Departamento de Computación

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

INFORME TÉCNICO

Modal Logic as a Software Engineering Tool

Areces, Carlos Eduardo
and
Hirsch, Dan Francisco

Report n.: 96004

Pabellón 1 - Planta Baja - Ciudad Universitaria
(1428) Buenos Aires
Argentina

http://www.dc.uba.ar

Title: **Modal Logic as a Software Engineering Tool**

Authors: **Areces, Carlos Eduardo**
and
Hirsch, Dan Francisco

E-mail: **postmast@logia.uba.ar**
and
dhirsch@compsc.uba.ar

Report n.: **96004**

Key-words: **Modal Logic, Model Checking, Software Engineering, Verification of Design Properties**

Abstract: A methodology to represent software system designs is presented jointly with the means to verify properties over them. Design graphs are considered as models of an extended modal logic. The procedures or logic techniques to derive the modal model associated to any design, the algorithm to check properties, the method to define new relations and the method of model filtration are introduced. These methods are applied in two design examples of different kind. The logic proposed is called **KPI**, polimodal with inverse operators (which in a connected model achieves total access) and will be used as a property specification language that will be verified through an algorithm of model checking. The techniques are traditional methods of the (modal and classical) logic modified to carry out the tasks usually found in the process of software engineering. The methods proposed seem to be effective and simple to be implemented.

To obtain a copy of this report please fill in your name and address and return this page to:

Infoteca
Departamento de Computación - FCEN
Pabellón 1 - Planta Baja - Ciudad Universitaria
(1428) Buenos Aires - Argentina

TEL/FAX: (54)(1) 783-0729
e-mail: infoteca@dc.uba.ar

You can also get a copy by anonymous ftp to: **zorzal.dc.uba.ar/pub/tr**

or visiting our web: **http://www.dc.uba.ar/people/proyinv/tr.html**

Name:.....

Address:.....

.....

Modal Logic as a Software Engineering Tool

Areces, Carlos Eduardo
and
Hirsch, Dan Francisco
Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria
Buenos Aires - Argentina
e-mail: postmast@logia.uba.ar, dhirsch@compsc.uba.ar

A methodology to represent software system designs is presented jointly with the means to verify properties over them. Design graphs are considered as models of an extended modal logic. The procedures or logic techniques to derive the modal model associated to any design, the algorithm to check properties, the method to define new relations and the method of model filtration are introduced. These methods are applied in two design examples of different kind. The logic proposed is called **KPI**, polimodal with inverse operators (which in a connected model achieves total access) and will be used as a property specification language that will be verified through an algorithm of model checking. The techniques are traditional methods of the (modal and classical) logic modified to carry out the tasks usually found in the process of software engineering. The methods proposed seem to be effective and simple to be implemented.

Key-words: Modal Logic, Model Checking, Software Engineering, Verification of Design Properties

---

Contents

The work in this report is part of the thesis of the authors. This work is partially supported by UBACyT proyect EX-186.

1. INTRODUCTION

Software Engineering is the field of the Computer Science that deals with the design and the construction of systems of so large and complex software that they should as a rule be built by a group of persons. Normally, these systems are used by several years and during their life suffer changes such as the correction of defects, the incorporation of new applications, the elimination of old facilities, or the migration to new ambients. As all engineering disciplines, the development of software systems requires the use of tools. But the successful development of a system requires the selection of tools adapted to achieve a specific purpose. Software Engineering environments should provide tools that let the engineer to better understand the system that is under design. These tools should be at the same time powerful, simple and as formal as necessary (or possible).

In our work, we will devote special atention to the Design Step in the Software Life Cycle.

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

Different methods have been used to carry out the Design. The simpler among them use graphics, annotations and even different colors to suggest an idea. But the designs obtained by these methods are ambiguous and to prove properties over them is almost impossible. The great variety and the free style involved make them improper for the formal proof. An important improvement on this situation was the definition and, mainly, the general acceptance of a set of tools specifically defined to treat this problem, as the Entity-Relation graphs, the Data Flow Graphs, the Structures Charts, etc., that constitute what is called the Structured Design.

The Structured Design was the first technique in reflecting and handling the complexity of the modern systems. Its principal contribution was to recognize that to assault the details of a problem, it is first very important to have a clear understanding of the problem itself. To this end, it provided a small set of tools for the semi-formal description of the problem.

In spite of the fact that these tools eliminated the variety, they did not exclude the ambiguity, and the formal proof of properties was still impossible [Ghezzi, C. et al. 1991]. In few words, a new global language for the description of design had been established and a graphic language was selected as the better option, but it lacked still, a *formal semantic*.

Currently, many researchers have been working in various aspects of this problem using new techniques. The work of [Cosens, M. et al. 1992], for example, has as a goal to provide tools to manipulate and to obtain simplified visions from the design, that provide a better understanding of the system. Other examples of projects in this line can be those of [Agustí, J. et al. 1995] that treats the specification of high level logic programs using operations on sets through a graphic interface; [Paul, S. and Prakash, A. 1995] who use a query algebra to reconstruct the design from code of a system (reingineering) and the more traditional approach of [Maibaum, T. et al. 1984] in which given a high level specification using ADTs, an scheme of module composition and refinement (theory of implementation) is proposed, through an extended first order logic.

Our opinion however, is that we should accomplish a jump toward a higher level of abstraction. In addition to a tool that provides simplified visions of the system we want to reason on graphics of design in a more abstract form. This superior level of abstraction is obtained when we consider the designs as modal structures. We think that the use of a modal language to describe both the design and its properties gives us the expressivity we are seeking.

The selection of a modal logic for the description of design is justified by the similarities existing among the models of both areas: directed graphs with labeled edges and nodes. But furthermore, the use that is given to both structures also coincides, given greater importance to the *relationships*, which determine the properties of the design.

The use of a modal logic for the description of the design provides a common language to characterize the model and its properties, with capacity of formal proof, an unambiguous semantics and the possibility of automatic verification.

2. POLIMODAL LOGIC WITH INVERSE OPERATORS

Classically, modal logic is understood as the logic of the modal operators [ ] and <>. 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 in which we base our work. The models we will use have many accessibility relations between worlds. Thus, we need different modalities $[a]$, one for

each relation $-a \rightarrow$ in the model. Our models will be t-uples $M =< W, \{-a \rightarrow \mid a \in L\}, V >$, where the usual relation R has been replaced by a set of relations. L is called the *sort* of the logic and is a set of labels that let us distinguish between the different accessibility relations. In our case L will always be a finite set. It can be shown that this extension adds expressive power to the language. Any classical model is admitted in the formalism (just use only one relation in the set of relations) and any formula that use more than one kind of modality cannot be equivalent to a formula in the old formalism. We will call the minimal normal logic extended by multiple modalities **KP** (polimodal **K**)

Polimodal logics were born in the frame of the logics for belief and knowledge. A classical example can be found in [Hintikka, J. 1962] where $[a]\varphi$ was interpreted as *a knows* or *believes* φ, or semantically (in a transition systems that represents beliefs or knowledge) $s - a \rightarrow t$ symbolizes the fact that a in the state s knows or believes the propositions in t. Polimodal logics can naturally be applied to the description of action and change.

The other extension is also traditional and is related with the area of temporal logic [Burgess, J. P. 1984]. In this area, the accessibility relation $\rightarrow$ is interpreted as the temporal relation that orders time instants: $t_1 \rightarrow t_2$ iff t_1 is an instant that precedes (temporally) t_2. If $\rightarrow$ is interpreted this way, $<>$ turns out to be a *future* operator: $<> \varphi$ is true at instant t_1 iff there exists an instant t_2 after t_1 in which φ is true. It is easy to verify that the dual operator $[\,]$ is not the *past* operator. $[\,]$ represents *always in the future*: $[\,]\varphi$ iff $\neg <> \neg\varphi$ iff there is no instant after the current one where $\neg\varphi$ is true or, in other words, all instants (if there is one) after the current one are instants where φ is true. With this interpretation and without any other modification, classical modal logic is a *temporal logic of pure future*, the expressions about the past cannot be represented in this language. The extension in this case is not obtained through the redefinition of the concept of model but through the addition of the inverse modality $<>_i$ and its dual $[\,]_i$. $M \models_m <>_i \varphi$ iff there exists a world m' in W such that $m' \rightarrow m$ and $M \models_{m'} \varphi$ (note that the positions of m' and m are interchanged with respect to the definition of $M \models_m <> \varphi$). In this way $<>_i \varphi$ is true at instant t_1 iff there exists an instant t_2 preceding t_1 in which φ is true. In our work, we will not use inverse operators as temporal operators; we add them because of the increased expressive power they give to the logic. That is the reason why we will not always make the assumption about transitivity and antisymmetry on the accessibility relation usually made in the temporal interpretation. Note that in this case, the operators $<>$ and $<>_i$ are intimately related (not as $<a>$ and $<b>$ with $a, b \in L$, in the last extension). In particular, they verify the property that given that φ is true now, it will always be the case that φ has been (and symmetrically to the past, it has always been the case that φ will be). This relation will be reflected in the axiomatization of the system. We will call the minimal normal logic extended by inverse modalities **KI** (Inverse **K**).

We will present now the polimodal logic with inverse operators **KPI** that adds both extensions.

DEFINITION ALPHABET. *Given a sort L of labels (always finite), the alphabet of the logic* $\mathbf{KPI}_L$ *is defined as* $\mathbf{API}_L = \{\vee, \neg, (,), \top\} \cup \{[a] \mid a \in L\} \cup \{[a]_i \mid a \in L\} \cup \{p_i \mid i \in I\!N\}$.

The restriction to a finite language of n propositional symbols ($n \in I\!N$) *is the alphabet* $\mathbf{API}_L^n = \{\vee, \neg, (,), \top\} \cup \{[a] \mid a \in L\} \cup \{[a]_i \mid a \in L\} \cup \{p_i \mid i < n\}$.

DEFINITION WELL FORMED FORMULAS. *Given a sort L of labels, the well formed formulas of the logic* $\mathbf{KPI}_L$ *are defined by recursion as:*

BF1. $\top$ *is a wff.*

BF2. Any propositional symbol is a wff.

BF3. If φ *is a wff, then* $\neg\varphi$ *is a wff.*

BF4. If φ *and* ψ *are wff, then* $(\varphi \vee \psi)$ *is a wff.*

BF5. If φ *is a wff and* $a \in L$, *then* $[a]\varphi$ *is a wff.*

BF6. If φ *is a wff and* $a \in L$, *then* $[a]_i\varphi$ *is a wff.*

DEFINITION SUBFORMULA. *Given a formula* φ *in* $\mathbf{KPI}_L$ *we say that* ψ *is a subformula of* φ *if* ψ *belongs to the set* $Sub(\varphi)$ *defined recursively as:*

1. *For* $\varphi = \top$, $Sub(\varphi) = \{\top\}$.
2. *For* $\varphi = p_i$, $Sub(\varphi) = \{p_i\}$.
3. *For* $\varphi = \neg\psi$, $Sub(\varphi) = \{\neg\psi\} \cup Sub(\psi)$.
4. *For* $\varphi = (\psi \vee \theta)$, $Sub(\varphi) = \{(\psi \vee \theta)\} \cup Sub(\psi) \cup Sub(\theta)$.
5. *For* $\varphi = [a]\psi$, $Sub(\varphi) = \{[a]\psi\} \cup Sub(\psi)$.
6. *For* $\varphi = [a]_i\psi$, $Sub(\varphi) = \{[a]_i\psi\} \cup Sub(\psi)$.

We say that a formula have size n iff $n = \#Sub(\varphi)$.

DEFINITION Derived Operators. *In* $\mathbf{KPI}_L$ *we will use the following derived operators:*

Def $-$. $- =_{\text{def}} \neg\top$.
Def $\wedge$. $(\varphi \wedge \psi) =_{\text{def}} \neg(\neg\varphi \vee \neg\psi)$.
Def $\rightarrow$. $(\varphi \rightarrow \psi) =_{\text{def}} (\neg\varphi \vee \psi)$.
Def $\leftrightarrow$. $(\varphi \leftrightarrow \psi) =_{\text{def}} ((\varphi \rightarrow \psi) \vee (\psi \rightarrow \varphi))$.
Def $<a>$. *Let* $a \in L$, $<a>\varphi =_{\text{def}} \neg[a]\neg\varphi$.
Def $<a>_i$. *Let* $a \in L$, $<a>_i \varphi =_{\text{def}} \neg[a]_i\neg\varphi$.

DEFINITION Normal Systems. *A system of polimodal logic with inverse operators S of sort L is normal, if it can be specified through the following rules.*

1. *If φ is a PC-tautology or a substitution instances of a PC-tautology then φ is in S.*
2. *S has all substitution instance of the formulas*
   Ka. $[a](\varphi \rightarrow \psi) \rightarrow ([a]\varphi \rightarrow [a]\psi)$, *for any* $a \in L$.
   Ka$_\mathbf{i}$. $[a]_i(\varphi \rightarrow \psi) \rightarrow ([a]_i\varphi \rightarrow [a]_i\psi)$, *for any* $a \in L$.
   It1. $\varphi \rightarrow [a] <a>_i \varphi$, *for any* $a \in L$.
   It2. $\varphi \rightarrow [a]_i <a> \varphi$, *for any* $a \in L$.
3. *S is closed by the following derivation rules*
   MP. *If φ is in S and $\varphi \rightarrow \psi$ is in S, then ψ is in S. (Modus Ponens).*
   Na. *If φ is in S then $[a]\varphi$ is in S, for $a \in L$. (Necessitation).*
   Na$_\mathbf{i}$. *If φ is in S then $[a]_i\varphi$ is in S, for $a \in L$. (Inverse Necessitation).*

DEFINITION Kripke Model. *A Kripke model for* $\mathbf{KPI}_L$ *is a t-uple* $M =< W, \{-a \rightarrow \mid a \in L\}, V >$ *where:*

1. *W is a non empty set.*
2. *L is a finite set.*
3. *Each $-a \rightarrow$, with $a \in L$ is a relation on $W \times W$.*
4. *V is a modal valuation ($V : \mathbb{P} \times W \mapsto \{0, 1\}$).*

DEFINITION Validity in Possible Worlds. *Given a model* $M =< W, \{-a \rightarrow \mid a \in L\}, V >$ *the notion of validity in possible worlds is defined recursively as:*

1. $M \models_m \top$.
2. $M \models_m p_i$ *iff* $V(p_i, m) = 1$, *with p_i a propositional symbol.*
3. $M \models_m \neg\varphi$ *iff it is not the case that* $M \models_m \varphi$.
4. $M \models_m (\varphi \vee \psi)$ *iff* $M \models_m \varphi$ *or* $M \models_m \psi$.
5. $M \models_m [a]\varphi$ *iff for any world m' in W, $m - a \rightarrow m'$ implies* $M \models_{m'} \varphi$.
6. $M \models_m [a]_i\varphi$ *iff for any world m' in W, $m' - a \rightarrow m$ implies* $M \models_{m'} \varphi$.

We say that a formula φ is valid in a model $M =< W, \{-a \rightarrow \mid a \in L\}, V >$ ($M \models \varphi$) *iff for any world $m \in W$, $M \models_m \varphi$. We say that a formula φ is valid in a class of models C ($\models_C \varphi$) iff for any model $M \in C$, $M \models \varphi$.*

DEFINITION L-Morphism. *Let M_1 and M_2 be two models such that* $M_1 =< W_1, \{-a \rightarrow \mid a \in L_1\}, V_1 >$ *and* $M_2 =< W_2, \{-a \rightarrow \mid a \in L_2\}, V_2 >$, *and L a set of labels, $f : W_1 \rightarrow W_2$ is an L-morphism from M_1 in M_2 if for any label $a \in L$, for any $m_i, m_j \in W_1$ holds that $m_i - a \rightarrow m_j$ implies $f(m_i) - a \rightarrow f(m_j)$.*

DEFINITION Label Set. *Let ? be a set of formulas we define $L(?)$ as the set of labels of the formulas in ?.*

DEFINITION ?-Morphism and ?-Filtration. *Let M_1 and M_2 be to models such that* $M_1 =< W_1, \{-a \rightarrow \mid a \in L_1\}, V_1 >$ *and* $M_2 =< W_2, \{-a \rightarrow \mid a \in L_2\}, V_2 >$, *and ? a set of formulas of the language of M_1 ($L(?) \subseteq L_1$).*

1. *A ?-morphism from M_1 in M_2 is an $L(?)$-morphism $f : W_1 \rightarrow W_2$ such that for any propositional variable $p_i \in ?$ holds that $V_1(p_i, m_j) = 1$ implies $V_2(p_i, f(m_j)) = 1$.*
2. *A ?-filtration from M_1 in M_2 is a ?-morphism f such that:*

(Sur) f is surjective.
(Var) For any propositional variable $p_i \in ?$ 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 $[a]\varphi \in ?$ holds that $M_1 \models_{m_j} [a]\varphi$ implies $M_1 \models_{m_k} \varphi$ for any pair of worlds m_j, m_k in W_1 such that $f(m_j) - a \rightarrow f(m_k)$ holds in the model M_2.
(Fil$_i$) For any label a and formula φ such that $[a]_i\varphi \in ?$ holds that $M_1 \models_{m_j} [a]_i\varphi$ implies $M_1 \models_{m_k} \varphi$ for any pair of worlds m_j, m_k in W_1 such that $f(m_k) - a \rightarrow f(m_j)$ holds in the model M_2.

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

THEOREM PRESERVATION BY ?-FILTRATIONS. *Let ? be a set of formulas closed by subformulas, and let f be a ?-filtration from $M_1 =< W_1, \{-a \rightarrow \mid a \in L_1\}, V_1 >$ in $M_2 =< W_2, \{-a \rightarrow \mid a \in L_2\}, V_2 >$. For any formula $\varphi \in ?$, for any world $m_i \in W_1$ holds that $M_1 \models_{m_i} \varphi$ if and only if $M_2 \models_{f(m_i)} \varphi$.*

The last result shows how the existence of a ?-filtration between two models warrants that the formulas of ? are "treated equivalently" in both models, despite of the differences that might be between them. In other words ?-filtrations are the tools that let us decide when two models are equivalent relative to a set of formulas. Aside from its theoretic interest in the proof of the decidability of the logic **KPI**, we will see in the next sections which is the real importance of this method from a practical point of view.

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 polimodal logic and the minimal temporal logic, and preservation results (these results can be found for example in [Popkorn, S. 1994]).

THEOREM SOUNDNESS, COMPLETENESS AND DECIDABILITY. *The system* **KPI** *is adequate and decidable in the class of all models.*

3. STATE OF THE ART IN VERIFICATION OF PROPERTIES

The use of modal logics in the verification of properties is not an original idea in the area of Computer Science. Different logics have been proposed to deal with the verification of algorithms, specially, concurrentt algorithms (PTL, Propositional Temporal Logic [Pnueli, A. 1977]; TLA, Temporal Logic of Actions [Lamport, L. 1994]; CTL, Computational Tree Logic [Clarke, E. et al. 1986], etc.). To prove that a program satisfies its specification, expressed as a temporal or modal formula, is equivalent to prove that the formula is true at a state (or set of states) or in a set of runs in the transition system that models the program.

Each logic was designed to tack a specific problem, but in general, they can be classified in two big families. PTL ad TLA, for example, try to characterize the algorithm syntactically through a set of formulas, and then prove that this characterization implies the properties to verify. The proposal of Manna and Pnueli for PTL uses proof theoretic techniques. They proposed that the program, via its models, had to be codified as a theory (a set of temporal formulas) ?. The required property must then follow from, or be a temporal consequence of ?. A correct and complete axiomatization of the formulas that are theorems of the logic gives the frame in which these deductions can be presented. This method sees the transition system of a program as secondary, and it will be put aside once the theory ? is obtained. Lamport's basic idea for TLA is similar (proof theory methods) but in this case the modalities are defined over an action logic (an not over a propositional logic), which gives the object language greater expressive power. In this way things like assignment to variables, for example, can be represented directly in the logic language, greatly simplifying the characterization work.

CTL on the other hand, let us only *enunciate* properties that are verified over an specific model of the algorithm (represented by its finite state transition system). This is a more direct method to establish that a set of states or runs has a given property, and is based in tableaux techniques. This method is the support of the *model checkers*, algorithms that automatically establish if a given property is true in a finite state system.

The principal difference between this two approaches is that the first is based on a syntactic characterization of the program (through a set of formulas) while the techniques used by the second are mainly semantic, using an specific model of the problem to work on and the semantic definition of the operators. Although the axiomatic approach seems to be more abstract and generic, it must struggle with the complexity that characterizes the symbolic proof of theorems. The automatic symbolic provers that nowadays exists, need in general the aid of the specialist to decide in crucial steps of the proof and it is usually necessary a high degree of skill to organize the proof in a convenient way. The semantic approach needs only check that certain formula is valid in the model, using specific algorithms for this purpose (the model checkers). The validity problem in a model in modal logics used in this area is always decidable and of low complexity, which permits the development of efficient checking algorithms.

The logic systems mentioned above are classical examples in each area. In the last years a great variety of modifications, restrictions and new languages that can in one or other way be fitted in the families described above have appeared. In any case, the last advances in verification of properties using modal logics have been developed using semantic techniques, corresponding in this way to the predominance of the semantic approach in all the modern Logic. The main problem the model checking area must confront is the combinatorial explosion in the number of states that occurs during the analysis of concurrentt programs [Henzinger et al. 1994]. New techniques that reduce the number of states in the model, and optimized algorithms let now verify properties over systems of millions of states, and this limit is more than enough for our purposes given that a design will never excess this magnitude.

The logic **KPI** was defined having in mind this predominance, and that is why it responds adequately to the semantic techniques of validation, being less useful as a pure syntactic technique.

4. MODAL LOGIC AS A SOFTWARE ENGINEERING TOOL

4.1 Motivation: Is Modal Logic adequate to Software Design?

We will give as introduction to the new design proposals a more detailed presentation of our motivation to use modal logic in software engineering:

The first important point of contact we found between both areas was the similarities among the models used in each one. What in design was represented as a module with certain characteristics, like a name X, a number of version Y, etc., can be seen as a possible world in which the properties "my name is X" and "my version is Y" were valid, while the interaction relations between the modules were no more than accessibility relations. Nevertheless, a mere similitude in the structures used was not enough to justify the introduction of all the modal apparatus. The use given to both structures should also be similar. We asked ourselves which was the fundamental concept in both structures, and the answer was: the relations. They were the ones who played in both the main role; determining the properties of the model and a great part of the information that can be derived from it. If the modal logic could be adapted in a natural and simple way to manipulate the designs themselves, we will obtain a technique that would unify the language used to characterize the models and their properties, with formal proof capacity, a clear semantics and the possibility of automatic verification. The logic **KPI** obtained from the minimal logic **K** through traditional extensions, has all these characteristics.

4.2 New Design Proposals

As we said above, the principal technique used nowadays in the verification of properties is model checking. In relation with this task some techniques will be developed which will let us manipulate the models to check. Finally we will discuss briefly the possibility of a syntactic approach, pointing problems that it involves.

During all 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 [Parnas, D. L. 1972] 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 an alphabetically ordered list of all the circular shift of all the lines. Various module decompositions have been proposed for this problem, [Garlan, D. and Shaw, M. 1993]; in this section we will use a decomposition based in abstract data types (ADTs).

The decomposition consist of a principal control component, and other five principal components that in turn are divided in submodules through which the data types will be accessed:

Input. This module reads the lines from the input media and stores them internally.

Characters. Provides functions to access characters in words that form the lines and to count the number of words in a line.

Circular_Shift. Provides functions like those in the Characters module but in this case working not only over the input lines, but over any possible shift.

Alphabetic_Shift. This module consists mainly of two functions: Alphabetizer alphabetically orders the lines and I-th returns the index of the circular shift in the i-th position in the alphabetic order.

Output. This module will give as output the set of circular shifts of the lines.

In this decomposition there are three basic relations between the components: *is_part_of*, *invokes* and *I/O*.

We have already said that the most natural way to describe a design is through a graph in which a symbol (generally a box or a rectangle) represents the components, and lines connecting them represents the relations between them, as in figure 1.

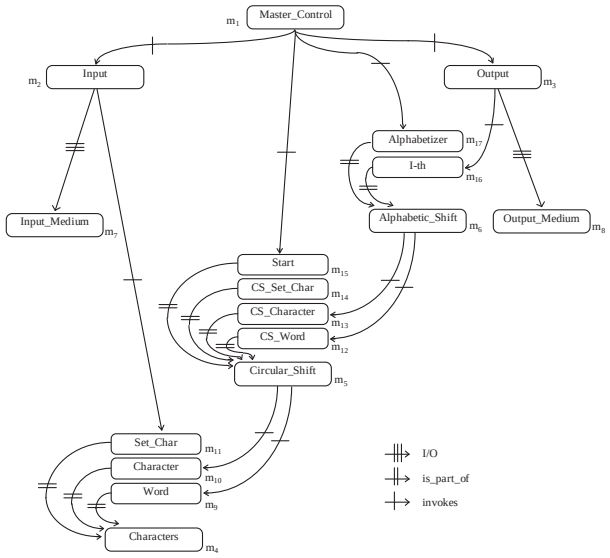

Fig. 1. KWIC Design

4.2.1 *Design Graph.* We can also give a formal definition of a design graph:

DEFINITION SCHEME OF DESIGN. *A general scheme of design is defined as a labeled directed graph* $< N, E, LN, LE, R_{LN}, R_{LE} >$, *where:*

1. N *is a set of Nodes (finite, non empty).*
2. E *is a set of Edges* ($E \subseteq N \times N$).
3. LN *and* LE *are sets of labels (disjoint and finite).*
4. R_{LN} *is the total function in* LN *that assigns a label to each node* ($R_{LN} : N \rightarrow LN$).
5. 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 *nodes 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}.

Example: The graph associated to the design in figure 1 would be $G =< N, E, LN, LE, R_{LN}, R_{LE} >$ with:

$$N = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17\}$$

$$\begin{aligned} E = \{ & (1,2), (1,3), (1,15), (1,17), (2,7), (2,11), (3,8), (3,16), \\ & (5,9), (5,10), (6,12), (6,13), (9,4), (10,4), (11,4), (12,5), \\ & (13,5), (14,5), (15,5), (16,6), (17,6) \} \end{aligned}$$

$$\begin{aligned} LN = \{ & Master\_Control, Input, Output, Characters, \\ & Circular\_Shift, Alphabetic\_Shift, Input\_Medium, Output\_Medium, \\ & Word, Character, Set\_Char, CS\_Word, \\ & CS\_Character, CS\_Set\_Char, Start, I-th, \\ & Alphabetizer \} \end{aligned}$$

$$LE = \{/, //, ///\}$$

$$\begin{aligned} R_{LN} = \{ & (1, Master\_Control), (2, Input), (3, Output), \\ & (4, Characters), (5, Circular\_Shift), (6, Alphabetic\_Shift), \\ & (7, Input\_Medium), (8, Output\_Medium), (9, Word), \\ & (10, Character), (11, Set\_Char), (12, CS\_Word), \\ & (13, CS\_Character), (14, CS\_Set\_Char), (15, Start), \\ & (16, I-th), (17, Alphabetizer) \} \end{aligned}$$

$$\begin{array}{rcl} R_{LE} & = & \{\ ((1,2),/),\quad ((1,3),/),\quad ((1,15),/),\quad ((1,17),/),\quad ((2,7),///),\\ & & ((2,11),/),\quad ((3,8),///),\quad ((3,16),/),\quad ((5,9),/),\quad ((5,10),/),\\ & & ((6,12),/),\quad ((6,13),/),\quad ((9,4),//),\quad ((10,4),//),\quad ((11,4),//),\\ & & ((12,5),//),\quad ((13,5),//),\quad ((14,5),//),\quad ((15,5),//),\quad ((16,6),//),\\ & & ((17,6),//)\ \} \end{array}$$

In this design the relation $-/\rightarrow$ represents *invokes*, $-//\rightarrow$ represents *is_part_of* and $-///\rightarrow$ represents *I/O*. □

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

4.2.2 *From Design Graph to Modal Model.* Given that we will work with modal logics, we have to transform the graph that represents an specific design to the new formalism.

DEFINITION Model Associated to a Design Graph. *Given a graph G corresponding to a design, $G =< N, E, LN, LE, R_{LN}, R_{LE} >$, following the definition introduced in the last section, we build the associated Kripke model, as $M =< W, \{-a\rightarrow \mid a \in L\}, V >$ 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. *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 Correctness of Translation. *If G is a design graph, M obtained from G is a Kripke model.*

The definition of design graph implies that the number of propositional variables that are valid in a world in the model is finite and coincides with the number of labels that were assigned to nodes in the graph. We can interpret the propositional variable p_{ln} as saying "The name of the module is ln".

Example: Starting from the design graph obtained from figure 1 we arrive to the model $M =< W, \{-/\rightarrow, -//\rightarrow, -///\rightarrow\}, V >$ with:

$$\begin{array}{rcl} W & = & \{\ m_1,\quad m_2,\quad m_3,\quad m_4,\quad m_5,\quad m_6,\quad m_7,\quad m_8,\quad m_9,\\ & & m_{10},\quad m_{11},\quad m_{12},\quad m_{13},\quad m_{14},\quad m_{15},\quad m_{16},\quad m_{17}\ \} \end{array}$$

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

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

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

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

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

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

4.2.3 *Verifying Properties in the Model.* We will give now the algorithm that let us check the validity of a **KPI** formula in a model. This algorithm is a modification of the algorithm in [Clarke, E. et al. 1986] to the operators in **KPI**.

Suppose that we have to determine the validity of a formula φ in a model 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 subformula of size n will have been determined and the subformula could be interpreted as a new propositional symbol p_i. The algorithm builds for each world m_j, a set C_j formed by the subformulas of φ valid in that world. As φ is subformula of itself, when the algorithm stops, φ would be valid in m_j if and only if $\varphi \in C_j$.

DEFINITION Model Checking Algorithm. *Given a formula φ of* **KPI** *and a model $M =< W, \{-a\rightarrow \mid a \in L\}, V >$ finite, the following 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, do
   $C_j = \emptyset$
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 $(\psi =<a>\theta$ & $((\exists m_k)(m_j - a \rightarrow m_k$ & $\theta \in C_k)))$ then $C_j := C_j \cup \{<a>\theta\}$
   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 $(\psi =<a>_i\theta$ & $((\exists m_k)(m_k - a \rightarrow m_j$ & $\theta \in C_k)))$ then $C_j := C_j \cup \{<a>_i\theta\}$

The following Proposition shows the correctness of the last algorithm.

PROPOSITION CORRECTNESS OF THE MODEL CHECKING ALGORITHM. *The Model Checking Algorithm stops for any formula φ of* **KPI**, *and upon termination for any subformula $\psi \in Sub(\varphi)$, $M \models_{m_j} \psi$ iff $\psi \in C_j$.*

Although the aim of our work is not to obtain optimal algorithms but to present a formal methodology, 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 M and e is the number of edges in M. Note that the truth value that M assigns to each subformula of φ (t in total) must be found, and that for each subformula, n steps will be needed to analyze what is happening in each world (if it is a propositional symbol, or a truth functional operator) or $n.e$ steps to analyze in each world all the edges that are refer to it (if it is a modal operator). The complexity of our algorithm is lesser than that of the algorithm presented in [Clarke, E. et al. 1986] due to the change 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 presuppose over the accessibility relations (transitivity, reflexivity, symmetry, etc.), the basic algorithm can be adapted to take into account these considerations. If this property were transitivity for example, our algorithm would be the direct transcription of Clarke's to our operators.

Example: Starting again from the model M corresponding to the design of the KWIC, let's verify the following property φ = "An ADT uses only services from others ADTs". In our model we can characterize ADTs as those modules that defines functions, then if a world corresponds to an ADT it must satisfy $<//>_i \top$. 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 $[/] <//> \top$. We must then verify in our model that $\varphi = <//>_i \top \rightarrow [/] <//> \top$ 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).

We will give now, the constructions made by our algorithm. Eliminating the abbreviations, the formula to check is $[//]_i\neg\top \vee [/]\neg[//]\neg\top$:

Size: 1, Sub: $\top$,
$C_j = \{\top\}$ for any m_j in W.
Size: 2, Sub: $\neg\top$,
$C_j = \{\top\}$ for any m_j in W.
Size: 3, Sub: $[//]_i\neg\top$,
$C_j = \{\top\}$ for j = 4 to 6,
$C_j = \{\top, [//]_i\neg\top\}$ otherwise.
Sub: $[//]\neg\top$,
$C_j = \{\top, [//]_i\neg\top, [//]\neg\top\}$ for j = 1, 2, 3, 7 and 8,
$C_j = \{\top, [//]\neg\top\}$ for j = 4 to 6,
$C_j = \{\top, [//]_i\neg\top\}$ otherwise.
Size: 4, Sub: $\neg[//]\neg\top$,
$C_j = \{\top, [//]_i\neg\top, [//]\neg\top\}$ for j = 1, 2, 3, 7 and 8,
$C_j = \{\top, [//]\neg\top\}$ for j = 4 to 6,
$C_j = \{\top, [//]_i\neg\top, \neg[//]\neg\top\}$ otherwise.

Size: 5, Sub: $[/]\neg[//]\neg\top$,
$C_j = \{\top, [//]_i\neg\top, [//]\neg\top\}$ for j = 1,
$C_j = \{\top, [//]_i\neg\top, [//]\neg\top, [/]\neg[//]\neg\top\}$ for j = 2, 3, 7 and 8,
$C_j = \{\top, [//]\neg\top, [/]\neg[//]\neg\top\}$ for j = 4 to 6,
$C_j = \{\top, [//]_i\neg\top, \neg[//]\neg\top, [/]\neg[//]\neg\top\}$ otherwise.
The formula φ, of size 6, is valid in all worlds because at least one of the terms in the disjunction is valid in any world. □

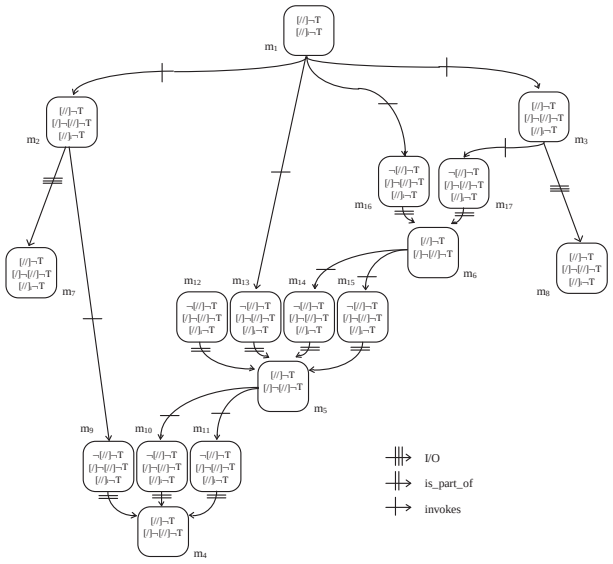

Fig. 2. Result of the model checking algorithm for the formula $\langle//\rangle_i \top \rightarrow [/] \langle//\rangle \top$

4.2.4 *Model Expansion.* The verification of a design is usually associated to the proof of properties about the relations involved. As we said above, given the habitual 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, the relations are associated to the modal operators of the logic. That is why the equivalent to create new relations is the definition of new modalities.

From a strictly logical and modal point of view, the idea of defining new modalities in the same logic is strange. In general, the incorporation of a new modality, like the operators P and F of past and future we commented above, gives rise to a new logic that tries to model it (syntactically and semantically) and is motivated by the search of grater expressivity power or versatility in the language. But this is not the case that interest us, 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 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.

The only references to a similar goal are found in the Theory of Explicit Definition of Beth in first order classical logic [Beth, E. 1953]. We do not know any work about this topic in the modal logic framework.

Beth's theory describes how new relational, functional and constant symbols can be added to a signature S of a first order language forming in this way 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 M corresponding to the original signature S, a model M^+ corresponding to the extended signature S^+ such that any formula in the original language preserves its validity and such that the formulas that use the new symbols preserve the meaning that was assigned to these shorthands via their definitions.

This result relies in the following lemmas:

LEMMA Expansion Lemma. *Let M (of signature S) and M^+ (of signature S^+) be first order structures, such that M^+ is a S^+-expansion of M, then for any formula $\varphi \in \mathcal{L}_S$ holds that, $M \models \varphi$ iff $M^+ \models \varphi$.*

LEMMA DEFINITION LEMMA [BETH, E. 1953]. *Let ? be a set of S-sentences and $S \subset S^+$. Let Δ be a set of S-definitions in ?, one for each symbol in $S^+ - S$. Given these conditions, for any S-structure M such that $M \models ?$ there exists a unique S^+-expansion M^+ such that $M^+ \models \Delta$.*

The language corresponding to the signature S^+ has all the formulas of the language of signature S plus formulas that use the new symbols in S^+. The expansion lemma tell us that the truth value of the formulas in $\mathcal{L}_S$ will be preserved in the expansion, while the definition lemma and the property of the first order logic that assures invariance under substitution by equivalents imply that any formula in $\mathcal{L}_{S+} - \mathcal{L}_S$ is equivalent to a formula in $\mathcal{L}_S$.

The importance of the definition lemma is shown principally in its *generality* (sentences in Δ need only to be definitions following the specification given by Beth) and in the result of *uniqueness* of the extension.

It seems that we can aspire to neither of these two qualities in a Theory of Definition for modal logic. In one hand, we cannot have uniqueness, as it is well known that because of the limitation in the expressivity of the modal languages there are always models structurally different to a given one that are nevertheless logically equivalent. In the other hand, it will be hard to find a specification of what is a *modal definition*: while in first order logic any formula $\varphi(x_1, ..., x_n)$, defines uniquely an n-ary relation $R(x_1, ..., x_n)$ (the relation formed by the n-uples $(a_1, ..., a_n)$ such that $\varphi(x_1, ..., x_n)[a_1, ..., a_n]$ holds), it is not true that any modal formula $\varphi(p_1, ..., p_n)$ defines a modality $M(p_1, ..., p_n)$ (or al 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 L^+-EXPANSION. *Let $M =< W, \{-a \rightarrow \mid a \in L\}, V >$ be a modal model, we say that the model $M^+ =< W^+, \{-a \rightarrow \mid a \in L^+\}, V^+ >$ is an L^+-expansion of M iff*

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

(that is to say, M^+ is identical to M, except that it can define new accessibility relations).

LEMMA MODAL EXPANSION LEMMA. *Let $M =< W, \{-a \rightarrow \mid a \in L\}, V >$ and $M^+ =< W^+, \{-a \rightarrow \mid a \in L^+\}, V^+ >$ be modal models, such that M^+ is an L^+-expansion of M, then for any formula $\varphi \in \mathcal{L}_L$ holds that for any $m \in W$, $M \models_m \varphi$ iff $M^+ \models_m \varphi$.*

This lemma proves that the formulas that do not use new symbols preserve their truth value from M to M^+, but what happen 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 the modal logic.

The following results lets us now for example, characterize the composition and union operations.

PROPOSITION COMPOSITION, $-c \rightarrow = -a \rightarrow \circ -b \rightarrow$. *Let $M =< W, \{-a \rightarrow \mid a \in L\}, V >$ be any modal model such that $a, b \in L$, then $M^+ =< W, \{-a \rightarrow \mid a \in L\} \cup \{-c \rightarrow \mid c \notin L$ and $m - c \rightarrow m'$ iff there exists m'' such that $m - a \rightarrow m''$ and $m'' - b \rightarrow m'\}, V >$ is an $L \cup \{c\}$-expansion of M which satisfies that for any $\varphi \in \mathcal{L}_{L \cup \{c\}}$, for any $m \in W$, $M^+ \models_m <c> \varphi \leftrightarrow <a><b> \varphi$.*

PROPOSITION UNION, $-c \rightarrow = -a \rightarrow \cup -b \rightarrow$. *Let $M =< W, \{-a \rightarrow \mid a \in L\}, V >$ be any modal model such that $a, b \in L$ then $M^+ =< W, \{-a \rightarrow \mid a \in L\} \cup \{-c \rightarrow \mid c \notin L$ and $m - c \rightarrow m'$ iff $m - a \rightarrow m'$ or $m - b \rightarrow m'\} >$ is an $L \cup \{c\}$-expansion of M which satisfies that for any $\varphi \in \mathcal{L}_{L \cup \{c\}}$, for any $m \in W$, $M^+ \models_m <c> \varphi \leftrightarrow (<a> \varphi \vee <b> \varphi)$.*

The characterization results on operations between relations are obviously limited by the expressivity of the modal logic. For example, given that the logic **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 expressivity power of the language with past operators. A similar result will make impossible to characterize in **KPI** the complement operation, as the logic with operator of inaccessibility (equivalent to accessibility over the complement of the relation) is more expressive than a logic with only accessibility operators.

PROPOSITION INVERSE, $-b\rightarrow = (-a\rightarrow)^{-1}$. *Let* $M =< W, \{-a\rightarrow \mid a \in L\}, V >$ *be any modal model such that* $a \in L$, *then* $M^+ =< W, \{-a\rightarrow \mid a \in L\} \cup \{-b\rightarrow \mid b \notin L$ *and* $m - b \rightarrow m'$ *iff* $m' - a \rightarrow m\}, V >$ *is an* $L \cup \{b\}$*-expansion of* M *which satisfies that for any* $\varphi \in \mathcal{L}_{L\cup\{b\}}$, *for any* $m \in W$, $M^+ \models_m <b> \varphi \leftrightarrow <a>_i \varphi$.

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

PROPOSITION INTERSECTION, $-c\rightarrow = -a\rightarrow \cap -b\rightarrow$. *Let* $M =< W, \{-a\rightarrow \mid a \in L\} >$ *be a model that corresponds to a design graph such that* $a, b \in L$ *then* $M^+ =< W, \{-a\rightarrow \mid a \in L\} \cup \{-c\rightarrow \mid c \notin L$ *and* $m - c \rightarrow m'$ *iff* $m - a \rightarrow m'$ *and* $m - b \rightarrow m'\} >$ *is an* $L \cup \{c\}$*-expansion of* M *which satisfies that for any* $\varphi \in \mathcal{L}_{L\cup\{c\}}$, $p \in I\!P$, $M^+ \models_m <c> (\varphi \wedge p) \leftrightarrow (<a> (\varphi \wedge p) \wedge <b> (\varphi \wedge p))$.

PROPOSITION SUBTRACTION, $-c\rightarrow = -a\rightarrow - - b\rightarrow$. *Let* $M =< W, \{-a\rightarrow \mid a \in L\} >$ *be a model that corresponds to a design graph such that* $a, b \in L$ *then* $M^+ =< W, \{-a\rightarrow \mid a \in L\} \cup \{-c\rightarrow \mid c \notin L$ *and* $m - c \rightarrow m'$ *iff* $m - a \rightarrow m'$ *and no* $m - b \rightarrow m'\} >$ *is an* $L \cup \{c\}$*-expansion of* M *which satisfies that for any* $\varphi \in \mathcal{L}_{L\cup\{c\}}$, $p \in I\!P$, $M^+ \models_m <c> (\varphi \wedge p) \leftrightarrow (<a> (\varphi \wedge p) \wedge \neg <b> (\varphi \vee p))$.

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* formula in the extended language. A formula $<c> \psi$ where $\psi \neq (\varphi \wedge p)$ has no 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 not more than a mere shorthand. Or in other words, the complete language $\mathcal{L}_{L\cup\{c\}}$, is in these cases potentially more expressive than $\mathcal{L}_L$.

Example: As an example of the application of this method let's define from the basic relations *invokes* and *is_part_of* in the KWIC model, the relation *uses_ADT* as $uses\_ADT = invokes \circ is\_part\_of$. We remind the definition of the original model
$M =< W, \{-/\rightarrow, -//\rightarrow, -///\rightarrow\}, V >$ as:

$$W = \{ m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, m_9, m_{10}, m_{11}, m_{12}, m_{13}, m_{14}, m_{15}, m_{16}, m_{17} \}$$

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

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

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

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

$V(p_i, m_j) = 0$ for any other pair (p_i, m_j)
With $-/\rightarrow$ the relation corresponding to *invokes*, $-//\rightarrow$ to *is_part_of* and $-///\rightarrow$ to *I/O*.
The model $M^+ =< W, \{\rightarrow, -/\rightarrow, -//\rightarrow, -///\rightarrow\}, V >$ has

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

With $\rightarrow$ corresponding to *uses_ADT*.
By the first Proposition then, the new model M^+ use <> as shorthand of </><//> providing furthermore the explicit relation corresponding to <> in the model. □

Another example of the application of this method over an algorithm of CRC can be found in [Areces, C. and Hirsch, D. 1996], where the hierarchy of the relation *uses* among modules was verified.

4.2.5 *Model Abstraction.* Even though the method of filtration was designed as a technique in the decidability proof of a logic, its concrete result is to help in the detection of a model simpler than the original one and that yet preserves the validity of the formulas that take part in the filtration. This is why, apart from the theoretic value of the method, we can use it as a purely practical instrument, applied to particular models.

The method of filtration let us define a new model that acts as an abstraction of the original model. This new model shows a simplified view, in which the details without interest have been eliminated while new more general and abstract concepts (that were hidden by those same details) have been highlighted.

We must make clear from the start, that obtaining simpler models through the method of filtration has not as aim to decrease the complexity of the property verification method. 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 for the task carried out and so included in the filtration set $?$. This new view provides conceptual information of a higher level than a simple validity result. The examples we are going to study show the potential of this method.

It calls the attention that an automatic method that can carry out this task actually exists. The abstractions obtained seem to require a good measure of cleverness and capacity, even though the filtered model does not always match the result that a designer would propose.

The algorithm that we specify below let us move from any model of design to an abstract view of it with the guaranty that a filtration could be defined for the set of formulas $?$ of interest.

DEFINITION FILTRATION ALGORITHM. *Let $M =< W, \{-a \rightarrow \mid a \in L\}, V >$ be a modal model and $?$ a finite set of formulas closed by subformulas, we define $M^* =< W^*, \{-a \rightarrow^* \mid a \in L^*\}, V^* >$ as the filtered model by $?$:*

Algorithm:

1. Run over M the model checking algorithm with formula $\bigwedge ?$.
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. $L^* := L(?)$
4. For any edge $m_i - a \rightarrow m_j$ and $a \in L^*$
   $[m_i] - a \rightarrow^* [m_j]$
5. For any $p_i \in ?$
   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 $p_i \notin ?$.

We must verify that the result of the last algorithm is really a model.

PROPOSITION CORRECTNESS OF THE FILTRATION ALGORITHM. *The filtration algorithm stops, and upon termination the result $M^* =< W^*, \{-a \rightarrow^* \mid a \in L^*\}, V^* >$ is a modal model.*

The algorithm we just gave builds from a model M and a set $?$ of formulas a model M^*, such that a $?$-filtration can be defined from M in M^*.

THEOREM FILTRATION EXISTENCE. *Let $M =< W, \{-a \rightarrow \mid a \in L\}, V >$ be a modal model and $M^* =< W^*, \{-a \rightarrow^* \mid a \in L^*\}, V^* >$ the model obtained from M for a set of formulas $?$ by the last algorithm. Then $f : W \rightarrow W^*$ such that $f(m_i) = [m_i]$ is a $?$-filtration from W in W^*.*

We can finish the study of the algorithm, obtaining its complexity order. The complexity of the first step is the complexity of the algorithm of model checking run over $\bigwedge ?$, that is $O(t.\max(n, n.e))$, where t is the size of $\bigwedge ?$, n the number of nodes in the model and e the total number of edges. Step 2 determines the equivalency classes and so has complexity n^2. Step 3 copies all the labels from $\bigwedge ?$ (at most t) to L^*. Step 4 copies the edges in the relations of M to the model M^*, at most e. Finally step 5 must define V^* for any world in W^* and for any propositional symbol $\bigwedge ?$, which takes a maximum of $t.n$ steps. A simple analysis shows that the algorithm has $O(t.\max(n, n.e) + n^2)$.

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

Example: Let's begin with a simple case. We will work over the model 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 $[uses\_ADT]\top$).

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

When we add to ? the propositional symbols and given that in our example each world 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 ?, step 4 will copy all the edges $-uses\_ADT \rightarrow$ from M^+ to $(M^+)^*$ while all the other relations will be ignored as they have no representatives in ?.

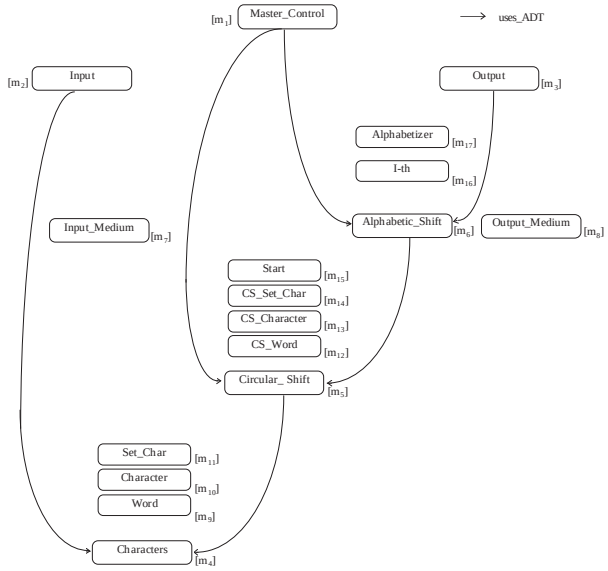

Fig. 3. Filtration of the model M^+ of the KWIC preserving the relation *uses_ADT*

As figure 3 shows, the filtered model preserves the relation we were interested in, and thanks to the fact that the definition method introduces the new relation explicitly and makes it independent of the other ones, all the other relations can be eliminated as irrelevant. □

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

Example: Looking at the last example we can see that some totally unconnected worlds were left in the filtration. These worlds are not relevant in 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 ?; given that they were mentioned, the filtration preserved them. We can define a new ? with the information obtained from the last filtration as $? = \{Input, Output, Master\_Control, Characters, Circular\_Shift, Alphabetic\_Shift, [uses\_ADT]\top\}$ and now, all the unconnected worlds will be collapsed in a single class that will represent them in the new abstraction. The result is shown in figure 4. □

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

Example: The examples above present rather simple applications of the filtration method. We consider the following one as a real sample of the potential of this method, letting us appreciate its qualities as a true technique for design abstraction.

Let us take again the original KWIC model and the formula we used in the model checking example and let $? = Sub(\langle//\rangle_i \top \rightarrow [/] \langle//\rangle \top)$. 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 5.

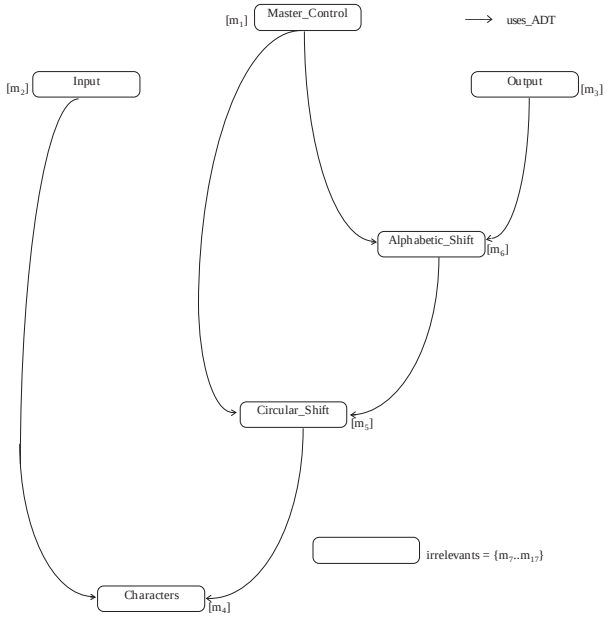

Fig. 4. Filtration of irrelevant modules

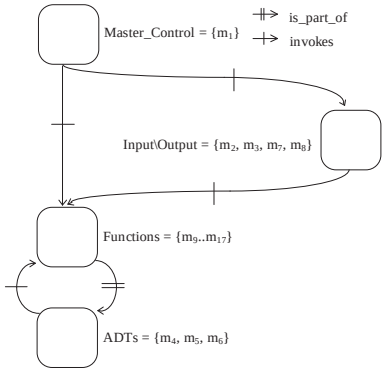

Fig. 5. Filtration of the KWIC model by property $\langle // \rangle_i \top \rightarrow [/] \langle // \rangle \top$

The modules of the original model were brought together in accordance to the role they play with respect to the formulas in ?. The subformulas of $<//>_i \top \rightarrow [/] <//> \top$ characterize the purpose of the different entities in the design. That is how we obtain the classes ADTs, Functions, I/O and Master_Control that we can now, thanks to the abstraction, clearly distinguish in the original model. In addition to the fact that, in this abstraction it is obvious that the ADTs only use functions that are part of others ADTs, 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.

4.2.6 *Syntactic Proofs.* Up to now, we have been using exclusively the model checking or semantic method to verify properties over a design. But, as we commented at the beginning of this section, a purely syntactical approach is also possible where the properties are derived using inference rules, from the axioms of the logic and a set of formulas that characterize uniquely the model.

To complete the analysis of the problem this work addresses, we will devote this section to show the inconveniences of trying to use the syntactic method with the logic **KPI**.

The syntactic method is based in the definition of a set of formulas that characterize uniquely the model we are interested in. But we must understand that "uniquely" is interpreted with respect to the expressivity of the language we are using. For example, given that the classical modal language do not let us speak of inaccessibility, the formulas valid in a world in a given model and those valid in the model that is obtained when we add a new unconnected world are the same, and then there is no set of formulas that let as distinguish between them. Nevertheless this is not a problem; we know beforehand the limitations of the modal language and we were only interested in working with those properties that *can* be expressed in the logic.

But we will have difficulties if, given two models with different properties (that can be expressed in the logic), we cannot build a description that differentiates one from the other, because if this is the case, we will only be able to prove syntactically the properties that both models have in common.

The general case of describing any model through a set of formulas is complex, but anyway that is not what we need, because models that corresponds to designs form a very particular class and we are looking for descriptions only for them. What makes the design models particular is the restriction imposed from the start over valuations. Given that valuations should represent the name of the modules in the design and that repeated modules are not allowed, each world satisfies one and only one propositional symbol that distinguish it. Furthermore, and given that connectivity cannot be expressed in the modal language, we can restrict ourselves to work with a connected component of the design at a time.

Let us think now, how we would describe in English a graph that had the characteristics of our models. The obvious way is to begin by any node saying "we start from X" and then describe the possible paths we can follow as "from X by an edge of type a we can go to Y and by an edge of type b to Z" and therefore continue from these nodes as "from Y we can go to ...". What we are trying to say is that it seems to be enough to include a node that mark the origin, and a formula that describes each edge of the model from this initial node (this can be done given that the model is connected) to obtain a complete description of the model. Let us do this in an example.

Example: Given the model of figure 6 we obtain from node m_1 the following description.

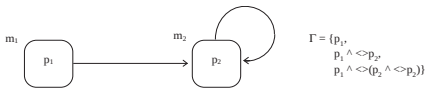

Fig. 6.

As each node m_i is identified by the propositional symbol p_i, the formula p_1 marks the initial node and then a formula is given for each edge of the model: $p_1 \wedge <> p_2$, says that starting from p_1 we can reach to p_2 by an edge, while the third formula says that from p_1 we can reach to p_2, from which we can keep walking by an edge to arrive again to p_2. Note that this description is not unique, and that we can construct others depending on the criteria we use to walk the graph (for example another description can be obtained changing the third formula to $(p_1 \wedge <> (p_2 \wedge <>_i p_2)))$. Even though the descriptions are multiple, the minimal model (in worlds and edges) that respects the restriction imposed over the valuations and satisfies all of them is the same: the model from which the descriptions were built. □

What is interesting about this method is that it can be automated [Hirsch, D. and Areces, C. 1995]. With this, it seems that all the problems are solved, because once we obtained the description of the

model (and furthermore, in an automatic way) we can verify a property φ over the design proving that φ can be derived from the set of formulas ? of the description in the axiomatic system **KPI** ($? \vdash_{KPI} \varphi$).

Given the restrictions we indicated above we have built a description that is only satisfied by the model from which we started. But we have not told the logic **KPI** *to take into account the restrictions.*

Example: Look at the following models that also satisfy the description in figure 6 from world m_1

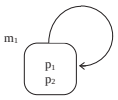

Fig. 7.

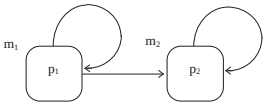

Fig. 8.

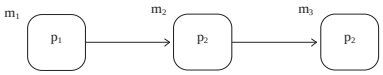

Fig. 9.

In figure 7 we have a model that satisfies in a world more than one propositional symbol. In figure 8 an extra arrow, not described by the formulas in ?, starts from world m_1; and finally, the model in figure 9 satisfies the formulas even though it has one more world, because the logic cannot distinguish between the loop in world m_2 in figure 6 and the edge that joins worlds m_2 and m_3 in figure 9.

This shows that the set of formulas we had called a description is not enough for the logic **KPI** to characterize uniquely the model of the design we wanted to verify. And in this way for example, the formula $\neg <> p_1$ that was satisfied by the original model in m_1 will not be obtained syntactically because the models in the figures 7 and 8 do not satisfy it.

But as we said above, the models we wanted to work on have characteristics that the models in figures 7, 8 and 9 do not have.

The model in figure 7 does not obey the restriction we imposed over the names of the modules. We will see below how we can tell the logic that these models must not be taken into account. The models in figures 8 and 9 show a different problem. In the descriptions we have been using (in English and in the modal language) we have always made the implicit assumption that *all the relevant information was being mentioned in the description and no more.* If we did not say explicitly that an specific edge joined two nodes, it should be interpreted that such edge did not exist, but as we see in figures 8 and 9, the logic **KPI** does not make such supposition. This minimality problem will be the main impediment that interferes with the use of the syntactic method in the verification of properties using **KPI**. □

To end this section, we will give the solution to the valuation problem and we will investigate more formally the complexities of the minimality problem.

Valuation

We must force in the logical system the condition saying that in each world a single propositional symbol is satisfied. In this way, models as those is figure 7 will not be accepted.

This condition can not be introduced axiomatically into a classical modal logic in the general case when the language has infinite propositional symbols, but this can be done if the language is finite. Note that even if the restriction to a finite language is not problematic from the point of view of the application (given that all our models are finite), it has quite an important effect from a logical point of view.[1]

[1] If we work outside the group of classical modal operators, we can find logics that add *nominal* symbols. These new atomic symbols has exactly the property we where talking about: to be valid exactly in a single point of the model and were introduced by [Blackburn, P. 1993]. In his work it is shown that the modal logic of <> is much less expressive than the extension obtained adding nominals.

Once we have restricted ourselves to languages with finite propositional symbols, we can give an axiomatization of this condition for each language $\mathcal{L}_n$ (where $\mathcal{L}_n$ is the original language $\mathcal{L}$ restricted to only n propositional symbols).

The axiom we need is $Vn.\bigvee_{(i\leq n)}(p_i \bigwedge_{(j\leq n, j\neq i)} \neg p_j)$.

This axiom achieves what we wanted: in all cases where we prove p_i for some propositional symbol, we will be able to derive $\neg p_j$ for any $j \neq i$. And with this (assuming correctness) models like those in figure 7 are not valid models, because, for example, in world m_1 it holds at the same time p_1 and p_2 from which, by the axiom Vn, we derive $\neg p_1$ and $\neg p_2$ obtaining an inconsistent valuation.

We can comment about some characteristics of the axiom Vn. First, Vn is an axiom, not an axiom scheme; that is, we do not consider as thesis of our logic all substitution instances of Vn, but only the formula Vn as it is written. A substitution rule will lead to an inconsistent logic, because, for example, a substitution rule would let us obtain $\{Vn\} \vdash (p_1 \wedge \neg p_1)$. In addition, it might be instructive to mention the semantic counterpart of axiom Vn. Semantically, the addition of this axiom is equivalent to the extension of the set of propositional tautologies, or in other words to the definition of a new relation of consequence that extends the classical notion. The new notion Cn' can be defined as the minimal extension to the classical notion that also verifies the rule $Cn'(\emptyset) = Cn(\{\bigvee_{(i\leq n)}(p_i \bigwedge_{(j\leq n, j\neq i)} \neg p_j)\})$.

In summary, the new logic proposed to solve the valuation problem is the system **KPIVn** axiomatized by:

PL. φ, where φ is an instance of PC-tautology.

Vn. $\bigvee_{(i\leq n)}(p_i \bigwedge_{(j\leq n, j\neq i)} \neg p_j)$.

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

Kai. Any instance of $[a]_i(\varphi \rightarrow \psi) \rightarrow ([a]_i\varphi \rightarrow [a]_i\psi)$, for any $a \in L$.

It1. Any instance of $\varphi \rightarrow [a] <a>_i \varphi$, for any $a \in L$.

It2. Any instance of $\varphi \rightarrow [a]_i <a> \varphi$, for any $a \in L$.

MP. From $\vdash_{KPIVn} \varphi \rightarrow \psi$ and $\vdash_{KPIVn} \varphi$, to obtain $\vdash_{KPIVn} \psi$.

Na. From $\vdash_{KPIVn} \varphi$, to obtain $\vdash_{KPIVn} [a]\varphi$, for any $a \in L$.

Nai. From $\vdash_{KPIVn} \varphi$, to obtain $\vdash_{KPIVn} [a]_i\varphi$, for any $a \in L$.

We will prove that this system is correct and complete with respect to the class of models we are interested.

THEOREM Correctness and Completeness of **KPIVn**. *The logic* **KPIVn** *is correct and complete relative to the class of models* $CVn = \{< W, \{-a \rightarrow \mid a \in L\}, V > \mid W \neq \emptyset$ *and for any* $m \in W$, *there exists* p_i *such that* $V(p_i, m) = 1$ *and for any* p_j, *with* $i \neq j$, $V(p_j, m) = 0\}$.

In this way we where able to introduce to the axiomatic system the restriction over valuations. Let's turn now to the minimality problem.

Minimality

Let's work in the example to see what characteristics presents the minimality problem.

We might think that the true problem is the way in which we built the description. But, it is easy to show that the minimality restriction cannot be captured adding more information (more modal formulas) to the description.

For example, take ? as the set that describes all the possible paths of the graphs (without edge repetition). This set incorporates more information than the one obtained by the original description criterion.

Example: If we take the model of figure 6, using the criterion above, we have a new description.

$? = \{p_1, p_1 \wedge <> p_2, p_1 \wedge <> (p_2 \wedge <> p_2), p_1 \wedge <> (p_2 \wedge <>_i p_2)\}$

Now the model of figure 9 does not satisfy this description (because it does not satisfy the last formula), but the model of figure 8 still satisfies all the formulas of ?. Anyway, we must not fool ourselves with this "improvement". In the general case, given any description (finite consistent set of formulas), we can built a non minimal model that satisfies it. If ? is finite, it describes the graph as it is seen from the initial world up to a fix distance d. Any model that match the original model inside this "radius" d will satisfy ?, no matter what happen after d steps. In the last argument, it is important for the set ? to be finite, but we cannot lighten this restriction because handling infinite sets would make the method intractable. □

The question then reminds open: is there no way to use the type of descriptions we initially proposed and find some means to add the minimality condition to the logic?. It seems quite certain that no

axiom can achieve this result (as we do with the valuation problem) and that others methods must be used. Perhaps we must use a proof method different from the standard (or semantically, a different consequence notion like the constructions used in nonmonotonic logics). The clues for the solution of this problem might perhaps be derived from the fact that the described model has the property of being the minimal model that satisfies the description and then is *initial* in its category. The problem is then how to translate the categorical notion of iniciality to the modal logic. There are some articles that link categories and temporal logic, one example is [Fiadeiro, J. et al. 1991].

5. APPLICATION EXAMPLES

This section will be devoted to the application of the new formal methods for design analysis discussed in the previous sections. We will comment two examples. One example shows a general method of how to detect the incorrect use of an interface module. The other was taken form the area of Object Oriented Design, and shows how the new methods can be applied on designs that have more information in the modules (not only their names), without modifying the formalism proposed.

5.1 Incorrect Use of Interface

One of the central notions of good design to obtain a correct modularization, 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. In this way lower coupling, higher reusability and, principally, adaptability to the system changes are achieved.

We can think of a library, as a set of submodules that are "administered" by an interface module that attends the external requirements and translates them to requirements for the different internal submodules. Figure 10 shows how we could represent the correct use of the interface of a module.

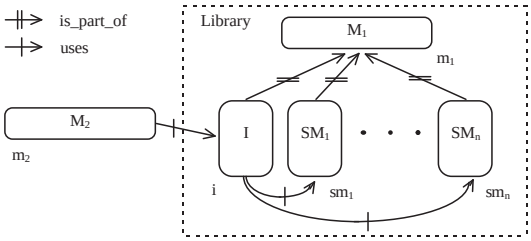

Fig. 10. Correct use of the interface of m_1 by m_2

Now, supposed that we have a design error, where the structure of figure 10 is not respected, as in figure 11. This means that m_2 accesses directly the submodule sm_1.

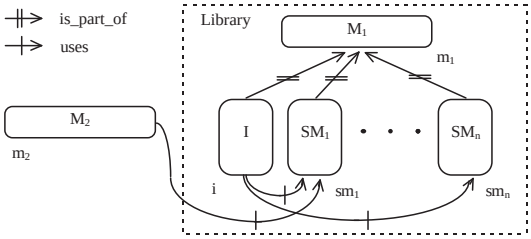

Fig. 11. m_2 uses the submodule sm_1, bypassing the interface of m_1

Comparing figures 10 and 11, it is easy to enunciate a formula that represents the correct use of the interface. Let begin with m_2 and see that m_2 is incorrectly using m_1 if it uses a part of m_1 that is not an interface, i. e. that is necessary that each time a part of m_1 is used it must be the interface: $[/](<//> M_1 \rightarrow I)$ (this can be read as M_2 is correctly using library M_1). This means that verifying (using the model checking algorithm, for example) $M \models_{m_2} [/](<//> M_1 \rightarrow I)$ proves that m_2 uses correctly m_1. It easy to see that this is not the case in figure 11.

The same property can be verified from m_1. The module m_1 is correctly used if all of its submodules satisfy that they are either the interface or are only invocated by the interface: $[//]_i(I \vee [/]_i I)$ (this can

be read as, the library M_1 is correctly used). Of course, this formula must be satisfied by m_1, not by m_2.

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

In this case, the property is characterized by the "structure" of the given formulas. This means that replacing the propositional symbols I and M_1 by adequate ones at each moment, does not alter the meaning of the formula.

5.2 Design in the Object Oriented Paradigm

This section is devoted to analyze an example from the Object Oriented Paradigm. The example (with minimal changes) we will next study can be found in [Wirfs-Brock, R. et al. 1990].

We will see by the techniques proposed in [Wirfs-Brock, R. et al. 1990], that it is usual to have much more information about each class in the models of design in de Object Oriented Paradigm. For example, the list of responsibilities that the class defines, the class type (abstract or concrete), etc. are usually included in the graph.

Our main interest is to show how we can handle the extra information in our formalism, using the ideas presented in [Bourdeau, R. and Cheng, B. 1995], by transforming the implicit information inside the class into explicit information.

In the design phase within the object oriented cycle of software development, the hierarchy graphs are used as a technique to model the inheritance relations between classes.

In this graphs, the nodes are classes and the relation between them is *is_subclass_of*, that defines the inheritance from superclass to subclass. In addition, the classes are marked as abstract (classes that do not possess instances, and are created to factor common properties) or concrete (classes that do possess instances, and will be the active objects during the execution of the program). Each class defines its responsibilities (the tasks they can perform and characterize their behavior) that, by the hierarchical relation, are inherited from superclass to subclass. 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 higher as possible in the hierarchy.
- Assure that the abstract classes do not inherited from concrete classes.
- Eliminate classes that do not add any functionality.

We will work with an example of a Drawing Editor Design. In particular, we will check the design of the hierarchy of the class Drawing_Element. The design of the class is presented on the next figure (a modification of figure 6-10 of [Wirfs-Brock, R. et al. 1990]):

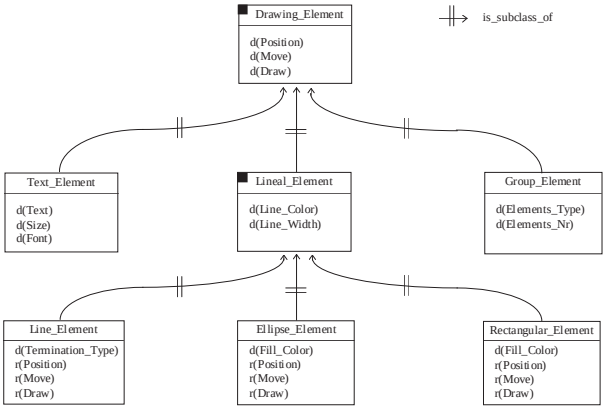

Fig. 12. Design of the class Drawing_Element

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 upper left corner).

As we said before, the responsibilities are inherited downward in the hierarchy. Each subclass specifies the *new* responsibilities that characterized it.

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

The flexibility that provides the use of accessibility relations allow us to incorporate all the additional information:

The monadic property *is_abstract_class* is no more than a subset S from the set of classes, that we can think of as the diagonal relation in $S \times S$. In our example $S =$ { Drawing_Element, Lineal_Element }, so the relation would be $-/ \rightarrow =$ { (Drawing_Element, Drawing_Element), (Lineal_Element, Lineal_Element)}. The properties of definition (d) and redefinition (r) are dyadic properties that relate classes with responsibilities and so can be directly translated as accessibility relations ($-d \rightarrow$ and $-r \rightarrow$), once a world has been assigned to each responsibility.

Let us see the result of this translation:

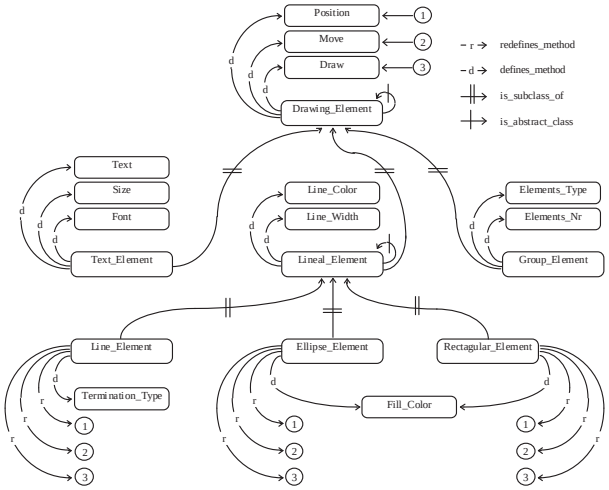

Fig. 13. Modal Model of the class Drawing_Element

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

Model the hierarchy is_subclass_of: A class should be a subclass of another one only if it supports all the responsibilities defined by the second one. To model the relation among classes has two purposes: one is to assure that it is a hierarchy, and the other is to help to verify that subclasses support at least all the responsibilities of their superclasses. This increases their reusability because it is easier to observe where a new class can be inserted in the old hierarchy.

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

The verification of the hierarchy is similar to the one done in the CRC example in [Areces, C. and Hirsch, D. 1996], where the model checking algorithm was used to detect a cycle in the hierarchy of the relation *use* among modules. In this case it has to be done over the relation is_subclass_of.

Factor common responsibilities as higher as possible in the hierarchy: If a set of classes supports a common responsibility, they should inherit this responsibility from a common 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. The main benefit of designing the hierarchy with abstract classes is observed when you want to incorporate a new functionality to an already existing application.

In our example, the verification of correct factorization can be done in the following way: Let's define a set ? of formulas as $? = \{(<d> p_i \wedge p_j) \rightarrow <//> [//]_i(<d> p_i \rightarrow p_j) | \ p_j$ 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 ?, for example $(<d> Size \wedge Text\_Element) \rightarrow <//> [//]_i(<d> Size \rightarrow Text\_Element)$ 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 is necessary that the definition of Size should

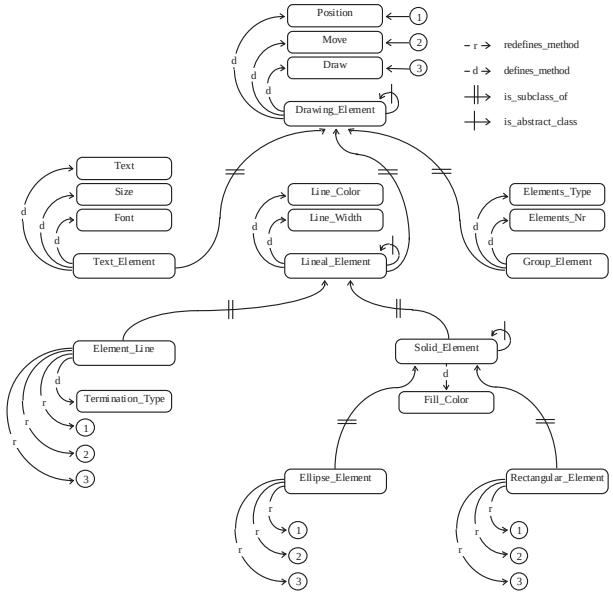

Fig. 14. Factorization of the abstract class Solid_Element

be done in the class Text_Element). If the model satisfies all the formulas in ?, we can assure that we have a good factorization. If not, the formulas that were not satisfied will show error cases.

In figure 13 we can check that the classes Ellipse_Element and Rectangular_Element share the responsibility Fill_Color and for this reason it will be factored in a new abstract class Solid_Element.

Assure that abstract classes do not inherit from concrete classes: For their nature, abstract classes support their responsibilities independently from the implementation. Therefore, they never have to inherit from concrete classes, which can specifically depend of the implementation. If a design has that kind of inheritance, the problem can be solved creating another abstract class from which the abstract as well as the concrete class can inherit their common behavior.

In the modal language the property to check says that it is necessary for any abstract class that its superclasses are abstract classes too: $\langle/\rangle \top \rightarrow [//] \langle/\rangle \top$ with the relation $-// \rightarrow$ transitive. The model checking algorithm will verify that the property is valid for the model of our example.

Eliminate classes that do not add any functionality: The classes that do not have responsibilities must be eliminated. If a class inherit a responsibility that will implement in a unique way, then it adds functionality in spite of not have responsibilities of its own, and it is not eliminated. On the other way, abstract classes that do not define or redefine responsibilities have no use and must be eliminated, because they will not increment reusability.

The last rule can be enunciated as the obligation of all classes to share part of the knowledge of the object that is defined. Or in other words, that all classes must cooperate in the definition or redefinition of the responsibilities that characterized the object hierarchy.

This property is simple to enunciate as $\langle d\rangle \top \vee \langle r\rangle \top$, and must be valid in the model. The classes that do not obey this property must be eliminated.

Note that the last property allows abstract classes to redefine responsibilities. It is not incorrect for an abstract class to have implementation code in spite that usually they only define the interfaces of the responsibilities for their subclasses. And yet, the designer has to take care that the code incorporated to an abstract class do not damage its condition of it.

Therefore, it can b useful to detect which are the classes that incorporates implementation code. This can be done checking the formula $\langle/\rangle \top \wedge \langle r\rangle \top$. The classes that satisfy this formula have to be review because of possible problems.

The application of our methods to the example of object oriented design, shows that the restriction over the valuations impose to the models of design does not limit in an excessive way the formalism. And we also believed that the restricted models are simpler to manipulate (even if they are of bigger in size than the models corresponding to the usual hierarchy graphs) and the use of explicit relations has in our case more relevance, because our description language contains modal operators that will be their syntactical counterpart.

6. FUTURE WORK

This work opens several possibilities of future research, purely theoretical in the area of Modal Logic and also applied to the area of Software Engineering:

The Theory of Modal Definition we begin to describe in this work, must be completed with an analysis of the interrelation that exists between the expressive power of the language and its capacity to characterize different operations on relations. It would be also important to find a general case of modal definition that avoid working with particular cases as we did. The cases of partial definitions that we have shown give rise to the question of which is exactly the expressive power the extended languages possess.

Remain of course open the questions we have already outlined during the syntactic analysis. On one hand to study how we can use a modal logic with nominals in the characterization of a model of design (valuation problem); on the other hand, the link between modal logic and category theory to capture the minimality restriction in the description of models (minimality problem).

A work strictly related to the area of complexity and optimization can also be accomplished, implementing better algorithms than the ones we proposed. An improvement of the prototype presented in [Areces, C. and Hirsch, D. 1996] through the introduction of a graphic interface would be also very desirable.

The application examples we studied in this work were elected to demonstrate the power of the techniques proposed, and also because they are cases known in the literature. We believe they do not constitute trivial examples, but in any case it would be interesting to apply our techniques in more and more varied cases. This would let us find new properties to verify and to investigate if the techniques are sufficient to handle them. It should be noted that the search for new application examples should not necessarily be restricted to the area of design. Areas as that of the System Configuration Management for example seem to adapt perfectly to our formalism.

As a side point, the real power as an abstraction method of the filtration technique should be studied. The results exhibited are encouraging and they should be analyzed in-depth.

The last example presented shows a case in which it is not necessary to modify the methodology proposed to handle models with some information in the modules. This does not mean the transformation could be carried out in all the cases. The formalism should be modified to cover other types of information such as constraints on the implementation.

Very related to the area of Software Design is the area of Software Architecture. It is possible that the present work can be applicable also to the description of architectures. In this area however, the problem has greater complexity. The graphs used to represent architectures seem to capture correctly the static components of the system, but not so the dynamic component that defines the interaction among the components [Garlan, D. and Shaw, M. 1993; Allen, R. and Garlan, D. 1994]. In these graphs, the relationships have a fundamental importance. For example, from the fact that two components are in the client-server relation, we can deduce part of the behavior of both components and the form in which they communicate (the client sends requests to the server in any moment, the server answers the requests without knowing who makes the requests, etc.). There exist some current work in this area, for example in [Dean, T. and Cordy, J. 1995] where a (non modal) syntactic language for the description of architectures is presented.

7. CONCLUSIONS

In this work we present a methodology for the representation of software system designs and for the verification of properties over them. The graphs of design are considered models of an extended modal logic; and the methods used to verify properties are developed from techniques characteristic in classical modal logic, extended to cover the differences in the modified formalism. We present the procedures or logic techniques to derive the modal model associated with a given design, the algorithm to check properties, the expansion technique to define new relationships and the model filtration method. These methods are demonstrated in two application examples of various type.

Given the kind of the work, we can offer conclusions from two points of view: the strictly modal work and the work of application to the specific topic of verification of properties in the design.

In the first case, the work contains some results of importance. A polimodal logic with inverse operator (**KPI**) is presented in order to use it as a property specification language to verify through a model checking algorithm. The existing verification languages (usually in the concurrent programming area) do not require inverse operators (though in certain cases quantification operators on branches of the model are used) since they work on the run of the program and in these models the future operators are enough for the description of the progress of the system. In the designs (that by definition are connected), the

inverse operators permit the access to all the other modules from any point.

The logic **KPI** is in truth a family of logics, each one characterized by the sort of labels that defines its alphabet. We devoted special atention to the problems that arise handling models of different signatures. In the literature, a fixed sort of labels to which answer all the models of the class, is usually assumed. In relation with this point, the definition of filtrations is now a morphism among models of different sorts (reduction of the set of labels) and the definition technique incorporates the concept of L-expansion of a model (extension of the set of labels). The results related to the Theory of Modal Definition presented in this work are only those necessary for the application in hand. We were not able to find references to similar work, which encourages further research in this last topic.

Finally, the limitations of the logic **KPI** for the syntactic proof of properties of design, were analyzed. These limitations are not corrected in the finite restriction **KPIVn** (and hypothetically they would neither be solved adding nominals) due to minimality problem in the characterization of models of design.

The work in the area of Software Engineering can be summarized in the following points:

From the similarity among the models used in two different areas, an specific logic for the use in the verification and description of design was defined. Then we proceeded to gather the basic notions in the literature of good design accepted in general form and we attempted their translation to modal notions that could be automatically checked on the models of design.

The obtained result was a formal language of specification of properties, an automatic checking algorithm and the design of a set of techniques that permits the manipulation of the models.

This set of techniques is really an adaptation of traditional methods from (classic and modal) logic.

We emphasize how easily these methods, once extended and adapted to the application, permit to accomplish the tasks that are commonly found in the process of Software Engineering. Specifically, the use of filtrations for the obtention of design abstractions and the use of the theory of modal definition for the definition of derived relationships.

As shown in the application examples, the methods proposed seem to be effective, but beyond the examples, the application of these methods (via automatic tools) is fundamental when a design has hundreds of components. The methodology was implemented using the functional language SML-NJ in [Areces, C. and Hirsch, D. 1996].

8. APPENDIX

THEOREM PRESERVATION BY ?-FILTRATION. *Let ? be a set of formulas closed by subformulas, and let f be a ?-filtration from $M_1 =< W_1, \{-a \rightarrow \mid a \in L_1\}, V_1 >$ in $M_2 =< W_2, \{-a \rightarrow \mid a \in L_2\}, V_2 >$. For any formula $\varphi \in ?$, for any world $m_i \in W_1$ holds that $M_1 \models_{m_i} \varphi$ if and only if $M_2 \models_{f(m_i)} \varphi$ for any world m_i in W_1.*

proof:

The proof proceeds by induction on the complexity of the formula $\varphi \in L$.

$\varphi = \top$. Trivial, any model satisfies $\top$.

$\varphi = p_j$. Given that φ is in ?, condition (Var) assures that $M_1 \models_{m_i} p_j$ if and only if $M_2 \models_{f(m_i)} p_j$.

Inductive Hypothesis: For φ of complexity n, $M_1 \models_{m_i} \varphi$ if and only if $M_2 \models_{f(m_i)} \varphi$.

Let φ be a formula of complexity n+1, then:

$\varphi = \neg\psi$. $M_1 \models_{m_i} \neg\psi$ iff it is not the case that $M_1 \models_{m_i} \psi$ iff (by IH, that can be applied given that ? is closed by subformulas and so $\psi \in ?$) it is not the case that $M_2 \models_{f(m_i)} \psi$ iff $M_2 \models_{f(m_i)} \neg\psi$.

$\varphi = (\psi \vee \theta)$. $M_1 \models_{m_i} (\psi \vee \theta)$ iff $M_1 \models_{m_i} \psi$ or $M_1 \models_{m_i} \theta$ iff (by IH, and given that $\psi, \theta \in ?$) $M_2 \models_{f(m_i)} \psi$ or $M_2 \models_{f(m_i)} \theta$ iff $M_2 \models_{f(m_i)} (\psi \vee \theta)$.

$\varphi = [a]\psi$ for some label a. This is the interesting case. We will prove the double implication.

Let m_i be any world, suppose that $M_1 \models_{m_i} [a]\psi$. Let m' be anyone in W_2 such that $f(m_i) - a \rightarrow m'$. As f is suryective, we have that there exists $m_j \in W_1$ such that $f(m_j) = m'$. But then (Fil) assures that $M_1 \models_{m_j} \psi$, now by IH $M_2 \models_{f(m_j)} \psi$. That is $M_2 \models_{m'} \psi$ and as m' was any such that $f(m_i) - a \rightarrow m'$ we have that $M_2 \models_{f(m_i)} [a]\psi$.)

Suppose now that $M_2 \models_{f(m_i)} [a]\psi$ and let m' be any in W_1 such that $m_i - a \rightarrow m'$. As $\varphi \in ?$ we have that $a \in L(?)$. As f is an $L(?)$-morphism we have that $f(m_i) - a \rightarrow f(m')$ and then $M_2 \models_{f(m')} \psi$. Now, by IH, $M_1 \models_{m'} \psi$ and then $M_1 \models_{m_i} [a]\psi$. □

PROPOSITION CORRECTNESS OF TRANSLATION. *If G is a design graph, M obtained from G is a Kripke model.*

proof:

We have to show that $M =< W, \{-a \rightarrow \mid a \in L\}, V >$ obtained from G, follows the definition of Kripke models:

1. $W \neq \emptyset$ iff $N \neq \emptyset$, assured by definition of G.
2. L is a finite set iff LE is a finite set, assured by definition of G.
3. Let a be any in L, $-a \rightarrow \subseteq W \times W$ by construction of M.
4. V is a relation in $((I\!P \times W) \times 0, 1)$ by construction, we have to show that V is a total function, but this is assured because R_{LN} is a total function. □

PROPOSITION CORRECTNESS OF THE MODEL CHECKING ALGORITHM. *The Model Checking Algorithm stops for any formula φ of* **KPI**, *and upon termination for any subformula $\psi \in Sub(\varphi)$, $M \models_{m_j} \psi$ iff $\psi \in C_j$.*

proof:

Given that φ is a formula of **KPI** it has finite size, then the set $Sub(\varphi)$ of its subformulas is finite. Furthermore, the model M is finite. This two facts assure termination.

We can prove that the algorithm obtains a correct result by induction in the complexity of the formula ψ in $Sub(\varphi)$.

$\psi = \top$. $M \models_{m_j} \top$ (by definition of $\models$) and $\top \in C_j$ (by step 2).

$\psi = p_i$. $M \models_{m_j} p_i$ iff $V(p_i, m_j) = 1$ iff $p_i \in C_j$ (by step 2).

Inductive Hypothesis: For ψ of complexity n, holds $M \models_{m_j} \psi$ iff $\psi \in C_j$.

Let ψ be a formula of complexity n+1, then:

$\psi = \neg\theta$. $M \models_{m_j} \neg\theta$ iff it is not the case that $M \models_{m_j} \theta$ iff (by IH and given that $\theta \in Sub(\varphi)$) we have that $\theta \notin C_j$. And by step 3, $\neg\theta \in C_j$.

$\psi = (\theta \vee \xi)$. $M \models_{m_j} (\theta \vee \xi)$ iff ($M \models_{m_j} \theta$ or $M \models_{m_j} \xi$) iff (by IH and $\theta, \xi \in Sub(\varphi)$) we have that ($\theta \in C_j$ or $\xi \in C_j$). And by step 3, $(\theta \vee \xi) \in C_j$.

$\psi = [a]\theta$. $M \models_{m_j} [a]\theta$ iff for any m_k such that $m_j - a \rightarrow m_k$ $M \models_{mk} \theta$ iff (by IH and $\theta \in Sub(\varphi)$) we have that for any m_k such that $m_j - a \rightarrow m_k$, $\theta \in C_k$. And by step 3, $[a]\theta \in C_j$.

$\psi = [a]_i\theta$. $M \models_{m_j} [a]_i\theta$ iff for any m_k such that $m_k - a \rightarrow m_j$ $M \models_{mk} \theta$ iff (by IH and $\theta \in Sub(\varphi)$) we have that for any m_k such that $m_k - a \rightarrow m_j$, $\theta \in C_k$. And by step 3, $[a]_i\theta \in C_j$. □

LEMMA MODAL EXPANSION LEMMA. *Let $M =< W, \{-a \rightarrow \mid a \in L\}, V >$ and $M^+ =< W^+, \{-a \rightarrow \mid a \in L^+\}, V^+ >$ be modal models, such that M^+ is an L^+-expansion of M, then for any formula $\varphi \in \mathcal{L}_L$ holds that for any $m \in W$, $M \models_m \varphi$ iff $M^+ \models_m \varphi$.*

proof:

We begin pointing that by 1 in the definition of L^+-expansion $W = W^+$ and so it has sense to claim the double implication of the lemma.

The proof is simple and proceeds by induction in the complexity of φ. The propositional cases are resolved using that $V = V^+$ by 3 in the definition. By induction we go over the Boolean operators. For the modal case, given that $\varphi \in \mathcal{L}_L$ its modalities must use labels in L and for such labels, relations $-a \rightarrow_L$ and $-a \rightarrow_{L^+}$ are equal by 2 in the definition. □

PROPOSITION COMPOSITION, $-c \rightarrow = -a \rightarrow \circ -b \rightarrow$. *Let $M =< W, \{-a \rightarrow \mid a \in L\}, V >$ be any modal model such that $a, b \in L$, then $M^+ =< W, \{-a \rightarrow \mid a \in L\} \cup \{-c \rightarrow \mid c \notin L$ and $m - c \rightarrow m'$ iff there exists m'' such that $m - a \rightarrow m''$ and $m'' - b \rightarrow m'\}, V >$ is an $L \cup \{c\}$-expansion of M which satisfies that for any $\varphi \in \mathcal{L}_{L\cup\{c\}}$, for any $m \in W$, $M^+ \models_m <c> \varphi \leftrightarrow <a><b> \varphi$.*

proof:

By construction M^+ is an $L \cup \{c\}$-expansion of M. We will show that for any $\varphi \in \mathcal{L}_{L\cup\{c\}}$, for any $m \in W$, $M \models_m <c> \varphi \leftrightarrow <a><b> \varphi$. Let φ be any in $\mathcal{L}_{L\cup\{c\}}$, m any in W, suppose that $M^+ \models_m <a><b> \varphi$ iff there exists m'' such that $m - a \rightarrow m''$ and $M^+ \models_{m''} <b> \varphi$ iff there exists m'' such that $m - a \rightarrow m''$ and there exists m' such that $m'' - a \rightarrow m'$ and $M^+ \models_{m'} \varphi$ iff there exists m' such that (there exists m'' such that $m - a \rightarrow m'' - a \rightarrow m'$ and $M^+ \models_{m'} \varphi$) iff there exists m' such that $m - c \rightarrow m'$ and $M^+ \models_{m'} \varphi$ iff $M^+ \models_m <c> \varphi$. □

PROPOSITION UNION, $-c \rightarrow = -a \rightarrow \cup -b \rightarrow$. *Let $M =< W, \{-a \rightarrow \mid a \in L\}, V >$ be any modal model such that $a, b \in L$ then $M^+ =< W, \{-a \rightarrow \mid a \in L\} \cup \{-c \rightarrow \mid c \notin L$ and $m - c \rightarrow m'$ iff $m - a \rightarrow m'$ or $m - b \rightarrow m'\} >$ is an $L \cup \{c\}$-expansion of M which satisfies that for any $\varphi \in \mathcal{L}_{L\cup\{c\}}$, for any $m \in W$, $M^+ \models_m <c> \varphi \leftrightarrow (<a> \varphi \vee <b> \varphi)$.*

proof:

By construction M^+ is an $L \cup \{c\}$-expansion of M. We will show that for any $\varphi \in \mathcal{L}_{L\cup\{c\}}$, for any $m \in W$, $M \models_m <c> \varphi \leftrightarrow (<a> \varphi \vee <b> \varphi)$. Let φ be any in $\mathcal{L}_{L\cup\{c\}}$, m any in W, suppose that $M^+ \models_m (<a> \varphi \vee <b> \varphi)$ iff $M^+ \models_m <a> \varphi$ or $M^+ \models_m <b> \varphi$ iff there exists m' such that $m - a \rightarrow m'$ and $M^+ \models_{m'} \varphi$ or there exists m'' such that $m - b \rightarrow m''$ and $M^+ \models_{m''} \varphi$ iff there exists m' such that $m - a \rightarrow m'$ or $m - b \rightarrow m'$ and $M^+ \models_{m'} \varphi$ iff there exists m' such that $m - c \rightarrow m'$ and $M^+ \models_{m'} \varphi$ iff $M^+ \models_m <c> \varphi$. □

PROPOSITION INVERSE, $-b \rightarrow = (-a \rightarrow)^{-1}$. *Let* $M =< W, \{-a \rightarrow \mid a \in L\}, V >$ *be any modal model such that* $a \in L$, *then* $M^+ =< W, \{-a \rightarrow \mid a \in L\} \cup \{-b \rightarrow \mid b \notin L$ *and* $m - b \rightarrow m'$ *iff* $m' - a \rightarrow m\}, V >$ *is an* $L \cup \{b\}$*-expansion of* M *which satisfies that for any* $\varphi \in \mathcal{L}_{L\cup\{b\}}$, *for any* $m \in W$, $M^+ \models_m <b> \varphi \leftrightarrow <a>_i \varphi$.

proof:
By construction M^+ is an $L \cup \{b\}$-expansion of M. We will show that for any $\varphi \in \mathcal{L}_{L\cup\{b\}}$, for any $m \in W$, $M^+ \models_m <b> \varphi \leftrightarrow <a>_i \varphi$. Let φ be any in $\mathcal{L}_{L\cup\{b\}}$, m any in W, suppose that $M^+ \models_m <a>_i \varphi$ iff there exists m' such that $m' - a \rightarrow m$ and $M^+ \models_{m'} \varphi$ iff there exists m' such that $m - b \rightarrow m'$ and $M^+ \models_{m'} \varphi$ iff $M^+ \models_m <b> \varphi$. □

PROPOSITION INTERSECTION, $-c \rightarrow = -a \rightarrow \cap -b \rightarrow$. *Let* $M =< W, \{-a \rightarrow \mid a \in L\} >$ *be a model that corresponds to a design graph such that* $a, b \in L$ *then* $M^+ =< W, \{-a \rightarrow \mid a \in L\} \cup \{-c \rightarrow \mid c \notin L$ *and* $m - c \rightarrow m'$ *iff* $m - a \rightarrow m'$ *and* $m - b \rightarrow m'\} >$ *is an* $L \cup \{c\}$*-expansion of* M *which satisfies that for any* $\varphi \in \mathcal{L}_{L\cup\{c\}}$, $p \in I\!P$, $M^+ \models_m <c> (\varphi \wedge p) \leftrightarrow (<a> (\varphi \wedge p) \wedge <b> (\varphi \wedge p))$.

proof:
By construction M^+ is an $L \cup \{c\}$-expansion of M. We will show that for any $\varphi \in \mathcal{L}_{L\cup\{c\}}$, for any $p \in I\!P$, for any $m \in W$, $M^+ \models_m <c> (\varphi \wedge p) \leftrightarrow (<a> (\varphi \wedge p) \wedge <b> (\varphi \wedge p))$. Let φ be any in $\mathcal{L}_{L\cup\{c\}}$, m any in W, p any in $I\!P$, suppose that $M^+ \models_m (<a> (\varphi \wedge p) \wedge <b> (\varphi \wedge p))$ iff $M^+ \models_m <a> (\varphi \wedge p)$ and $M^+ \models_m <b> (\varphi \wedge p)$ iff there exists m' such that $m - a \rightarrow m'$ and $M^+ \models_{m'} (\varphi \wedge p)$ and there exists m'' such that $m - b \rightarrow m''$ and $M^+ \models_{m''} (\varphi \wedge p)$ (From the existence of worlds m' and m'' that satisfy the corresponding conditions it does not follow that there exists a single world that satisfies them. But we can claim this in our case, because M is a model corresponding to a design graph and then if in both m' and m'' holds p, we can assert $m' = m''$.) iff there exists m' such that $m - a \rightarrow m'$ and $m - b \rightarrow m'$ and $M^+ \models_{m'} (\varphi \wedge p)$ iff there exists m' such that $m - c \rightarrow m'$ and $M^+ \models_{m'} (\varphi \wedge p)$ iff $M^+ \models_m <c> (\varphi \wedge p)$. □

PROPOSITION SUBTRACTION, $-c \rightarrow = -a \rightarrow - - b \rightarrow$. *Let* $M =< W, \{-a \rightarrow \mid a \in L\} >$ *be a model that corresponds to a design graph such that* $a, b \in L$ *then* $M^+ =< W, \{-a \rightarrow \mid a \in L\} \cup \{-c \rightarrow \mid c \notin L$ *and* $m - c \rightarrow m'$ *iff* $m - a \rightarrow m'$ *and no* $m - b \rightarrow m'\} >$ *is an* $L \cup \{c\}$*-expansion of* M *which satisfies that for any* $\varphi \in \mathcal{L}_{L\cup\{c\}}$, $p \in I\!P$, $M^+ \models_m <c> (\varphi \wedge p) \leftrightarrow (<a> (\varphi \wedge p) \wedge \neg <b> (\varphi \vee p))$.

proof:
By construction M^+ is an $L \cup \{c\}$-expansion of M. We will show that for any $\varphi \in \mathcal{L}_{L\cup\{c\}}$, for any $p \in I\!P$, for any $m \in W$, $M^+ \models_m <c> (\varphi \wedge p) \leftrightarrow (<a> (\varphi \wedge p) \wedge \neg <b> (\varphi \wedge p))$. Let φ be any in $\mathcal{L}_{L\cup\{c\}}$, m any in W, p any in $I\!P$, suppose that $M^+ \models_m (<a> (\varphi \wedge p) \wedge \neg <b> (\varphi \wedge p))$ iff $M^+ \models_m <a> (\varphi \wedge p)$ and $M^+ \models_m \neg <b> (\varphi \wedge p)$ iff there exists m' such that $m - a \rightarrow m'$ and $M^+ \models_{m'} (\varphi \wedge p)$ and there does not exist m'' such that $m - b \rightarrow m''$ and $M^+ \models_{m''} (\varphi \wedge p)$ (This case is similar to the proof of the last proposition, the implication $\Rightarrow$ is valid for any model but not so $\Leftarrow$. Only the fact that there exists a single world satisfying proposition p, let us go from $m - a \rightarrow m'$ and not $m - b \rightarrow m'$, to there is no m'' such that $m - b \rightarrow m''$) iff there exists m' such that $m - a \rightarrow m'$ and no $m - b \rightarrow m'$ and $M^+ \models_{m'} (\varphi \wedge p)$ iff there exists m' such that $m - c \rightarrow m'$ and $M^+ \models_{m'} (\varphi \wedge p)$ iff $M^+ \models_m <c> (\varphi \wedge p)$. □

PROPOSITION CORRECTNESS OF THE FILTRATION ALGORITHM. *The filtration algorithm stops, and upon termination the result* $M^* =< W^*, \{-a \rightarrow^* \mid a \in L^*\}, V^* >$ *is a modal model.*

proof:
As ? is finite, $\bigwedge ?$ is a formula of **KPI** and in this case we already showed that the model checking algorithm stops, so step 1 stops. Step 2 is a cycle over the number of elements in W', a finite number of worlds. Step 4 is a cycle over the number of edges in M, also finite. Step 5 is a double cycle over the number of propositions in ? and the number of worlds in W, both finites. With this we proved that the algorithm stops.

If M is a model then W is non empty. Let then $m_i \in W$, we will have by step 2 that $[m_i] \in W^*$. Step 3 defines L^* as $L(?)$ the set of labels of ?. Step 4 defines the elements of $-a \rightarrow^*$ for a in L^* as pairs of $W^* \times W^*$, then $-a \rightarrow^*$ are binary relations over the set of worlds. Step 5 defines V^* as

$V^*(p_i,[m_j]) = V(p_i,m_j)$ if $p_i \in ?$ and $V^*(p_i,[m_j]) = 0$ otherwise. We must now prove that it cannot be $V^*(p_i,[m_j]) \neq V^*(p_i,[m_k])$ if $[m_j] = [m_k]$. If p_i does not belong to $?$, then $V^*(p_i,[m_l]) = 0$ for any m_l. If p_i belongs to $?$, and $[m_j] = [m_k]$ then step 2 assures that $C_j = C_k$. Then m_j and m_k assign the same value to p_i, because p_i belongs to both sets or neither. Then $V(p_i,m_j) = V(p_i,m_k)$ and so $V^*(p_i,[m_j]) = V^*(p_i,[m_k])$. □

THEOREM FILTRATION EXISTENCE. *Let* $M =< W, \{-a \rightarrow \mid a \in L\}, V >$ *be a modal model and* $M^* =< W^*, \{-a \rightarrow^* \mid a \in L^*\}, V^* >$ *the model obtained from* M *for a set of formulas* $?$ *by the last algorithm. Then* $f : W \rightarrow W^*$ *such that* $f(m_i) = [m_i]$ *is a* $?$*-filtration from* W *in* W^*.

proof:

For f to be a $?$-filtration, it must be a suryective morphism that satisfies conditions (Var) and (Fil).

Let's first analyze some properties of the objects built by the algorithm:

Step 2 builds the set $W^* = \{[m_i] \in \mathcal{P}(W) \mid m_i \in W\}$, with the property that $m_i \in [m_i]$ and m_j, m_k belongs to $[m_i]$ iff $C_j = C_k$ that is iff for any $\varphi \in ?$, $M \models_{m_j} \varphi$ iff $M \models_{mk} \varphi$. We can define $m_j \sim m_k$ iff $m_j, m_k \in [m_i]$ for some i. This is an equivalency relation over W and $W^* = W/\sim$.

But then f is the canonical function that assigns to an element of W its class in W^*, so f is suryective.

To prove that f is an $L(?)$-morphism, take m_i, m_j in W such that $m_i - a \rightarrow m_j$ for some label $a \in L(?)$. We have to show that $f(m_i) - a \rightarrow^* f(m_j)$. But $f(m_i) = [m_i]$ and $f(m_j) = [m_j]$, and step 4 assures that if $m_i - a \rightarrow m_j$ and $a \in L(?)$ we have that $[m_i] - a \rightarrow^* [m_j]$.

Step 5 in the algorithm assures that for any propositional symbol p_i in $?$, $V^*(p_i,[m_j]) = 1$ iff $V(p_i,m_j) = 1$. Or in other words, for any $p_i \in ?$, for any $m_j \in W$, $V(p_i,m_j) = V^*(p_i,f(m_j))$, satisfying condition (Var).

To verify condition (Fil) take two elements $m_i, m_j \in W$ such that $f(m_i) - a \rightarrow^* f(m_j)$ for some a in $L(?)$. By step 4, $[m_i] - a \rightarrow^* [m_j]$ iff there exists $m_k \in [m_i]$ and $m_l \in [m_j]$ such that $m_k - a \rightarrow m_l$. Then, if we have that $[a]\varphi \in ?$, $M \models_{m_i} [a]\varphi$ implies $M \models_{mk} [a]\varphi$ because they belong to the same equivalence class. But then $M \models_{ml} \varphi$ because m_l is accessible from m_k. And then, again because they belong to the same class, $M \models_{m_j} \varphi$. □

THEOREM CORRECTNESS AND COMPLETENESS OF **KPIVn**. *The logic* **KPIVn** *is correct and complete relative to the class of models* $CVn = \{< W, \{-a \rightarrow \mid a \in L\}, V > \mid W \neq \emptyset$ *and for any* $m \in W$, *there exists* p_i *such that* $V(p_i,m) = 1$ *and for any* p_j, *with* $i \neq j$, $V(p_j,m) = 0\}$.

proof:

Correctness:

To complete the correctness proof, and given that we have correctness for **KPI** in the class of all the models (and then in the lesser class of all the models in CVn) it is enough to show that Vn is valid in any model of CVn.

Let M be any model of CVn, let m be any world in W, we want to prove that

$M \models_m \bigvee_{(i \leq n)}(p_i \bigwedge_{(j \leq n, j \neq i)} \neg p_j)$.

Suppose not, then (by definition of $\models$) $M \not\models_m \bigvee_{(i \leq n)}(p_i \bigwedge_{(j \leq n, j \neq i)} \neg p_j)$ iff

$M \models_m \neg \bigvee_{(i \leq n)}(p_i \bigwedge_{(j \leq n, j \neq i)} \neg p_j)$ iff

$M \models_m \bigvee_{(i \leq n, j \leq n, i \neq j)}(p_i \wedge p_j) \vee \bigwedge_{(i \leq n)} \neg p_i$.

That is, there exist different i, j such that $M \models_m (p_i \wedge p_j)$ or for any i $M \models_m \neg p_i$ iff

$(V(p_i,m) = 1$ and $V(p_j,m) = 1)$ or $V(p_i,m) = 0$ for any p_i, but then M is not a model in CVn. *Contradiction.*

Completeness:

Using the canonical model method, if Σ is the set of all the axioms of the logic, it is enough to show that the canonical model M_Σ built from Σ belong to CVn. To prove that M_Σ is in CVn, given that by construction $W_\Sigma \neq \emptyset$, it is enough to show that for any $m \in W_\Sigma$, there exists p_i, such that $V_\Sigma(p_i,m) = 1$ and for any p_j with $i \neq j$ we have that $V_\Sigma(p_j,m) = 0$.

Suppose not, then there exists m such that either

1. we can found $i \neq j$ such that $V_\Sigma(p_i,m) = 1$ and $V_\Sigma(p_j,m) = 1$ or either

2. $V_\Sigma(p_i,m) = 0$ for any i.

Case 1.

Then by definition of V_Σ, p_i is in m and p_j is in m.

But m is a maximal consistent set relative to Σ, and so $p_i \wedge p_j$ is in m.

But Vn is in m because it is an axiom, and $\{Vn\} \vdash (p_i \rightarrow \neg p_j)$ and $\{Vn\} \vdash (p_j \rightarrow \neg p_i)$.

Then in m we have that $(p_i \wedge p_j) \wedge (\neg p_i \wedge \neg p_j)$ an instance of contradiction, and m would not be a maximal consistent set. *Contradiction.*

Case 2.

By definition of V_Σ, $V_\Sigma(p_i, m) = 0$ iff $\neg p_i \in m$, then $\bigwedge_{(i \leq n)} \neg p_i \in m$ (because m is a maximal consistent set). But $\{\bigwedge_{(i \leq n)} \neg p_i\} \vdash \neg Vn$, and so, $\neg Vn \in m$ and at the same time $Vn \in m$. *Contradiction.* □

ACKNOWLEDGMENTS

The authors wish to thank their directors Dr. Daniel Yankelevich and Dr. Miguel Felder for their guidance, the referees Verónica Becher and Gabriel Baum for their comments and Ricardo Rodriguez and Eduardo Fermé for their help.

REFERENCES

Agustí, J., Robertson, D., and Puigsegur, J. 1995. GraSp: A GRAphical SPecification language for the preliminary specification of logic programs.

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

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

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

Blackburn, P. 1993. Nominal tense logic. *Notre Dame Journal of Formal Logic* 34.

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

Burgess, J. P. 1984. Basic tense logic. In *Handbook of Philosophical Logic*, Volume II. D. Reidel Publishing Company.

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

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

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

Fiadeiro, J., Sernadas, C., Maibaum, T., and Saake, G. 1991. Proof-theoretic semantics of object-oriented specification constructs. In K. Kent and Meersman (Eds.), *Object-Oriented Databases: Analysis, Design and Construction.* North-Holland.

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

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

Henzinger, T. A., Nicollin, X., Sifakis, J., and Yovine, S. 1994. Symbolic model checking for real-time systems. *Information and Computation 111,* 2 (June), 193–244.

Hintikka, J. 1962. *Knowledge and Belief.*

Hirsch, D. and Areces, C. 1995. From boxes to worlds. In *Proceedings of the First CACIC.*

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

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

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

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

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

Popkorn, S. 1994. *First Steps in Modal Logic.* Cambridge University Press.

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