From Boxes to Worlds

Areces, Carlos Eduardo
email:postmast@logia.uba.ar

Hirsch, Dan Francisco
email:dh7h@zorzal.uba.ar

Universidad de Buenos Aires, FCEyN
Departamento de Computación
Pabellón I, Ciudad Universitaria
(1428) Buenos Aires, Argentina

July, 1995

Abstract

In this paper we show an innovative way to represent graphic designs of systems and to verify design properties. The graphic designs are thought as models of an extended modal logic; and the methods used to verify properties are developed from techniques typical of classic modal logic, extended to cover the differences in the underlying formalism. We present the procedures or logical tools to derive the modal model associated to a given design, the filtration of models and the construction of modal descriptions.

A higher level of abstraction is obtained in this way, and it lets us reason over designs in a strictly formal manner. The power of formal provability is also achieved.

We use a working example to show how our tools help to verify properties of design like the detection of cycles (self loops).

Keywords

Software Design, Graphic Languages, Modal Logics, Formal Verification

Acknowledgments

We specially thank Professors Miguel Felder and Daniel Yankelevich for their help and advice in the development of this work. We also thank Verónica Becher and Eduardo Fermé for helping us in the discussion of the first ideas.

From Boxes to Worlds

1. Introduction

One of the steps in the software development cycle is the design stage. After obtaining a specification of the problem to solve (in a more or less strict formal language), we must give the general lines that define the components of the system, the different inter-relations among them, the properties or restrictions the components must satisfy, etc.

Different approaches have been used to do this. The simplest among them use sketches, graphics, annotations and even different colors to suggest an idea. But the designs so obtained are ambiguous and proving its properties is almost impossible. The great variety and the free style involved make them impractical.

An important improvement over this situation was the definition and, principally, the general acceptance of a set of tools specifically tuned to deal with this problem, like E-R diagrams, Data Flow Diagrams, Structured Charts, etc.

Even though these tools eliminated the variety, they didn't exclude ambiguity. Not to mention the demonstration of properties once a design was complete.

In short, a new *global language* for design description was established and a graphical one was selected as the best option. But a formal *semantic* for this language was missing.

Currently, many researchers have been working in this problem using new techniques. The work of [Consens & Mendelzon, 1992], for example, is interested in providing tools to handle designs and obtain simplified visions of the systems that help the designers to understand them. They use GraphLog (a graphical language) as a visualization tool over a Prolog database to represent a design specification.

In our opinion, a jump to a higher level of abstraction is needed. We are searching for a tool that lets us reason over these graphs in a more *abstract* way, in addition to obtain new simplified visions. We think that the use of a modal language to describe both designs and its properties, will give us the exact expressiveness we were looking for. We get also the power of formal provability that comes with all logics.

This higher level of abstraction is obtained when we consider designs as modal structures. In this way components, relations and properties of the design are expressed in the *logic* language. And *logic* operations will let us work with designs, obtaining from them the desired visions, proving their properties and giving them meaning.

In Section 2 we introduce the basic notions of modal logics needed in this article. Section 3 presents the new procedures that let us work with the design in the new environment just proposed. Section 4 shows examples of application like the detection of cycles in the design hierarchy and the testing of consistency between the different relations. In Section 5 we present the conclusions and finally in Section 6 we comment different points not yet analyzed or under development.

2. Modal Logics

The classic modal calculus is obtained as an extension of the classic propositional calculus, introducing the possibility operator <> and obtaining in this way an alphabet A = {(,), ¬, ∨, <>} [Hughes & Cresswell, 1968]. As usual, the dual operator of necessity is defined as $[] \equiv_{def} \neg <> \neg$.

In this paper, we extend the notion of *normal modal logics*, that is, we assume that the following axioms and rules of inference are valid in our systems:

K. $[](p \rightarrow q) \rightarrow ([]p \rightarrow []q)$.
RN. From $\vdash p$, infer $\vdash []p$.
MP. From $\vdash p$ y $\vdash p \rightarrow q$, infer $\vdash q$.

Normal modal logics are minimal logics (see [Chellas, 1980]). Every model in the class of all the models satisfies these axioms and rules. $\vdash$ will be use to express provability in these minimal systems.

The semantic of a modal logic is defined using *Kripke Models.*
A Kripke Model is a tuple $M = < W, ..., R, P >$, where:

W is a set, the elements of W will be called *worlds* or *possible worlds*;
R is a relation over W (i.e. $R \subseteq W \times W$), usually named *accessibility relation*;
P is a function on a set I of indexes (numerable) such that for each index , p_i is a subset of W (i.e. $P: I \rightarrow P(W)$), this function is a *valuation* that assigns propositional variables to worlds. P(i) = X means that the proposition $[p_i]$ = X.

The ellipsis in the tuple indicates the possibility of additional elements among the components of a model *M*.
A model is said to be finite if its set of worlds has only a finite number of elements; otherwise the model is infinite.

We write $\models^M_\alpha A$ to mean that A is valid at the possible world α in the model *M*.
The notion of validity in possible worlds is defined recursively as follows:

1. $\models^M_\alpha p_n$ iff $\alpha \in P(n)$.
2. $\models^M_\alpha \neg A$ iff not $\models^M_\alpha A$.
3. $\models^M_\alpha A \vee B$ iff $\models^M_\alpha A$ or $\models^M_\alpha B$.
4. $\models^M_\alpha <>A$ iff there exist a world $\beta \in W$ such that $(\alpha, \beta) \in R$ and $\models^M_\beta A$.

As we can see, A is possible in the world α if there is in the model a world β where A is valid, and β is reachable from α via the relation R. Based in this semantics, the inverse modality $<>_I$ is defined as follows:

5. $\models^M_\alpha <>_I A$ iff there exist a world $\beta \in W$ such that $(\beta, \alpha) \in R$ and $\models^M_\beta A$.

This modality is equivalent to the Past operator (P) of Temporal Logic and has the same properties [Burguess, 1984]. Its dual operator is equivalent to H (always in the past), $[]_i \equiv_{def} \neg <>_i \neg$. That is, our minimal logic is really an extension of the minimal temporal logic and so verifies the following axioms and rules:

K. $[](p \rightarrow q) \rightarrow ([]p \rightarrow []q)$.
KI. $[]_i(p \rightarrow q) \rightarrow ([]_i p \rightarrow []_i q)$.
It. $p \rightarrow []<>_i p$
ItI. $p \rightarrow []_i <> p$

RN. From $\vdash p$, infer $\vdash \Box p$.
RNI. From $\vdash p$, infer $\vdash \Box_i p$.
MP. From $\vdash p$ y $\vdash p \rightarrow q$, infer $\vdash q$.

In our paper, we will use models that have more than one accessibility relation, and so we have one modality accompanying each of them. That is, our models will be $M = \langle W, R_1, ..., R_n, P \rangle$ and we will have, for each R_j in the model:

4.j $\models^M_\alpha \langle j \rangle A$ iff there exist a world $\beta \in W$ such that $(\alpha, \beta) \in R_j$ and $\models^M_\beta A$.

5.j $\models^M_\alpha \langle j \rangle_i A$ iff there exist a world $\beta \in W$ such that $(\beta, \alpha) \in R_j$ and $\models^M_\beta A$.

The group of axioms that characterize our logic must be extended accordingly.

We write $\models^M A$ to mean that A is valid in the model (i.e., A is valid in all the worlds of the model *M*) and if C is a class of models (for example, the models where the relations R_i are transitive), we write $\models_C A$ to mean that A is valid at every model in the class.

3. New Working Tools

We already said that the more natural way to describe a design is to sketch a graph in which we use a symbol (usually a box or a rectangle) to represent the components, and lines connecting them as the relations.

For example, take part of the design of a system that checks the CRC value of a file.

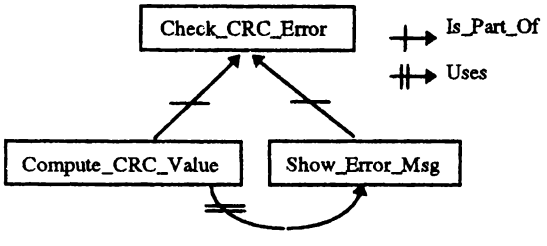

Figure N°1

The main component is organized in two modules: one computes the CRC value of the file and the other deals with errors that may appear. Every time the module that computes the CRC value detects an error, the Error module shows a message

In this design there are arrows that depict different interactions between components. They can be represented as labeled edges.

The components are also labeled with theirs names.

More formally, a general scheme of design is defined as a direct labeled graph $\langle N, E, LN, LE, R_{LN}, R_{LE} \rangle$, where:

N is the set of Nodes (finite).

E is a set of Edges ($E \subseteq N \times N$).

LN and LE are sets of labels (disjoint and finites).

R_{LN} is a total function in LN that assigns a label to each node (R_{LN}: N □ LN).

R_{LE} is a 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*, names for boxes are *node labels*, names for arrows are *edge labels*, to give a name to a box is *to include a pair in* R_{LN} and to give a name to an arrow is *to Include a pair in* R_{LE}.

We will begin now to develop the different tools that will be used in the verification of design properties. During all this section we will work with the following example, to show how the different procedures are used.

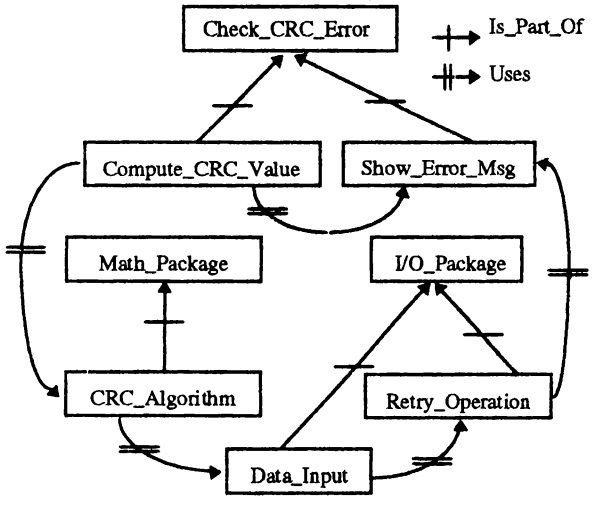

Figure N°2

This graph corresponds to a short design of a system that verify the CRC code of a file.

The main component contains the routines to compute the CRC value and then notify if errors are found. To compute the value, it calls functions from a Mathematics Library. This functions access the file using I/O routines. If there is a problem during the I/O operations a message is sent to the error routine.

There are two relations:

Is_Part_Of: is used to show the relation between components and principal components.

Uses: shows the use of functions between components.

The graph associated to this design would be G = < N, E, LN, LE, R_{LN}, R_{LE} > with:

N = {1, 2, 3, 4, 5, 6, 7, 8}
LN = {Check_CRC_Error, Compute_CRC_Value, Show_Error_Msg, Math_Package, I/O_Package, CRC_Algorithm, Retry_Operation, Data_Input}
R_{LN} = {(1,Check_CRC_Error), (2,Compute_CRC_Value), (3,Show_Error_Msg), (4,Math_Package), (5,I/O_Package), (6,CRC_Algorithm), (7,Retry_Operation), (8,Data_Input)}
E = {(2,1), (3,1), (2,6), (7,3), (6,4), (6,8), (8,5), (7,5)}
LE = {/, //}
R_{LE} = {((2,1),/), ((3,1),/), ((2,6),//), ((2,3),//), ((7,3),//), ((6,4),/), ((6,8),//), ((8,5),/), ((7,5),/), ((8,7),//)}

3.1 From a Graph of Design to a Modal Model

As we will work with modal logics, we must first transform the graph that represents a given design to the new language.

Given a graph < N, E, LN, LE, R_{LN}, R_{LE} > corresponding to a design, we can construct the associated Kripke model, as M = < W, R_1, ..., R_n, P > where:

W is a set of worlds w_i with i ∈ #N.

R_i = {(w_k, w_l) ∈ W × W | ((k, l), i) ∈ R_{LE}}.

P = {(ln, {w_n}) ∈ LN × P(W) | (n, ln) ∈ R_{LN}}.

From the definition we can infer that the number of propositional variables of the model is finite and coincides with the number of labels the original graph has. We interpret the proposition variable p_{ln} as "The name of the component is ln ".

Example: To give an example of the method to define the model we will work over the graph in Figure N°2. From this graph, we would get the model M = <W, $R_/$, $R_{//}$, P> with:

$W = \{w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8\}$
$R_/ = \{(w_2, w_1), (w_3, w_1), (w_6, w_4), (w_8, w_5), (w_7, w_5)\}$
$R_{//} = \{(w_2, w_6), (w_6, w_8), (w_7, w_3)\}, (w_8, w_7), (w_2, w_3)\}$
P = {(Check_CRC_Error, w_1), (Compute_CRC_Value, w_2), (Show_Error_Msg, w_3), (Math_Package, w_4), (I/O_Package, w_5), (CRC_Algorithm, w_6), (Retry_Operation, w_7), (Data_Input, w_8)}

3.2 Filtration of Models

The verification of the properties of a design is typically associated to the demonstration of properties about the relations involved. As we said above, given the usual complexity of a design, the usual method is to try to obtain a more global vision defining new relations over the original ones or giving attention to a subset of them.

In a modal language, the relations are linked to the modal operators of the logic. It is for this reason that the equivalent to compose relations is the definition of new "modalities". Given the relations R_i y R_j we can represent $R_i \circ R_j$ by the definition (in the metalanguage) of $\langle\rangle p \equiv_{def} \langle i\rangle\langle j\rangle p$.

The procedure of filtration lets us arrive at a new model where this definition and its associated relation are introduced in the language.

If the filtration is used without introducing new modalities, what we obtain is a cut of the original model. Only the selected relations are present

Let Γ be a set of sentences closed under subsentences, we define for a given model *M*, the equivalence relation $\equiv$ over the worlds of *M* as $\alpha \equiv \beta$ iff for every $A \in \Gamma$, $\models^M_\alpha A$ if and only if $\models^M_\beta A$. Then we define $[\alpha] = \{\beta \in W \mid \alpha \equiv \beta\}$, and for a set of worlds X, $[X] = \{[\alpha] \in P(W) \mid \alpha \in X\}$. $[\alpha]$ is the class of equivalence for a given world α and [X] is the set of classes of equivalence corresponding to the worlds in X.

Let $M = \langle W, R_1, ..., R_n, P\rangle$ be a model, and let Γ be a set of sentences closed under subsentences. We define a *filtration of M under* Γ as a model $M^* = \langle W^*, R^*_1, ..., R^*_m, P^*\rangle$ where

1. $W^* = [W]$
2. for every $\alpha, \beta \in W$
   a. if $[\alpha]\, R^*_i\, [\beta]$ then for every sentence $[i]A \in \Gamma$, if $\models^M_\alpha [i]A$ then $\models^M_\beta A$.
   b. if $[\alpha]\, R^*_i\, [\beta]$ then for every sentence $\langle i\rangle A \in \Gamma$, if $\models^M_\beta A$ then $\models^M_\alpha \langle i\rangle A$.
   c. if $[i]A$ or $\langle i\rangle A \in \Gamma$ then ($\alpha R_i \beta$ imply $[\alpha]\, R^*_i\, [\beta]$).
3. $P_n^* = [P_n]$ for every n such that $p_n \in \Gamma$.

The filtration of a model under a set is not uniquely defined. While W^* and P_n^* are uniquely defined given Γ and *M*, there exist different relations R^*_i that verify the conditions.

As we want to obtain the minimal model, we must use the filtration that collapses as much as possible. The coarser filtration is defined for a model *M*, as:

$[\alpha]\, R^*_j\, [\beta]$ iff for every $[j]A \in \Gamma$, if $\models^M_\alpha [j]A$ then $\models^M_\beta A$,
for every $[j]_i A \in \Gamma$, if $\models^M_\beta [j]_i A$ then $\models^M_\alpha A$,
for every $\langle j\rangle A \in \Gamma$, if $\models^M_\beta A$ then $\models^M_\alpha \langle j\rangle A$ and
for every $\langle j\rangle_i A \in \Gamma$, if $\models^M_\alpha A$ then $\models^M_\beta \langle j\rangle_i A$.

Example: To give an example of the composition of relations using filtrations, we define <>p $\equiv_{def}$ <//></>p. If we filter the original model obtain from Figure N°2 under the set $\Gamma = L_{<>}$ (the logic over the alphabet A = {(,), ¬, ∨, <>}), we obtain the model M^* = <W*, R*, P*>, where:

W* = {{w_1}, {w_2}, {w_3}, {w_4}, {w_5}, {w_6}, {w_7}, {w_8}}
R* = {({w_2}, {w_4}), ({w_6}, {w_5}), ({w_7}, {w_1}), ({w_2}, {w_1}), ({w_8}, {w_5})}}
P* = {(Check_CRC_Error, {w_1}), (Compute_CRC_Value, {w_2}), (Show_Error_Msg, {w_3}), (Math_Package, {w_4}), (I/O_Package, {w_5}), (CRC_Algorithm, {w_6}), (Retry_Operation, {w_7}), (Data_Input, {w_8})}

In the design associated to the new model, the relation $R_{//}{}^{\circ}R_{/}$ is made explicit.

Example: To give an example of a cut of relations using filtrations, we do not need to define a new modality. Simply, we filter the original model under the set $\Gamma = L_{</>}$ (the logic over the alphabet A = {(,), ¬, ∨, </>}), and we obtain the model M^* = <W*, R*, P*>, where:

W* = {{w_1}, {w_2}, {w_3}, {w_4}, {w_5}, {w_6}, {w_7}, {w_8}}
R* = {({w_2}, {w_1}), ({w_3}, {w_1}), ({w_6}, {w_4}), ({w_8}, {w_5}), ({w_7}, {w_5})}
P* = {(Check_CRC_Error, {w_1}), (Compute_CRC_Value, {w_2}), (Show_Error_Msg, {w_3}), (Math_Package, {w_4}), (I/O_Package, {w_5}), (CRC_Algorithm, {w_6}), (Retry_Operation, {w_7}), (Data_Input, {w_8})}

In the associated design the unique relation is $R_{/}$.

3.3 From Modal Models to Modal Descriptions

After doing a filtration, we arrive at a new modal model. We could obtain semantic proofs of properties over this model (for example, we could prove $\models^M{}_{\alpha}\varphi$, that is, that the formula φ is valid in the world α). But if we want to make this proof automatically, it is better to use a syntactic prover. We will give now a procedure that let us get, given a model and a world, an axiomatic base (modal description) from which all the valid formulas in that world can be proved.

Some preliminary definitions:
We say that the model M is minimal with respect to a set of formulas Δ, if there are a world α in the model such that, for every $\varphi \in \Delta$, $\models^M{}_{\alpha}\varphi$ and it is the simplest one with this characteristic. That is, it has the least number of worlds, relations, edges in relations, etc.

A modal description from a world w_i with respect to a model M, is a set of sentences Δ such that the minimal model that satisfies Δ is M. A modal description contains in a sense, all the information that the model has.

We will give a procedure to obtain the description of a connected component of the graph. If the graph has more than a connected component, it is enough to calculate the description of each one and just join the sets of formulas obtained.

Given a model M = < W, R_1, ..., R_n, P > and a world $w_i \in W$, we want to get the modal description Δ from w_i. We will construct Δ in stages. Δ will have for each edge, a formula that identifies it. Clearly, the relation $R = \bigcup_{i \in I}(R_i \cup R_i^{-1})$ is connected,

then there is a minimal path beginning in w_i that contains each edge in the model. We will obtain Δ incrementally, considering in each step n the edges belonging to minimal paths of length n, beginning in w_i.

Consider the path of length 0. To represent it, we have just to identify the world w_i, so we put $\Delta_0 = \{p_i\}$, where $\models^M_{w_i} p_i$.

We assume constructed Δ_n. That is, given a path $P = \{(w_{j0}, w_{j1}), ..., (w_{jn-1}, w_{jn})\}$ of length n, with $w_{j0} = w_i$, we have a formula $\varphi \in \Delta_n$ that identifies the edge (w_{jn-1}, w_{jn}).

To construct Δ_{n+1} we have to add to Δ_n a formula for every edge at a minimum distance of n+1. Let $e = (w_{jn}, w_{jn+1})$ be one of such edges. Let $P' = \{(w_{j0}, w_{j1}), ..., (w_{jn}, w_{jn+1})\}$ be the minimal path that contains e. The edge $(w_{jn}, w_{jn+1}) \in R$ and then $(w_{jn}, w_{jn+1}) \in R_i$ or $(w_{jn}, w_{jn+1}) \in R_{i-1}$ for R_i in *M*. Then, we have to add to Δ_{n+1} the formula $\varphi' = \varphi[p_{jn}/(p_{jn} \wedge \langle h \rangle p_{jn+1})]$ (if $(w_{jn}, w_{jn+1}) \in R_i$) or $\varphi' = \varphi[p_{jn}/(p_{jn} \wedge \langle h \rangle_i p_{jn+1})]$ (if $(w_{jn}, w_{jn+1}) \in R_i^{-1}$).

Example: Given the model obtained from Figure N°2, we show the construction of the modal description from w_1:

$M = < W, R_/, R_{//}, P >$ con:

$W = \{w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8\}$
$R_/ = \{(w_2, w_1), (w_3, w_1), (w_6, w_4), (w_8, w_6), (w_7, w_5)\}$
$R_{//} = \{(w_2, w_6), (w_6, w_8), (w_7, w_3), (w_8, w_7), (w_2, w_3)\}$
$P = \{$(Check_CRC_Error, w_1), (Compute_CRC_Value, w_2), (Show_Error_Msg, w_3), (Math_Package, w_4), (I/O_Package, w_5), (CRC_Algorithm, w_6), (Retry_Operation, w_7), (Data_Input, w_8)$\}$
$R = R_/ \cup R_/^{-1} \cup R_{//} \cup R_{//}^{-1}$

Paths of length 0:
Δ_0 = {Check_CRC_Error}

Paths of length 1:
$\{(w_1, w_2)\}$
φ' = Check_CRC_Error$[^{\text{Check_CRC_Error}}/_{(\text{Check_CRC_Error} \wedge </>_i \text{ Compute_CRC_Value})}]$
$\{(w_1, w_3)\}$
φ' = Check_CRC_Error$[^{\text{Check_CRC_Error}}/_{(\text{Check_CRC_Error} \wedge </>_i \text{ Show_Error_Msg})}]$
$\Delta_1 = \Delta_0 \cup \{$ (Check_CRC_Error $\wedge$ $</>_i$ Compute_CRC_Value),
(Check_CRC_Error $\wedge$ $</>_i$ Show_Error_Msg)$\}$

Paths of length 2:
$\{(w_1, w_2), (w_2, w_6)\}$
φ' = (Check_CRC_Error $\wedge$ $</>_i$ Compute_CRC_Value)$[^{\text{Compute\_CRC\_Value}}/_{(\text{Compute\_CRC\_Value} \wedge </> \text{ CRC\_Algorithm})}]$
$\{(w_1, w_3), (w_3, w_7)\}$
φ' = (Check_CRC_Error $\wedge$ $</>_i$ Show_Error_Msg)$[^{\text{Show\_Error\_Msg}}/_{(\text{Show\_Error\_Msg} \wedge </>_i \text{ Retry\_Operation})}]$
$\{(w_1, w_2), (w_2, w_3)\}$
φ' = (Check_CRC_Error $\wedge$ $</>_i$ Compute_CRC_Value)$[^{\text{Compute\_CRC\_Value}}/_{(\text{Compute\_CRC\_Value} \wedge </> \text{ Show\_Error\_Msg})}]$
$\Delta_2 = \Delta_1 \cup \{$ (Check_CRC_Error $\wedge$ $</>_i$ (Compute_CRC_Value $\wedge$ </> CRC_Algorithm)),
(Check_CRC_Error $\wedge$ $</>_i$ (Show_Error_Msg $\wedge$ $</>_i$ Retry_Operation)),
(Check_CRC_Error $\wedge$ $</>_i$ (Compute_CRC_Value $\wedge$ </> Show_Error_Msg))$\}$

Paths of length 3:
$\{(w_1, w_2), (w_2, w_6), (w_6, w_4)\}$
φ' = (Check_CRC_Error $\wedge$ $</>_i$ (Compute_CRC_Value $\wedge$ </> CRC_Algorithm))$[^{\text{CRC\_Algorithm}}/_{(\text{CRC\_Algorithm} \wedge </> \text{ Math\_Package})}]$
$\{(w_1, w_2), (w_2, w_6), (w_6, w_8)\}$
φ' = (Check_CRC_Error $\wedge$ $</>_i$ (Compute_CRC_Value $\wedge$ </> CRC_Algorithm))$[^{\text{CRC\_Algorithm}}/_{(\text{CRC\_Algorithm} \wedge </> \text{ Data\_Input})}]$
$\{(w_1, w_3), (w_3, w_7), (w_7, w_5)\}$
φ' = (Check_CRC_Error $\wedge$ $</>_i$ (Show_Error_Msg $\wedge$ $</>_i$ Retry_Operation))$[^{\text{Retry\_Operation}}/_{(\text{Retry\_Operation} \wedge </> \text{ I/O\_Package})}]$
$\{(w_1, w_3), (w_3, w_7), (w_7, w_8)\}$
φ' = (Check_CRC_Error $\wedge$ $</>_i$ (Show_Error_Msg $\wedge$ $</>_i$ Retry_Operation)) $[^{\text{Retry\_Operation}}/_{(\text{Retry\_Operation} \wedge </>_i \text{ Data\_Input})}]$
$\Delta_3 = \Delta_2 \cup \{$(Check_CRC_Error $\wedge$ $</>_i$ (Compute_CRC_Value $\wedge$ </> (CRC_Algorithm $\wedge$ </> Math_Package))),
(Check_CRC_Error $\wedge$ $</>_i$ (Compute_CRC_Value $\wedge$ </> (CRC_Algorithm $\wedge$ </> Data_Input))),
(Check_CRC_Error $\wedge$ $</>_i$ (Show_Error_Msg $\wedge$ $</>_i$ (Retry_Operation $\wedge$ </> I/O_Package))),
(Check_CRC_Error $\wedge$ $</>_i$ (Show_Error_Msg $\wedge$ $</>_i$ (Retry_Operation $\wedge$ $</>_i$ Data_Input))) $\}$

Paths of length 4:
{(w_1, w_2), (w_2, w_6), (w_6, w_8),(w_8, w_5)}
φ' = (Check_CRC_Error $\wedge$ </>$_I$ (Compute_CRC_Value $\wedge$ <//> (CRC_Algorithm $\wedge$ <//> Data_Input)))[$^{Data\_Input}/_{(Data\_Input \wedge$ </> I/O_Package)]
$\Delta_4 = \Delta_3 \cup$ { (Check_CRC_Error $\wedge$ </>$_I$ (Compute_CRC_Value $\wedge$ <//> (CRC_Algorithm $\wedge$ <//> (Data_Input $\wedge$ </> I/O_Package)))))}

4. Applications

Next, we will see two specific examples where we apply the procedures to detect design problems. Note that in general, the steps to follow in each example do not depend on the particular graph we are analyzing. The procedure can, therefore, be automated.

4.1 Checking Relations

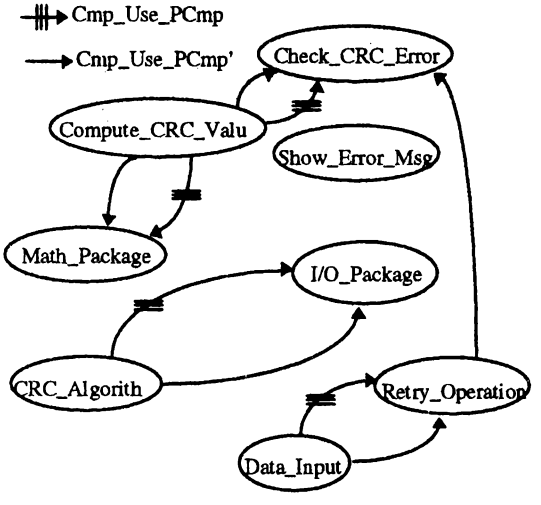

Figure N°3

Suppose that the original design contained the relation "Component_Use_Principal_Component" with the intended meaning of a non principal component using some function of a principal component. But suppose that there was an error and the arrow linking Retry_Operation and Check_CRC_Error (justified by the calling to Show_Error_Msg) is missing. The modal model obtained will be M' = < W', $R_/'$, $R_{//}'$, $R_{///}'$, P' > with:

W' ={w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8}
$R_/'$ ={(w_2, w_1), (w_3, w_1), (w_6, w_4), (w_8, w_5), (w_7, w_5)}
$R_{//}'$ ={(w_2, w_6), (w_6, w_8), (w_7, w_3), (w_8, w_7), (w_2, w_3)}
$R_{///}'$ ={(w_2, w_4), (w_6, w_5), (w_8, w_5), (w_2, w_1)}
P' ={(Check_CRC_Error, w_1), (Compute_CRC_Value, w_2), (Show_Error_Msg, w_3), (Math_Package, w_4), (I/O_Package, w_5), (CRC_Algorithm, w_6), (Retry_Operation, w_7), (Data_Input, w_8)}

We can now detect the error in the design using our tools. If the relation $R_{///}'$ were correct, that is, if it corresponded to the expected meaning, $R_{///}'$ should be the composition of the relation $R_{//}'$ with $R_/'$. Thus we define: <>p $\equiv_{def}$ <//></>p

If we filter M' under the logic $L_{<>, </>}$ we obtain the model M'^* of Figure N°3:

W'* = {{w_1}, {w_2}, {w_3}, {w_4}, {w_5}, {w_6}, {w_7}, {w_8}}
R'* = {({w_2}, {w_4}), ({w_6}, {w_5}), ({w_8}, {w_5}), ({w_2}, {w_1}), ({w_7}, {w_1})}
$R_{///}'^*$ = {({{w_2}, {w_4}), ({w_6}, {w_5}), ({w_8}, {w_5}), ({w_2}, {w_1})}
P'* = {(Check_CRC_Error, {w_1}), (Compute_CRC_Value, {w_2}), (Show_Error_Msg, {w_3}), (Math_Package, {w_4}), (I/O_Package, {w_5}), (CRC_Algorithm, {w_6}), (Retry_Operation, {w_7}), (Data_Input, {w_8})}

Then $R_{///}$ is correct if the formula <>p $\leftrightarrow$ <///>p is valid in the model or equivalently that $\vdash_\Delta$ <>p $\leftrightarrow$ <///>p holds for every description Δ obtained from a world in M'. But this is not the case, as is easy to see. For example in {w_7} we have <>Check_CRC_Error and $\neg$<///>Check_CRC_Error.

4.2 Cycle Detection

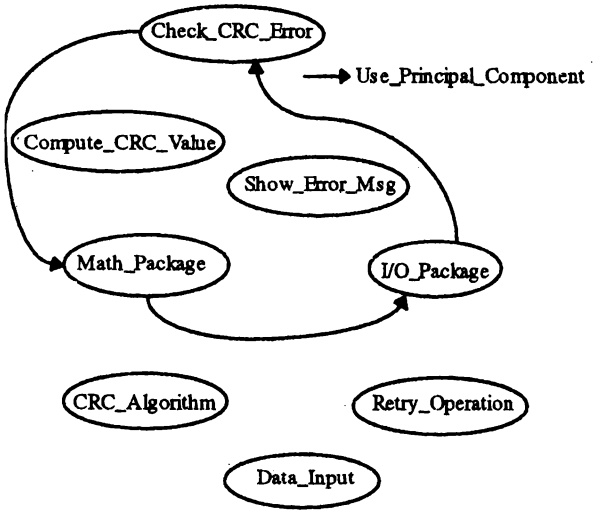

Figure N°4

An important fault of design consists in the violation of the hierarchy of components. This can be seen as the existence of cycles in the relation that involves the principal components (self loops).

Cycle detection is a basic step in a proof of design correctness and this example is usually used to test the power of a verifier.

In the example we are analyzing this relation is not explicit, and we must build it from the basic relations.

We propose the following definition:

$$<>p \equiv_{def} \neg p \wedge </>_{|}<//></>p$$

The formula $</>_{|}<//></>p$ lets us, by composition of relations, obtain the more general "Use_Principal_Component", while the formula $\neg p$ prevents that this relation includes calls inside the same principal component.

Now, we filter the model corresponding to Figure N°2 under the logic $L_{<>}$ and we obtain the model M^* of Figure N°4:

$W^* = \{\{w_1\}, \{w_2\}, \{w_3\}, \{w_4\}, \{w_5\}, \{w_6\}, \{w_7\}, \{w_8\}\}$
$R^* = \{(\{w_1\}, \{w_4\}), (\{w_4\}, \{w_5\}), (\{w_6\}, \{w_1\})\}$
$P^* = \{$(Check_CRC_Error, $\{w_1\}$), (Compute_CRC_Value, $\{w_2\}$), (Show_Error_Msg, $\{w_3\}$), (Math_Package, $\{w_4\}$), (I/O_Package, $\{w_5\}$), (CRC_Algorithm, $\{w_6\}$), (Retry_Operation, $\{w_7\}$), (Data_Input, $\{w_8\}$)$\}$

If we take the modal description Δ of M^* from $\{w_1\}$ and we obtain the transitivity closure adding the axiom $<><>p \rightarrow <>p$, we can prove $\vdash_\Delta$ Check_CRC_Error $\wedge$ <>Check_CRC_Error; and this shows that the component Check_CRC_Error from the original design was part of a cycle.

Repeating this procedure for every world of the model we obtain the set of the components with this characteristic.

5. Conclusions

As the examples show, the methods proposed seem to be effective. It is also important the simplicity with which they can be implemented (we could use any syntactic prover that let us define new modal operators). Of course, the use of a prover takes the complexity to exponential levels, but on the other side, the proofs are always done over modal descriptions that have a low number of axioms (linear in the number of edges in the model). In exchange, we gain a formal tool of great power and generality.

6. Future Work

It is possible that the present work may be applied also to the description of Software Architectures [Garlan & Shaw, 1993]. In this area, anyway, the problem has more complexity. The representation graphs used seem to capture well the static components of the system, but not so the dynamic components that define the interactions between the components [Allen & Garlan, 1994]. In this graph, the relations take a fundamental importance. For example, from the fact that two components are in the client-server relation, we can infer the behavior of the components and the way they communicate (the client makes requests to the server at any time; the server answers the requests without knowing who makes them, etc.).

Other point not covered in the present work, has to do with the detection of components with the same identification label. Some of these cases are easily treated in our formalism, but some modifications may be needed to cover all the possibilities.

There are also importat work to do in formal concept directly related to modal and temporal logics with models with more than one relation of accessibility.

References

- [Allen & Garlan, 1994]
  Allen, R. & Garlan, D. *Formalizing Architectural Connection.*
  Proc. International Conference on Software Engineering. May 1994.
- [Burguess, 1984]
  Burguess, John P. *Basic Tense Logic.*
  Handbook of Philosophical Logic, Vol.II. D. Reidel Publishing Company. 1984.
- [Chellas, 1980]
  Chellas, Brian F. *Modal Logic, An Introduction.*
  Cambridge University Press. 1980.
- [Consens & Mendelzon, 1992]
  Consens, M. & Mendelzon, A. *Visualizing and Querying Software Structures.*
  ACM. 1992.
- [Garlan & Shaw, 1993]
  Garlan, D. & Shaw, M. *An Introduction to Software Architecture.*
  Advances in Software Engineering and Knowledge Engineering, Vol I. World Scientific Publishing Company. 1993.
- [Hughes & Cresswell, 1968]
  Hughes, G. E. & Cresswell, M. J. *An Introduction to Modal Logic.*
  Methuen and Co. Ltd. 1968.4. Applications