows how the existence of a Γ-filtration between two models force formulas in Γ to be "treated equivalently" in both models, despite the differences that might exist between them.

Note that the aim of obtaining simpler models through filtration is not to decrease the complexity of the verification algorithm. This is obvious, given that the filtration algorithm has to establish the validity of all the formulas in the set of filtration. The filtration method aspires to a result of higher conceptual value. The filtered model shows the design as it is "seen" from the point of view of the properties that were chosen as relevant and included in the filtration set Γ.

The algorithm we propose constructs from any finite model of design and any finite set of formulas Γ, an abstract view with the guaranty that a Γ-filtration can be defined between them.

Definition 3.13 (Filtration Algorithm) *Let $\mathfrak{M} = \langle W, \{-a\rightarrow \mid a \in L\}, v\rangle$ be a modal model and Γ a finite set of formulas closed by subformulas, the Filtration Algorithm builds a model $\mathfrak{M}^* = \langle W^*, \{-a\rightarrow^* \mid a \in L^*\}, v^*\rangle$ as the filtered model by Γ.*

Algorithm:

1. Run over $\mathfrak{M}$ the model checking algorithm with formula $\bigwedge\Gamma$.
2. $W' := W$
   $W^* := \emptyset$
   While $W' \neq \emptyset$

Take $m_i \in W'$
$|m_i| := \{m_i\}$
$W' := W' - \{m_i\}$
For any $m_j \in W'$
If $(C_i = C_j)$ then
$W' := W' - \{m_j\}$
$|m_i| := |m_i| \cup \{m_j\}$
$W^* := W^* \cup \{|m_i|\}$

3. $L^* := L(\Gamma)$
4. For any edge $m_i -a\rightarrow m_j$ and $a \in L^*$
$|m_i| -a\rightarrow^* |m_j|$
5. For any $p_i \in \Gamma$
For any $m_j \in W$
If $(v(p_i, m_j) = 1)$ then
$v^*(p_i, |m_j|) := 1$

// We will assume $v^*(p_i, |m_j|) := 0$ for any $m_j \in W$ and for any $p_i \notin \Gamma$.

Proposition 3.14 (Correctness of the Filtration Algorithm) *Let $\mathfrak{M} = \langle W, \{-a\rightarrow \mid a \in L\}, v\rangle$ be a finite modal model and Γ a finite set of formulas closed under subformulas, the filtration algorithm executed over $\mathfrak{M}$ stops and upon termination the model $\mathfrak{M}^* = \langle W^*, \{-a\rightarrow^* \mid a \in L^*\}, v^*\rangle$ ($\mathfrak{M}$ filtered by Γ) is obtained.*

Furthermore, any model can be filtered, i.e. for any model $\mathfrak{M}$, for any set of formulas Γ there exists at least a model $\mathfrak{M}'$, such that a Γ filtration can be established from $\mathfrak{M}$ to $\mathfrak{M}'$.

Theorem 3.15 (Filtration Existence) *Let $\mathfrak{M} = \langle W, \{-a\rightarrow \mid a \in L\}, v\rangle$ be a modal model and $\mathfrak{M}^* = \langle W^*, \{-a\rightarrow^* \mid a \in L^*\}, v^*\rangle$ the model obtained from $\mathfrak{M}$ for a set of formulas Γ by the filtration algorithm. Define the relation $\sim$ as $m_i \sim m_j$ iff m_i and m_j make the same subset of Γ true in $\mathfrak{M}$. Define $|m_i| = \{m_j \in W \mid m_i \sim m_j\}$. Then $f : W \mapsto W^*$ such that $f(m_i) = |m_i|$ is a Γ-filtration from W onto W^*.*

A simple analysis shows that the filtration algorithm has $O(t.\max(n, n.e) + n^2)$, where t is the size of $\bigwedge \Gamma$, n the number of nodes in the model and e the total number of edges.

Note that the model obtained will depend on the formulas contained in Γ. We must examine the composition of Γ if we want to comprehend how the method builds the abstraction. We will see that a well defined Γ returns the expected abstraction. Here are some examples:

Example: Let us begin with a simple case. We will work over the model $\mathfrak{M}^+$ in the Model Expansion example. Suppose we are interested in working only with the properties that involve the new relation *uses_ADT* defined as the composition of *invokes* and *is_part_of*. Then we are only interested in the modules of the model (that syntactically are represented by the propositional symbols), and the mentioned relation (that we can abstract as $[use\_ADT]\top$.)

We propose $\Gamma = \{Master\_Control, Characters, \ldots, Alphabetic\_Shift, [uses\_ADT]\top\}$.

As Γ contains all propositional symbols and each world in our example satisfies a single propositional symbol (its name) we will have $|m_i| = \{m_i\}$ for any i, the worlds will not be collapsed in step 2 of the algorithm. As $[uses\_ADT]\top$ is in Γ, step 4 will copy all the edges $-uses\_ADT\rightarrow$ from $\mathfrak{M}^+$ to $(\mathfrak{M}^+)^*$ while all the other relations will be ignored as they have no representative in Γ.

The filtered model would preserve the relation we were interested in, and as the definition method introduced the new relation explicitly and made it independent of the basic relations, all the others can be eliminated as irrelevant.

In the model obtained by this first filtration we can observe that some totally unconnected worlds were left in $(\mathfrak{M}^+)^*$. These worlds are not relevant to the analysis of the relation *uses_ADT* and yet, they were not eliminated

in the abstraction. This is justified by the incorporation of their names in Γ, given that they were mentioned the filtration preserved them. We can define a new Γ with the information obtained from the last filtration as $\Gamma = \{Input, Output, Master\_Control, Characters, Circular\_Shift, Alphabetic\_Shift, [uses\_ADT]\top\}$ and now, all the unconnected worlds will be collapsed into a single class that will represent them in the new abstraction.

The final result is shown in Figure 3. □

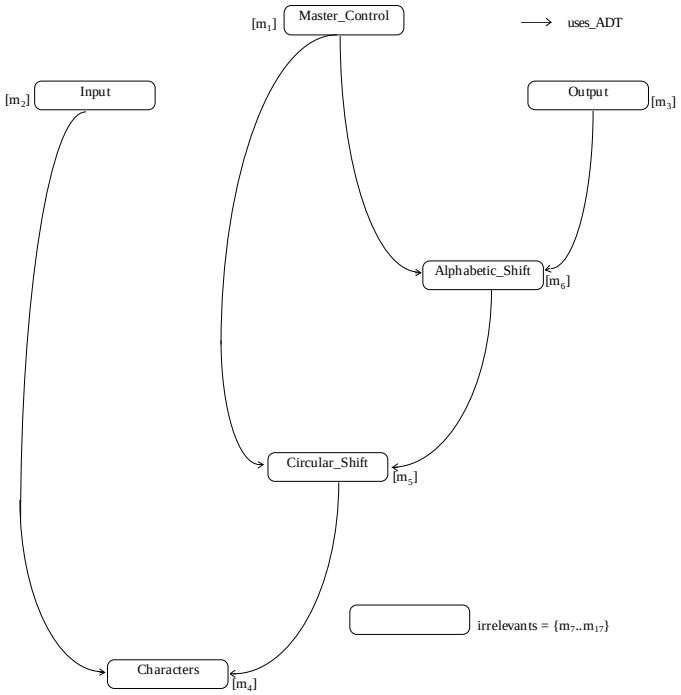

Figure 3: $\mathfrak{M}^+$ filtrated preserving the relation *use_ADT*

This brief example shows how the correct definition of Γ can be achieved in steps, using the information obtained in successive abstractions.

Note that we can generalize this example for any number of relations $-a_i \rightarrow$ simply adding to Γ the corresponding $[a_i]\top$ formulas.

Example: The previous example presents a rather simple application of filtrations. We consider the following one as a real sample of the potential of this method.

Let us take again the original KWIC model and the formula we used in the model checking example and let $\Gamma = Sub(\langle // \rangle_i \top \rightarrow [/]\langle // \rangle \top)$. In other words, we want to obtain a view of the model when what we have in mind is the property we already know valid: "An ADT uses only services from others ADTs". The algorithm will return the model in Figure 4.

The modules of the original model were brought together according to the role they play with respect to the formulas in Γ. The subformulas of $\langle // \rangle_i \top \rightarrow [/]\langle // \rangle \top$ characterize the purpose of the different entities in the design. That is how we obtain the classes ADTs, Functions, I/O and Master_Control (the names are ours, they are not provided by the algorithm) that we can now, thanks to the abstraction, clearly distinguish in the original model. In this abstraction it is obvious that the ADTs only use functions that are part of others ADTs, but the result obtained reaches further: the abstraction shows the *design architecture*, reflecting the primacy of the use of functions from ADTs and other peculiarities, like the centralized administration of I/O via Master_Control. □

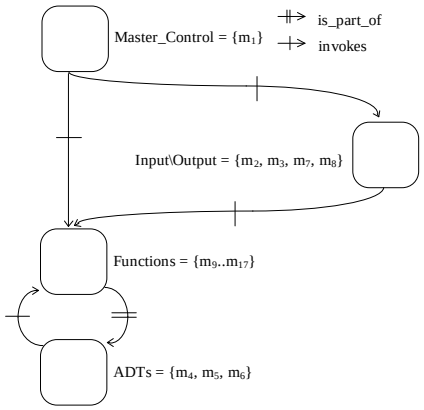

Figure 4: $\mathfrak{M}$ filtrated by property $\langle /// \rangle_i \top \rightarrow [/] \langle /// \rangle \top$

4 Application Examples

This section will be devoted to the application of the formal methods for design analysis discussed above. We comment on two examples: a general scheme showing how to detect the incorrect use of an interface module, and a case taken from the area of Object Oriented Design, showing how our methods can be applied on graphs with more information in the modules.

4.1 Incorrect Use of Interface

One of the central notions used to obtain a correct modularization of a system is the concept of Information Hiding. In many cases, this notion is enforced through strict use of interfaces to access the modules. The interface makes the implementation of the module functionality independent from how that functionality is accessed.

We can think of a library as a set of submodules that are "administered" by an interface module attending the external requirements and translating them into requirements for the different internal submodules. Figure 5*a*. shows how we could represent the correct use of the interface of a library.

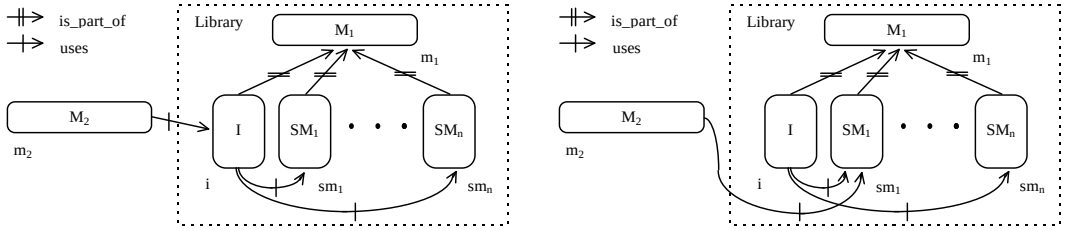

Figure 5: Interfaces:
a. M_2 uses SM_1 through the interface of M_1. *b*. M_2 uses SM_1 bypassing the interface of M_1.

Now, supposed that we have a design error where the correct use of the interface shown by Figure 5*a*. is not respected, as in Figure 5*b*.

Comparing these two pictures, we can produce a formula that represents the correct use of the interface. For example, M_2 is incorrectly using M_1 if it uses a part of m_1 that is not an

interface, i.e. it is necessary that each time a part of M_1 is used, this part must be the interface: $[/](\langle // \rangle M_1 \rightarrow I)$ (this represents "M_2 is correctly using library M_1".) Verifying (using the model checking algorithm) $\mathfrak{M} \models_{m_2} [/](\langle // \rangle M_1 \rightarrow I)$ proves that M_2 uses correctly M_1. It is easy to see that this is not the case in Figure 5*b*. The same property can be verified from m_1. The module m_1 is used correctly if all of its submodules satisfy that they are either the interface or are only called by the interface: $[//]_i(I \vee [/]_i I)$ (this represents "the library M_1 is used correctly".) Of course, this formula must be valid in m_1, not in m_2.

Note that the existence of two formulas for checking the property corresponds to the common method of checking the system when a library is included and rechecking each time a new module which uses the library is added.

4.2 Design in the Object Oriented Paradigm

Now we turn to the analysis of an example from the Object Oriented Paradigm. The design we will study (with minimal changes) can be found in [WWW90].

In the Object Oriented Paradigm, we usually have much more information about each class in the design. For example, the list of responsibilities that the class defines, the class type (abstract or concrete), etc. are generally included in the graph.

We will show how to handle the extra information in our formalism, using ideas presented in [BC95], by transforming the implicit information inside the class into explicit information.

In the design phase of the object oriented cycle of software development, hierarchy graphs are used as a technique to model the inheritance relations between classes. In these graphs, the nodes are classes and the relation between them is *is_subclass_of* defining the inheritance from superclass to subclass. In addition, the classes are marked as abstract (classes that do not possess instances and are created only to factor common properties) or concrete (classes that do possess instances, and are active objects during the execution of the program.) Each class defines its responsibilities (the tasks they can perform and characterize their behavior) which, by the hierarchical relation, are inherited by subclasses. However, a subclass can redefine an inherited responsibility to adequate it to its particular behavior.

Once the classes of a system and the inheritance hierarchy have been determined, these relations can be analyzed for identification and solution of design problems. This analysis is based in the following points:

- Model the hierarchy *is_subclass_of*.

- Factor the common responsibilities as high as possible in the hierarchy.

- Check that abstract classes do not inherit from concrete classes.

- Eliminate classes that do not add any functionality.

We will check the design of the hierarchy of the class Drawing_Element of a Drawing Editor Design (Figure 6.)

The boxes represent the different subclasses, and for each of them the responsibilities they define (d) or redefine (r) are specified. The abstract classes are also distinguished (marked on the

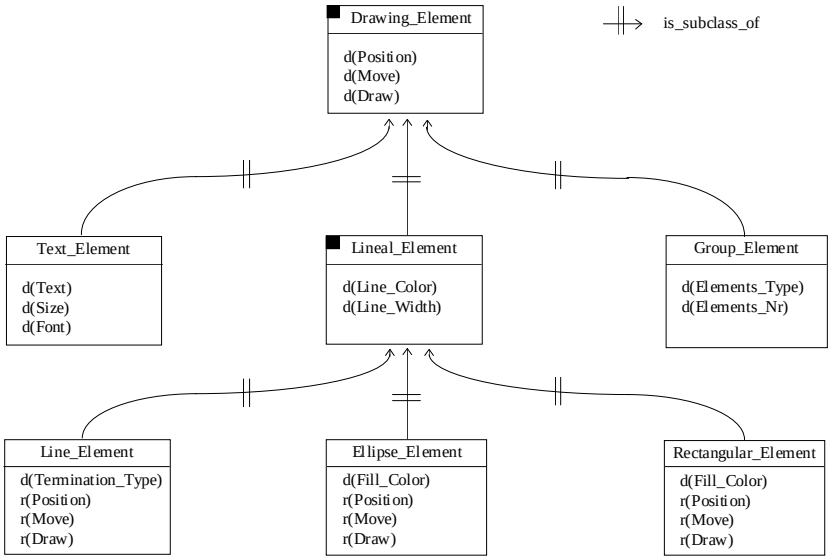

Figure 6: Design of the class Drawing_Element

upper left corner.) As we said before, the responsibilities are inherited downward in the hierarchy. Each subclass specifies the *new* responsibilities that characterizes it.

At first sight, the design looks like containing much more information. And yet, we will show that it is possible to model the above design in our formalism.

The flexibility provided by the use of accessibility relations lets us incorporate all the additional information in the following way: The monadic property *is_abstract_class* is really a subset S of the set of classes, but we can represent this set as the diagonal relation on $S \times S$. Definition and redefinition of responsibilities are dyadic relations between classes and responsibilities and can be directly translated as accessibility relations $-d\rightarrow$ and $-r\rightarrow$ once a world has been assigned to each responsibility.

We can see in Figure 7 the result of this translation.

Now we can use the methods presented in the above sections to analyze each of the rules of good design that must be taken into account.

Modeling the hierarchy *is_subclass_of*: "Modeling the relation among classes helps to define the class hierarchy and to guaranty that subclasses support at least all the responsibilities of their superclasses."

Given the way classes are modeled, the inheritance of responsibilities from class to subclass is immediate. By definition, the responsibilities of a class are those explicitly defined $\langle d\rangle p_i$ and those defined by their superclasses $\langle //\rangle\langle d\rangle p_j$ (with $-//\rightarrow$ transitive.)

The verification of the hierarchy is similar to the one carried out in the CRC example in [AH96a]. In that article the definition method, the filtration method and the model checking algorithm are used to detect a cycle in the hierarchy of the relation *uses* among modules.

Factoring common responsibilities as high as possible in the hierarchy: "If a set of classes supports a common responsibility they should inherit this responsibility from a common

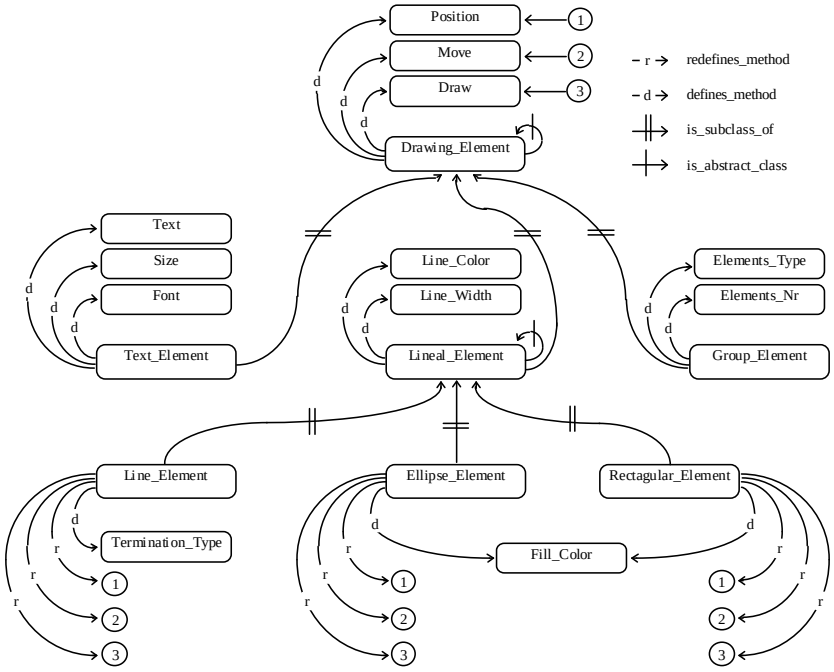

Figure 7: Modal Model of the class Drawing_Element

superclass. If this superclass does not exist one must be created, passing the responsibility to the new class. Probably, this new class will be an abstract class."

In our example, the verification of correct factorization can be done in the following way: define Γ as $\{(\langle d\rangle p_i \wedge p_j) \rightarrow \langle //\rangle[//]_i(\langle d\rangle p_i \rightarrow p_j) |$ *p_j are the names of the classes and p_i are the responsibilities that correspond to each of the classes p_j*$\}$. Each one of the formulas in Γ, for example $(\langle d\rangle Size \wedge Text\_Element) \rightarrow \langle //\rangle[//]_i(\langle d\rangle Size \rightarrow Text\_Element)$ says that if a class (Text_Element) defines a responsibility (Size), this responsibility is not defined in any other module that is part of the hierarchy (in the superclass of Text_Element it is necessary that the definition of Size should be done in the class Text_Element.) If all the formulas in Γ are valid in the model, we can guarantee that we have a good factorization. If not, the formulas that were not satisfied will show error cases.

In Figure 7 we can check that the classes Ellipse_Element and Rectangular_Element share the responsibility Fill_Color and for this reason the responsibility will be factored in a new abstract class Solid_Element, as shown in Figure 8.

Checking that abstract classes do not inherit from concrete classes: "By their nature, abstract classes support their responsibilities independently from the implementation. Therefore, they should never inherit from concrete classes, which can specifically depend on the implementation. If a design has this kind of inheritance, the problem can be solved by creating a new abstract class from which the abstract as well as the concrete class can inherit their common behavior."

We can capture this rule by the formula saying: all the superclasses of an abstract class are also abstract classes $\langle /\rangle\top \rightarrow [//]\langle /\rangle\top$ (with the relation $-//\rightarrow$ transitive.) The model checking algorithm will verify that the property is valid in our example.

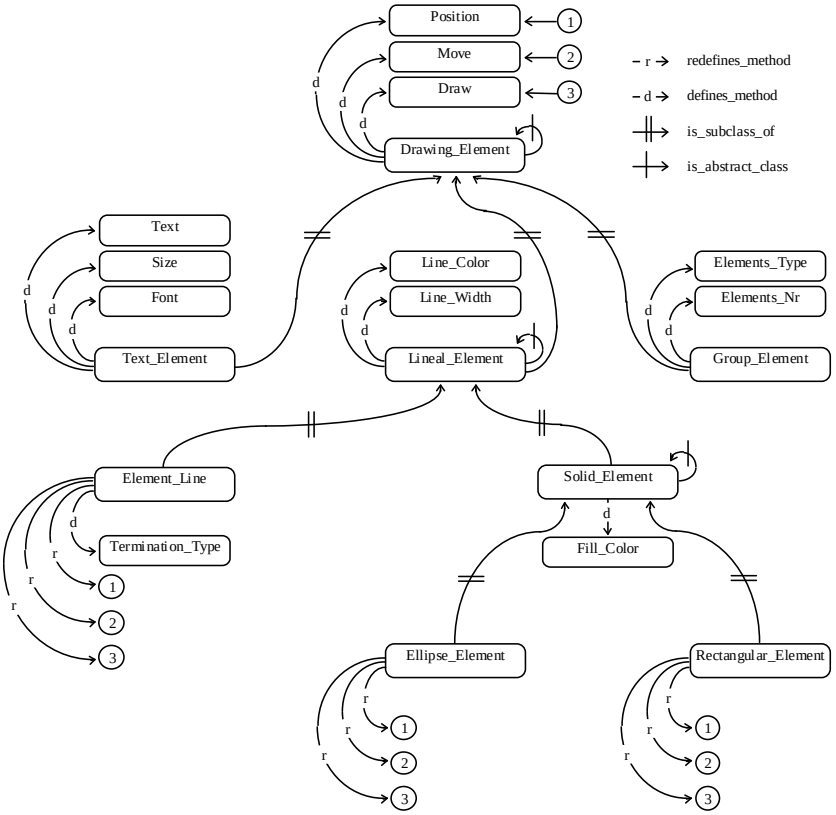

Figure 8: Factorization of the abstract class Solid_Element

Eliminating classes that do not add any functionality: "Classes that do not have responsibilities must be eliminated. If a class inherits a responsibility that will be implemented in a unique way, then it adds functionality in spite of not having responsibilities of its own, and it should not be eliminated. Abstract classes that do not define or redefine responsibilities have no use and must be eliminated because they will not increment reusability."

The formula associated with this rule is simply $\langle d \rangle \top \vee \langle r \rangle \top$, which must be valid everywhere in the model. Note that it is permitted for abstract classes to redefine responsibilities. It is not incorrect for an abstract class to have implementation code even though they usually define only the interfaces of the responsibilities for their subclasses. In any case, the designer has to take care that the code incorporated does not affect its condition of being an abstract class. Therefore, it can be useful to detect which are the abstract classes that incorporate implementation code. This can be done checking the formula $\langle / \rangle \top \wedge \langle r \rangle \top$.

5 Conclusions

In this work we have presented a notation for the description of software system designs and a method for the the verification of properties over them.

From the similarity among the models used in the areas of Modal Logic and Software Design a logic was defined to be used as a specification language in the verification and description of designs. The design graphs are considered models of the poly-modal logic with inverse operator

KPI. The methods used to verify properties are developed from techniques in classical modal logic extended to cover the differences in the modified formalism. We present the procedures to derive the modal model associated with a design, the model checking algorithm to verify properties, the expansion method to define new relations and the model filtration method for abstraction. These methods are demonstrated in two application examples. The proofs of the results presented in this work can be found in [AH96b].

We believe that these methods help to accomplish the tasks that are commonly found in the process of Software Development and to formalize the process of design verification.

As shown in the application examples the methods proposed are useful, but for "real-life" cases with design of hundreds of components an automatic tool is needed. A prototype covering all the concepts and functionalities described was implemented using the functional language SML-NJ in [AH96a].

This work opens several possibilities of research in the area of Modal Logic applied to Software Engineering:

The Theory of Modal Definition must be completed with an analysis of the interrelation existing between the expressive power of the language and its capacity to characterize different operations on relations.

The results obtained through the filtration method are encouraging but the real power of this technique as an abstraction method remains to be studied in detail.

The last application example shows how to handle in our formalisms graphs of design with complex information in the modules. We do not claim though, that this can be done in all cases. How should the formalism be modified to cover other types of information such as, for example, constraints on the implementation?

It would be interesting to apply our techniques in more and more varied cases (not only in the area of design) to find new properties to verify. Our approach might also be useful in the description of software architectures. In this area however, the problem has higher complexity. The graphs used in this field seem to capture correctly the static components of the system, but not so the dynamic components that define the interaction among them [GS93, AG94]. In these graphs, relations have fundamental importance and the modalities used should be able to reflect this relevance. There exist some work in this area as, for example, [DC95] where a (non modal) syntactic language for the description of architectures is presented.

References

[AG94] Allen, R. and Garlan, D. Formalizing architectural connection. In *International Conference on Software Engineering*, May 1994.

[AH96a] Areces, C. and Hirsch, D. La lógica modal como herramienta de ingeniería de software. 1996. Tesis de Licenciatura.

[AH96b] Areces, C. and Hirsch, D. Modal logic as a software engineering tool. Technical Report 96-0004, Universidad de Buenos Aires. Facultad de Ciencias Exactas y Naturales. Departamento de Computación., 1996. http://www.dc.uba.ar.

[ARP95] Agustí, J., Robertson, D., and Puigsegur, J. Grasp: A graphical specification language for the preliminary specification of logic programs. Technical report, Institut d'Investigació en Intel.ligència Artificial (CSIC), 1995.

[BC95] Bourdeau, R. and Cheng, B. A formal semantics for object model diagrams. *IEEE Transactions on Software Engineering*, 21(10), October 1995.

[Bet53] Beth, E. On Padoa's method in the theory of definition. *Koninklijke Nederlandse Akad*, pages 330–339, 1953.

[BT93] Bernhard, S. and Tiziana, M. Module configuration by minimal model construction. Technical Report MIP-9313, Universität Passau. Fakultät für Mathematik und Informatik, 1993.

[BT96] Bernhard, S. and Tiziana, M. Automatic synthesis of design plans in METAframe. Technical Report MIP-9610, Universität Passau. Fakultät für Mathematik und Informatik, 1996.

[CES86] Clarke, E., Emerson, E., and Sistla, A. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2), 1986.

[CMR92] Cosens, M., Mendelzon, A., and Ryman, A. Visualizing and quering software structures. In *ICSE'92 Proceedings of the 14th International Conference on Software Engineering*, 1992.

[DC95] Dean, T. and Cordy, J. A syntactic theory of software architecture. *IEEE Transactions on Software Engineering*, 21(4), April 1995.

[GJM91] Ghezzi, C., Jazayeri, M., and Mandrioli, D. *Fundamentals of Software Engineering.* Prentice Hall, 1991.

[GS93] Garlan, D. and Shaw, M. An introduction to software architecture. *Advances in Software Engineering and Knowledge Engineering*, I, 1993.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, June 1994.

[Lam94] Lamport, L. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3), 1994.

[MSV84] Maibaum, T., Sadler, M., and Veloso, P. Logical specification and implementation. *Lecture Notes in Computer Science*, (18), 1984.

[Par72] Parnas, D. L. On the criteria to be used in decomposing systems into modules. *Communications of ACM*, pages 1053–1058, December 1972.

[Per87] Perry, D. Software interconnection models. In *Proceedings of the 9th International Conference on Software Engineering*, pages 61–69, Monterey, California, March 1987.

[Pnu77] Pnueli, A. The temporal logic of programs. In *In Proceedings of the 18th. Annual Symposium on the Foundations of Computer Science*, New York, 1977. IEEE.

[PP95] Paul, S. and Prakash, A. A query algebra for program databases. *IEEE Transactions on Software Engineering*, 22(3), March 1995.

[WWW90] Wirfs-Brock, R., Wilkerson, B., and Wiener, L. *Designing Object-Oriented Software.* Prentice Hall, 1990.