Reconciling Operational and Declarative Specifications

J. Hagelstein and D. Roelants
{hagelste,roelants}@sema.be

Sema Group Belgium
5, Place du Champ de Mars
B-1050 Brussels, Belgium

Abstract. There are two broad approaches to the specification of the dynamics of information systems, namely the operational and the declarative one. The declarative approach has advantages in terms of abstractness and of locality of information. Its main drawback is the so called frame problem, i.e. the need to explicitly forbid unwanted changes. We propose an extension of the declarative approach which incorporates some aspects of the operational one, thereby eliminating the frame problem. The technique is illustrated on an example, and the resulting specification is compared with the corresponding operational and declarative ones.

1 Introduction

The requirements engineering of information systems includes an identification of the structure of the relevant information, and of its evolution over time, or dynamics. There are two broad approaches to the description of these dynamics, namely the *operational* one and the *declarative* one, of which the *deductive* approach is a variant.

The declarative, or logic-based, approach uses logical formulas to constrain the evolution of information. It is supported by such languages as CIAM [GKB82], RML [GBM86], Infolog [FS85], or ERAE [Hag88]. This approach generalises the concept of integrity constraint to the temporal dimension: the same language is used to characterise valid states ('all salaries are greater than 5000') as well as valid changes ('salaries may only grow'). This is illustrated below using a function *salary* from types *Person* and *Time* to *Salary*. The logic used is first order logic.

$salary(Person, Time) : Salary$

% all salaries are greater than 5000
$\forall p \forall t\ (salary(p,t) > 5000)$

% salaries may only grow
$\forall p \forall t_1\ \forall t_2\ (t_1 > t_2 \Rightarrow salary(p,t_1) \geq salary(p,t_2))$

This work was partly funded by the Commission of the European Communities under the ESPRIT project ICARUS.

A variant of first order logic called *temporal logic* is used by some authors [Hag88] [DHR90] [FS86] to avoid the explicit reference to time. The formal semantics of temporal logic are recalled in Section 3.1, but their intuitive meaning is the following. Temporal logic formulas are interpreted in sequences of successive states, the valid sequences being those where formulas hold in every state. Most functions and predicates, like ***salary***, are state-dependent, i.e. they may have different interpretations in different states. Temporal logic formulas may contain special operators ($\circ$, $\bullet$, $\Diamond$, etc) which cause the subsequent sub-formula or term to be evaluated in a different state. For instance, the term '$\bullet\ \tau$' denotes the value of τ in the previous state. The temporal logic version of the specification above can be written:

$$salary(Person) : Salary$$
$$\forall p\ (salary(p) > 5000)$$
$$\forall p\ (salary(p) \geq \bullet\ salary(p))$$

The effect of an event is among the dynamic constraints that declarative languages can express. An event is modelled by a predicate which holds whenever the event occurs. The example above may be extended to express that an event ***promotion*** induces a salary increase:

$$promotion(Person) : Event$$
$$\forall p\ (promotion(p) \Rightarrow salary(p) = 1.1\ *\ \bullet\ salary(p))$$

The declarative approach has many advantages in terms of abstractness and locality of information [Oli86]. It suffers however from a drawback known as the ***frame problem*** [Min74]: it is not sufficient to require the needed changes; one must also exclude the undesired ones. The last formula above forces the salary to vary in case of promotion, but does not prevent it to change freely at other moments. To this end, we must add the following formula:

$$\forall p\ (salary(p) \neq \bullet\ salary(p) \Rightarrow promotion(p))$$

The deductive approach [Oli86] is a variant of the declarative approach which suffers however from the same drawback. It will be discussed in Section 4.

The operational approach controls the evolution of information in a program-like style. Typically, events trigger the execution of routines explicitly modifying the information. This approach avoids the frame problem through the implicit assumption that no change takes place, except if an operation explicitly requires it. However, it suffers from other drawbacks, the first one being a lack of abstractness. A property like 'salaries never decrease' cannot be directly expressed; it can only be deduced from a careful analysis of all routines. Its other weaknesses will be discussed in Section 4.

In face of this situation where the declarative and operational approaches have good qualities and drawbacks, we propose to extend the former in a way that circumvents the frame problem. The suggested technique is introduced in Section 2, and illustrated on a

simple language called MINI-FOL. This language is given formal semantics in Section 3. It is compared to the operational, purely declarative, and deductive approaches in Section 4. Further extensions are proposed in the conclusion.

2 The MINI-FOL language

2.1 Active and Reactive Happenings

The difference between the declarative and operational approaches may be synthesised in the following two principles, as far as the specification of dynamics is concerned:

> *Declarativeness principle: any change is allowed, except those forbidden by the specification.*
>
> *Operationality principle: any change is forbidden, except those imposed by the specification.*

When specifying an information system, one seems to need the two principles in turn. There are indeed two categories of happenings that the specification is concerned with: the *active* ones and the *reactive* ones.

- An active happening is one that the planned information system will not control, typically an action of the user. Some authors call these happenings *external events*. The specification can express hypotheses that limit their occurrences, but it cannot force them to happen. The declarativeness principle suits their description optimally.
- A reactive happening is one that the information system will control. It happens in reaction (hence 'reactive') to an active happening and should only take place if requested. The specification will impose its occurrence in strictly limited circumstances. The operationality principle is better suited to describe this second category of phenomena.

As an example, consider an information system maintaining information about projects, and departments responsible for them. The start of a project in a department, and the end of a project are active events. The specification may not force them to happen; it may just prevent them from happening in certain circumstances. The record of which projects are running at any moment is reactive. The specification will force it to change when start or end events occur.

2.2 Description of MINI-FOL

The contradiction between the optimal handling of active and reactive happenings is resolved in the language MINI-FOL. This language is not meant as a complete specification language, but rather as a toy language illustrating a certain extension of the declarative approach. This extension is applicable to any practical declarative specification language.

Let us start from the example above, which can be modelled in temporal logic by means of the components below. (From now on, we use *component* to denote a function or predicate.

$$Start(Project, Department) : Event$$
$$End(Project) : Event$$
$$running(Project) : Boolean$$

An attempt at specifying the value of *running* may lead to the following formulas:

$$\forall p \forall d \; (Start(p, d) \Rightarrow running(p))$$
$$\forall p \; (End(p) \Rightarrow \neg running(p))$$

By convention, the universal quantifications at the outermost of formulas may be omitted. These formulas can thus be rewritten as follows.

$$Start(p, d) \Rightarrow running(p)$$
$$End(p) \Rightarrow \neg running(p)$$

In a purely declarative approach, these formulas are not sufficient: they only constrain the value of *running* when the events *Start* and *End* occur, and at no other moment. The formulas do not express that *running* holds from a *Start* to the subsequent *End*. The correct formulas are substantially more complex.

Still, these naive formulas are appealing and they would be correct if we could add that *running* is reactive, in the sense of Section 2.1, and only changes if required. MINI-FOL allows it through the following extensions:

- Functions and predicates are partitioned into active and reactive ones.
- Formulas are similarly partitioned into *action restrictions* and *reaction conditions*.

Active and reactive components are declared in two separate sections introduced by the keywords **active** and **reactive**, respectively:

active
$$Start(Project, Department) : Event$$
$$End(Project) : Event$$

reactive
$$running(Project) : Boolean$$

The reaction conditions are given in a section introduced by the keywords **reaction condition**. This section associates each reactive component with a set of formulas controlling its changes:

reaction condition
$$running : Start(p, d) \Rightarrow running(p)$$
$$End(p) \Rightarrow \neg running(p)$$

There may be two other sections in a MINI-FOL specification. One, introduced by the keywords **action restriction**, lists the action restrictions. In the example above, we could require that only running projects be ended:

action restriction
$End(p) \Rightarrow \bullet\ running(p)$

This formula cannot cause the event *End* to occur, nor does it cause ***running*** to change. It is only meant to prevent *End* from occurring.

Finally, a section introduced by the keyword **initially** lists conditions that must hold in the initial state, in addition to the other formulas. For example, that there is initially no running project:

initially
$\neg running(p)$

We postulate that all events are false in the initial state. The specification is thus interpreted as if the initially section implicitly contained the formula expressing this fact. In the example,

$\neg Start(p, d) \wedge \neg End(p).$

The semantics of MINI-FOL ensure that the declarativeness principle holds for the active components and the operationality principle holds for the reactive ones. A MINI-FOL specification admits a subset of the models obtained with the standard temporal logic interpretation, namely those that minimise the changes from state to state (hence its name which stands for MINImalised First Order Logic). These minimally changing models are those whose state transitions can be obtained as follows: first, the active components are changed within the limits of the action restrictions; after this, the reactive components may change, but only for the elements of their domain invalidating the reaction condition. In the example, *running* will only change for a project p if either $Start(p, d)$ occurs while p is not running, or $End(p)$ occurs while p is running.

In general, however, the situation is slightly more complex than suggested by this simple example. Let us introduce a second reactive component, the function *dept* giving the department that is responsible for a project:

reactive
$dept(Project) : Department$

This department is the one in which the project was started, or is ***none*** if the project is not running. The reaction conditions for *dept* are the following ones:

reaction condition
$dept : Start(p, d) \Rightarrow dept(p) = d$
$\neg running(p) \Rightarrow dept(p) = none$

One observes that the reaction conditions of *dept* mention the reactive predicate *running*, which induces a dependency between the two reactive components. Therefore, a state transition generally consists in changes to the active components, followed by a *reaction chain* during which the reactive components are changed in sequence. If a component x depends on another y, then x must be changed after y in the chain. The reactive components must therefore be ordered by the specifier in the reactive section:

reactive
running(*Project*) : *Boolean*
dept(*Project*) : *Department*

$running \prec dept$

In the context of this ordering, the models of a MINI-FOL specification are the temporal logic models whose state transitions can be obtained as follows:

- active components are first changed within the limits permitted by the action restrictions;
- after this, the reactive functions and predicates are considered in any order compatible with their partial ordering; each of them is changed minimally to restore the associated reaction conditions; 'changed minimally' means that the value of the function or predicate is changed for as few elements in its domain as possible.

Section 3 formalises these semantics.

The order $\prec$ may be partial, because independent components need not be ordered. The following well-formedness rules ensure that this ordering is consistent with its intended purpose.

- First, $\prec$ must indeed define a partial order, i.e. its transitive closure may not contain any loop.
- If a component c is constrained by a statement ϕ, then ϕ may only refer to another component c' if this component is guaranteed to be updated before c. This is however not necessary if c' occurs within the scope of $\bullet$, in which case no dependency is induced.

  This rule has the following implications:

  - reactive components may not occur in action restrictions, except in the scope of $\bullet$;
  - a reactive component x occurring in the reaction condition of a different component y must verify $x \prec^* y$ (where $\prec^*$ is the transitive closure of $\prec$), except if x occurs within the scope of $\bullet$;
  - in any of its reaction conditions, a reactive predicate or function x must have its arguments bound to the same variables in all its occurrences not within the scope of $\bullet$; this is to avoid dependencies between the values of a predicate or function for different elements of its domain.

In practice, these rules are easily obeyed, provided the dependencies between components are well identified.

Although it is not critical, it often simplifies the specifications to assume that only one event occurs, or only one other active component changes from one state to the next. This property is taken into account by the following implicit formula, required to hold in all but the initial state:

$$\bigvee_{i=1}^{n'} (\exists!p_i\ x_i(p_i) \wedge \bigwedge_{j=1,j\neq i}^{n'} \forall p_j\ \neg x_j(p_j) \wedge \bigwedge_{j=n'+1}^{n} \forall p_j\ x_j(p_j) = \bullet\, x_j(p_j))$$
$$\vee$$
$$\bigvee_{i=n'+1}^{n} (\exists!p_i\ x_i(p_i) \neq \bullet\, x_i(p_i) \wedge \bigwedge_{j=n'+1,j\neq i}^{n} \forall p_j\ x_j(p_j) = \bullet\, x_j(p_j) \wedge \bigwedge_{j=1}^{n'} \forall p_j\ \neg x_j(p_j))$$

where '$\exists!$' means 'there exists exactly one', $x_1, \ldots, x_{n'}$ are all active events and $x_{n'+1}, \ldots, x_n$ are the other active components, and where, for $i \in \{1, \ldots, n\}$, p_i is a tuple of variables on the domain of x_i.

2.3 A Larger Example

In this section, we give the complete MINI-FOL specification of an extension of the example used in the previous sections.

> The planned information system maintains information about projects running in departments and programmers assigned to these projects. Projects can be started in a department and ended; programmers can be assigned to or removed from a project; departments can receive the responsibility for a project or lose it.
>
> The information system must record which projects are running, which programmer works for which project, which department is responsible for which project, and which programmer has ever worked for which department.
>
> It must prevent the ending of non running projects, the start of a running project, and the removing of a programmer from a project to which he or she is not assigned.

The corresponding MINI-FOL specification is the following:

active

Start(Project, Department) : *Event*
End(Project) : *Event*
Assign(Programmer, Project) : *Event*
Remove(Programmer, Project) : *Event*

reactive

running(Project) : *Boolean*
dept(Project) : *Department*

$assigned(Programmer, Project) : Boolean$
$has\text{-}worked(Programmer, Dept) : Boolean$

$running \prec assigned \prec dept \prec has\text{-}worked$

initially

$\neg running(p)$
$\neg has\text{-}worked(pg, d)$

action restriction

$End(p) \Rightarrow \bullet\ running(p)$
$Start(p, d) \Rightarrow \bullet\ \neg running(p)$
$Remove(pg, p) \Rightarrow \bullet\ assigned(pg, p)$

reaction condition

$running$: $Start(p, d) \Rightarrow running(p)$
$End(p) \Rightarrow \neg running(p)$

$dept$: $Start(p, d) \Rightarrow dept(p) = d$
$\neg running(p) \Rightarrow dept(p) = none$

$assigned$: $Assign(pg, p) \Rightarrow assigned(pg, p)$
$Remove(pg, p) \Rightarrow \neg assigned(pg, p)$
$\neg(\neg running(p) \wedge assigned(pg, p))$

$has\text{-}worked$: $assigned(pg, p) \wedge dept(p) = d \Rightarrow has\text{-}worked(pg, d)$

It can be verified that the various formulas comply to the well-formedness conditions. In this case, the declared ordering of reactive components is slightly stronger than needed, as *assigned* and *dept* are actually independent of each other. This strengthening does no harm.

Notice that the usual temporal logic deductions are still valid, as all action restrictions and reaction conditions hold in all states. In particular, we can conclude from the initially section that no project is initially assigned to any department

$$dept(p) = none$$

and that no programmer is initially assigned to any project

$$\neg assigned(pg, p).$$

3 Formal semantics of MINI-FOL

This section presents two equivalent formal semantics for MINI-FOL. The first one is constructive and specifies the elaboration of a model state by state. This semantics is probably the most intuitive. The other is a rewrite semantics, specifying the formulas

that must be added to a MINI-FOL specification to reduce it to a classical temporal logic specification. This clearly identifies the expressive power of MINI-FOL and guarantees that it is amenable to deduction and theorem proving. The proof of the equivalence of the two semantics is in appendix.

3.1 Temporal Logic

MINI-FOL is an extension of a linear temporal logic which is first defined in this section. Its models are right-infinite sequences of states of the form $\Sigma = (\sigma_m : m \in N)$. As in first order logic, a state is a valuation function for variable, function and predicate symbols. The valuation of variables is identical in all states of a sequence.

A formula holds in a model Σ if its value is true with respect to every state of this model. The value of the formula ϕ with respect to the i^{th} state of a model Σ is noted '$val(\Sigma, i, \phi)$' The value of a term τ is noted '$val(\Sigma, i, \tau)$'. The function val is defined recursively as follows (p, f, and x denote respectively a predicate symbol, a function symbol, and a variable):

$val(\Sigma, i, x) = \sigma_i(x)$

$val(\Sigma, i, f(\tau_1, \ldots, \tau_n)) = \sigma_i(f)(val(\Sigma, i, \tau_1), \ldots, val(\Sigma, i, \tau_n))$

$val(\Sigma, i, p(\tau_1, \ldots, \tau_n)) = \sigma_i(p)(val(\Sigma, i, \tau_1), \ldots, val(\Sigma, i, \tau_n))$

$val(\Sigma, i, \neg\phi) = \text{true}$ iff $val(\Sigma, i, \phi) = \text{false}$

$val(\Sigma, i, \phi_1 \wedge \phi_2) = \text{true}$ iff $val(\Sigma, i, \phi_1) = \text{true}$ and $val(\Sigma, i, \phi_2) = \text{true}$

$val(\Sigma, i, \forall x \phi) = \text{true}$ iff $val(\Sigma', i, \phi) = \text{true}$ for all Σ' differing from Σ at most by the value assigned to x in all states

$val(\Sigma, i, \bullet \tau) = \text{if } i > 0 \text{ then } val(\Sigma, i\text{-}1, \tau) \text{ else } val(\Sigma, 0, \tau)$

$val(\Sigma, i, \bullet \Phi) = \text{if } i > 0 \text{ then } val(\Sigma, i\text{-}1, \Phi) \text{ else } val(\Sigma, 0, \Phi)$

Additional operators ($\exists$, $\vee$) can be defined in terms of these. We also need the predefined predicate *initially* which holds only in the first state of a sequence. Its semantics is given by the following property:

$\sigma_i(initially) = \text{true}$ iff $i = 0$.

3.2 Constructive Semantics

Let S be a MINI-FOL specification, where the implicit formulas restricting the simultaneous changes have been added to the action restriction section (prefixed by '$\neg$ *initially* $\Rightarrow$ ...'):

active $x_1 \ldots x_n$
reactive $u_1 \ldots u_k$
initially Γ
action restriction Ψ
reaction condition $u_i : \forall p_i \; \Phi_i(p_i) \quad (1 \leq i \leq k)$

For each $i \in \{1, \ldots, k\}$, p_i is a tuple of variables on the domain of u_i. We assume that the partial order between reactive components is such that if $u_i \prec u_j$ then $i < j$. The specification is well-formed, which implies that

- any u_i in Ψ is under the scope of $\bullet$;
- any u_j $(j > i)$ and any $u_i(p_i')$ $(p_i' \neq p_i)$ in $\Phi_i(p_i)$ is under the scope of $\bullet$.

The constructive semantics reflects the following intuition. A model of a specification is a sequence of states, where every next state is obtained in two steps. First some arbitrarily chosen active components are changed without invalidating the action restrictions. Then, the reactive components are checked and updated if necessary, starting with the smallest one.

Formally, a sequence of states $(\sigma_i : i \in \mathbf{N})$ is a model for the specification S if σ_0 satisfies

$$\Gamma \wedge \Psi \wedge \bigwedge_{i=1}^{k} \forall p_i \; \Phi_i(p_i)$$

and if the following conditions hold for any two successive states σ and σ':

- there is an intermediate state ρ_0 which is obtained from σ by changing the values assigned to the active components in a way that preserves Ψ;
- there is a sequence of intermediate states $\rho_1, \ldots, \rho_k$, where $\rho_k = \sigma'$, verifying the following properties:
  - ρ_i $(i > 0)$ differs from ρ_{i-1} at most by the value assigned to u_i;
  - consider the p_i in the domain of u_i in any order; if the formula $\Phi_i(p_i)$ evaluates to true in state ρ_{i-1}[1], then $\rho_i(u_i(p_i)) = \rho_{i-1}(u_i(p_i))$, else $\rho_i(u_i(p_i))$ is any value such that $\Phi_i(p_i)$ is true at state ρ_i.

This semantics may only be called constructive if the domains of all reactive components are finite. This restriction is quite natural in the field of information systems specification.

To distinguish σ and σ' from the states ρ_i which do not belong to the model of the specification, we call the former *observable* states and the latter *intermediate* states. It is proven in appendix that all action restrictions and reaction conditions hold in all observable states.

3.3 Rewrite Semantics

The purpose of these second semantics is to identify the formulas that must be added to a MINI-FOL specification, to reduce it to a specification in the temporal logic of Section 3.1. We will need the following notation:

[1]The evaluation is according to the rules of temporal logic as defined in Section 3.1, with ρ_{i-1} considered the successor of σ.

$\Phi[y : c]$ means that each occurrence of y which is not under the scope of $\bullet$ in the formula Φ is replaced by c; $\Phi[y_1 : c_1, y_2 : c_2] = (\Phi[y_1 : c_1])[y_2 : c_2]$.

The initial state should satisfy the initially section, the action restrictions, and the reaction conditions:

$$initially \Rightarrow \Gamma \wedge \Psi \wedge \bigwedge_{i=1}^{k} \forall p_i \ \Phi_i(p_i)$$

In each other state, an active component is chosen and its value may be changed without invalidating the action restrictions. We therefore have:

$$\neg initially \Rightarrow \Psi$$

During the subsequent reaction chain, all reaction conditions must be satisfied, and a reactive component u_i should only change if its associated formula is invalidated.

$$\neg initially \Rightarrow \bigwedge_{i=1}^{k} \forall p_i \ (\Phi_i(p_i) \wedge (\Phi_i(p_i)[u_i(p_i) : \bullet u_i(p_i)] \Rightarrow u_i(p_i) = \bullet u_i(p_i)))$$

We can simplify these formulas into the four following ones:

$$initially \Rightarrow \Gamma$$
$$\Psi$$
$$\bigwedge_{i=1}^{k} \forall p_i \ \Phi_i(p_i)$$
$$\bigwedge_{i=1}^{k} \forall p_i \ (\Phi_i(p_i)[u_i(p_i) : \bullet u_i(p_i)] \Rightarrow u_i(p_i) = \bullet u_i(p_i))$$

The reduction of a MINI-FOL specification to temporal logic therefore consists in converting the initially section into its obvious counterpart, in taking the action restrictions and reaction conditions, and in adding the formula

$$\bigwedge_{i=1}^{k} \forall p_i \ (\Phi_i(p_i)[u_i(p_i) : \bullet u_i(p_i)] \Rightarrow u_i(p_i) = \bullet u_i(p_i))$$

which expresses the minimisation of changes in the reactive components.

As an example, the formula that corresponds to the predicate *has-worked* is the following (p_i is (pg, d) for this predicate):

$$\forall pg \ \forall d \ (\forall p \ (assigned(pg, p) \wedge dept(p) = d \Rightarrow \bullet \ has\text{-}worked(pg, d)) \\ \Rightarrow has\text{-}worked(pg, d) = \bullet \ has\text{-}worked(pg, d))$$

4 Comparison

This section uses the example of Section 2.3 to highlight the differences between MINI-FOL and the operational approach, the purely declarative one, and the deductive one.

4.1 The Operational Approach

In an operational approach, external (active) happenings trigger routines which modify the internal (reactive) variables. For example, the routine triggered by the event *End(p)* may be written as follows:

if *running(p)*
then *running(p)* := *false*
dept(p) := *none*
foreach *pg*
do *assigned(pg, p)* := *false*

A first difference between this specification and the MINI-FOL one is that the routine associated to *End* must consider all possible consequences of this event, be they direct (stopping a project) or indirect (setting the project department to *none* and unassigning all programmers). In comparison, the MINI-FOL approach cuts reaction chains into pieces handled by separate formulas: one saying that *End(p)* sets *running(p)* to false; another to say that a stopped project has *none* as department; etc.

The need to consider all consequences at once reduces the readability of operational specifications: for example, the conditions under which *assigned* changes must be gathered from the routines triggered by the events *End*, *Assign* and *Remove*. The maintainability is reduced as well: if the definition of *assigned* changes, the operational specification must be adapted in all these places. These drawbacks are of course amplified when the size of the specification grows.

Besides, the operational style suffers from its inherent inability to directly state crucial properties like 'a project which is not running has no assigned programmer'. Such properties can only be deduced from a careful analysis of the specification.

Note that the advantages of MINI-FOL over the operational style are shared by the other non-operational approaches, i.e. the declarative and deductive ones.

4.2 The Declarative Approach

A purely declarative specification written in first order or temporal logic contains the same formulas as the MINI-FOL one, plus those required to explicitly circumvent the frame problem. These formulas, which must prevent the undesired change of reactive variables, could be the following ones:

$$\neg Start(p, d) \wedge \neg End(p) \Rightarrow (running(p) \Leftrightarrow \bullet\, running(p))$$
$$\neg Start(p, d) \wedge (running(p) \Leftrightarrow \bullet\, running(p)) \Rightarrow dept(p) = \bullet\, dept(p)$$
$$\neg Assign(pg, p) \wedge \neg Remove(pg, p) \wedge running(p) \Rightarrow (assigned(pg, p) \Leftrightarrow \bullet\, assigned(pg, p))$$
$$has\text{-}worked(pg, d) \Leftrightarrow (\bullet\, has\text{-}worked(pg, d) \vee \exists p\, (assigned(pg, p) \wedge dept(p) = d))$$

Each formula is devoted to a specific reactive component, and states that it may not change outside of the conditions known to require a change. For example, the first formula prevents any change of *running*, if there is no *Start* or *End* event.

These formulas can be proven equivalent to those automatically generated by the rewrite semantics of MINI-FOL. Leaving these formulas implicit has several benefits. Of course, it rids the user of finding them, which is often not too easy. It also reduces the total size of the specification, which is always a win. Finally, it improves the maintainability of the specification, i.e. its ability to be easily changed. Suppose we add the possibility to change the department responsible for a project:

active
$ChangeDept(Project, Department) : Event$

action restriction
$ChangeDept(p, d) \Rightarrow \bullet\, running(p)$

reaction condition
$dept : ChangeDept(p, d) \Rightarrow dept(p) = d$

In the first-order logic approach, the mere adjunction of the last formula simply prevents any $ChangeDept$ to occur, because this would contradict the second frame formula above, namely:

$$\neg Start(p, d) \wedge (running(p) \Leftrightarrow \bullet\, running(p)) \Rightarrow dept(p) = \bullet\, dept(p).$$

This axiom must also be changed, and replaced by:

$$\neg Start(p, d) \wedge \neg ChangeDept(p, d) \wedge (running(p) \Leftrightarrow \bullet\, running(p)) \Rightarrow dept(p) = \bullet\, dept(p)$$

The need to change the frame axioms makes any modification more delicate, especially if the specification is large.

4.3 The Deductive Approach

The deductive approach [Oli89] is a variant of the declarative approach. External events are modelled by adding or removing *base* predicates, corresponding to the MINI-FOL active predicates. Active functions have no counterpart. These changes are constrained by *integrity constraints*, similar to the MINI-FOL action restrictions. Information that must be maintained by the information system is represented by *derived* predicates, corresponding to the MINI-FOL reactive predicates. Reactive functions have no counterpart.

The main difference is in the expression of reaction conditions which are called *deduction rules* in the deductive approach. There is no assumption that reactive components change minimally; there is another assumption, sometimes called the *closed world assumption*, implying that the predicates are false in all states where they are not said to be true. For example, the derived predicate $assigned(pg, p)$ can be specified by the following deduction rules (notations in [Oli89] are adapted to temporal logic):

$$Assign(pg,p) \Rightarrow assigned(pg,p)$$
$$running(p) \wedge \neg Remove(pg,p) \wedge \bullet\, assigned(pg,p) \Rightarrow assigned(pg,p).$$

The closed world assumption implicitly adds

$$assigned(pg,p) \Rightarrow Assign(pg,p) \vee (running(p) \wedge \neg Remove(pg,p) \wedge \bullet\, assigned(pg,p))$$

The deductive approach allows some saving in writing, compared to the declarative approach, but it does not help avoiding the frame problem : a specifier still has to specify when a predicate does not change, as in the second formula above. As a consequence, a MINI-FOL specification is usually easier to write, more compact and more readable than a deductive specification.

Besides, the deductive approach does not allow the use of functions and restricts the forms of rules to ensure their efficient automatic interpretation. This limited form may obscure simple properties, or require to introduce intermediate predicates.

5 Conclusion

A declarative (logic-based) specification has advantages over an operational (assignment-based) one, in terms of modularity and expressiveness. The former suffers, however, from a drawback known as the *frame problem*, i.e. the need to explicitly state what may not change in addition to saying what must change. Actually, each approach is best suited to specify part of the dynamics of information systems. The declarative approach assumes that all changes are possible, except those explicitly excluded. This is optimal for specifying the events which are external to the information system, and constitute its input. The operational approach assumes that no change is allowed, except the explicitly required ones. This avoids the frame problem and is optimal for specifying the reactions of the information system.

We have proposed a technique that reconciles the two approaches. It extends the declarative approach by distinguishing two kinds of application-dependent functions and predicates, the active and reactive ones. The former are treated in the classical declarative style; they may change freely, except if constrained by formulas called action restrictions. The reactive functions and predicates may not change, except if forced to do so to maintain the truth of associated formulas called reaction conditions. This technique is illustrated in a simple language called MINI-FOL, which has been given formal semantics. The specification of a simple case study allows to compare this approach with the operational one, the purely declarative one, and a variant called the deductive approach.

We are currently working on several extensions of MINI-FOL. The first one consists in suppressing the requirement to order reactive components. This not only frees the specifier from some work, but also allows to treat the special cases where these components cannot be ordered, because of mutual dependencies. The second extension in sight is the inclusion of more temporal operators than just $\bullet$. Introducing other temporal operators referring to the past is almost straightforward. More delicate is the introduction of temporal operators referring to the future, like $\circ$, $\Diamond$, or $\Box$.

References

[DHR90] E. Dubois, J. Hagelstein, and A. Rifaut. *ERAE : A Formal Language for Expressing and Structuring Real-time Requirements.* Manuscript M 353, Philips Research Laboratory Belgium, 1990.

[FS86] J. Fiadeiro and A. Sernadas. The Infolog linear tense propositional logic of events and transactions. *Information Systems*, 11(1), 1986.

[GBM86] S.J. Greenspan, A. Borgida, and J. Mylopoulos. A declarative approach to conceptual information modeling. *Information Systems*, 11(1):9–23, 1986.

[GKB82] M.R. Gustafsson, T. Karlsson, and J.A. Bubenko. A declarative approach to conceptual information modeling. In T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, editors, *Information System Design Methodologies : A Comparative Review*, pages 93–142, North-Holland, 1982.

[Hag88] J. Hagelstein. Declarative approach to information systems requirements. *Knowledge Based Systems*, 1(4):211–220, 1988.

[Min74] M. Minsky. *A Framework for Representing Knowledge.* Artificial Intelligence Memo 306, MIT, 1974.

[Oli86] A. Olivé. A comparison of the operational and deductive approaches to conceptual information systems modeling. In *IFIP'86*, pages 91–96, North-Holland, 1986.

[Oli89] A. Olivé. On the design and implementation of information systems from deductive conceptual models. In Peter Apers and Gio Wiederhold, editors, *Proc. 15th International Conference on Very Large Databases*, pages 3–11, Amsterdam, The Netherlands, August 1989.

Appendix : Equivalence of rewrite and constructive semantics

Let S be the MINI-FOL specification of Section 3. Let $\Sigma = (\sigma_m : m \in N)$ be a temporal logic model. For practical reasons, we use the following notation to denote the truth of a formula Υ in Σ with respect to the m^{th} state :

$$\Sigma, m \models \Upsilon.$$

This notation is equivalent to '$val(\Sigma, m, \Upsilon)$ is true'.

Theorem

Each constructive model of S corresponds to a temporal logic model of the specification obtained from S by the rewrite semantics.

Proof : Let $\Sigma = (\sigma_m : m \in N)$ be a constructive model of S.

1. By definition of a constructive model, we have

   $$\Sigma, 0 \models \Gamma \wedge \Psi \wedge \bigwedge_{i=1}^{k} \forall p_i \; \Phi_i(p_i)$$

   and hence the following formula is true in Σ:

   $$\textit{initially} \Rightarrow \Gamma \wedge \Psi \wedge \bigwedge_{i=1}^{k} \forall p_i \; \Phi_i(p_i).$$

2. We have to prove $\Sigma, m \models \Psi$, for any $m > 0$.

   By definition of a constructive model, there exists a state ρ_0 obtained from σ_{m-1} by changing only the values assigned to active components, and such that ρ_0 satisfies Ψ (according to the rules of temporal logic as given in Section 3.1, with ρ_0 considered the successor of σ_{m-1}).

   Since the values assigned to $x_1, \ldots, x_n$ in σ_m are the same as in ρ_0 and since none of $u_1, \ldots, u_k$ occurs outside the scope of $\bullet$ in Ψ, Ψ cannot be invalidated by the subsequent changes of reactive components. Hence, $\Sigma, m \models \Psi$.

3. We prove that, for any $m > 0$, for any i $(1 \leq i \leq k)$,

   $$\Sigma, m \models \forall p_i \; (\Phi_i(p_i) \wedge (\Phi_i(p_i) | u_i(p_i) : \bullet \, u_i(p_i)] \Rightarrow u_i(p_i) = \bullet \, u_i(p_i))).$$

   By definition of a constructive model, there exists a sequence of states $(\rho_1, \ldots, \rho_k)$ such that ρ_i differs from ρ_{i-1} at most in the value assigned to u_i, and such that $\sigma_m = \rho_k$.

   As a consequence, for any i $(1 \leq i \leq k)$,

   $$\begin{array}{ll} \rho_i(u_g) = \sigma_m(u_g) & 1 \leq g \leq i \\ \rho_i(u_g) = \sigma_{m-1}(u_g) & i+1 \leq g \leq k \end{array} \tag{1}$$

   and hence, if ρ_i satisfies Υ, then

   $$\Sigma, m \models \Upsilon | u_{i+1} : \bullet \, u_{i+1}, \ldots, u_k : \bullet \, u_k]. \tag{2}$$

   Let p_i be in the domain of u_i. According to the constructive semantics, the value of $u_i(p_i)$ in state ρ_i is obtained differently, depending the case:

   (a) In the first case, $\Phi_i(p_i)$ is satisfied in ρ_{i-1}. By the definition of a constructive model, we then have $\rho_i(u_i(p_i)) = \rho_{i-1}(u_i(p_i))$, and, hence,

   $$\Phi_i(p_i) | u_i(p_i) : \bullet \, u_i(p_i)] \Rightarrow u_i(p_i) = \bullet \, u_i(p_i))$$

   holds trivially in ρ_i. From (1), and since $u_i(p_i')$ $(p_i' \neq p_i)$ occurs only under the scope of $\bullet$ in $\Phi_i(p_i)$, $\Phi(p_i)$ cannot be invalidated by changes to $u_i(p_i')$ $(p_i' \neq p_i)$, hence $\Phi_i(p_i)$ holds in ρ_i.

(b) In the second case, ρ_{i-1} does not satisfy $\Phi_i(p_i)$. Then, since $u_i(p_i')$ $(p_i' \neq p_i)$ occurs only under the scope of $\bullet$ in $\Phi_i(p_i)$, ρ_i does not satisfy $\Phi_i(p_i)[u_i(p_i) : \bullet\, u_i(p_i)]$, and, hence,

$$\Phi_i(p_i)[u_i(p_i) : \bullet\, u_i(p_i)] \Rightarrow u_i(p_i) = \bullet\, u_i(p_i)$$

holds in ρ_i. Moreover, $\Phi_i(p_i)$ holds in ρ_i by definition in a constructive model.

Hence, for any p_i,

$$\Phi_i(p_i) \wedge (\Phi_i(p_i)[u_i(p_i) : \bullet\, u_i(p_i)] \Rightarrow u_i(p_i) = \bullet\, u_i(p_i))$$

holds in ρ_i. By (2), and since $u_{i+1}, \ldots, u_k$ occur only under the scope of $\bullet$ in $\Phi_i(p_i)$, it follows that

$$\Sigma, m \models \forall p_i\ (\Phi_i(p_i) \wedge (\Phi_i(p_i)[u_i(p_i) : \bullet\, u_i(p_i)] \Rightarrow u_i(p_i) = \bullet\, u_i(p_i))).$$

Theorem

Each temporal logic model of the specification obtained from the specification S by the rewrite semantics is a constructive model of S.

Proof : Let $\Sigma = (\sigma_m : m \in N)$ be a temporal logic model for the specification obtained from S by the rewrite semantics.

- As we have '*initially* $\Rightarrow \Gamma \wedge \Psi \wedge \bigwedge_{i=1}^{k} \forall p_i\ \Phi_i(p_i)$', it is clear that

  $$\Sigma, 0 \models \Gamma \wedge \Psi \wedge \bigwedge_{i=1}^{k} \forall p_i\ \Phi_i(p_i).$$

- We prove that for each $m > 0$, σ_m can be obtained from σ_{m-1} as described in the constructive semantics.

  - We have to find a state ρ_0 which can be obtained from σ_{m-1} by changing only the values assigned to active components, and such that ρ_0 satisfies Ψ.

    Let ρ_0 be obtained from σ_{m-1} by setting x_i to $\sigma_m(x_i)$ for $1 \leq i \leq n$. Since $\Sigma, m \models \Psi$ and since $u_1, \ldots, u_k$ occur only under the scope of $\bullet$ in Ψ, it follows that ρ_0 satisfies Ψ.

  - We have to find a sequence of states $(\rho_1, \ldots, \rho_k)$ with the properties mentioned in the constructive semantics.

    Let $(\rho_1, \ldots, \rho_k)$ be the states such that ρ_i is obtained from ρ_{i-1} by setting the value of u_i to $\sigma_m(u_i)$. In other words, for $1 \leq i \leq k$,

    $$\begin{aligned} &\rho_i(u_g) = \sigma_m(u_g) \text{ for } 1 \leq g \leq i \\ &\rho_i(u_g) = \sigma_{m-1}(u_g) \text{ for } i+1 \leq g \leq k \end{aligned} \tag{3}$$

    Note that it follows that $\rho_k = \sigma_m$.

    We prove that ρ_i can be obtained from ρ_{i-1} as described in the constructive semantics. Let p_i be in the domain of u_i.

(a) Suppose $\Phi_i(p_i)$ is satisfied in ρ_{i-1}. Then, since $u_i(p_i')$ $(p_i' \neq p_i)$ occurs only under the scope of $\bullet$ in $\Phi_i(p_i)$, $\Sigma, m \models \Phi_i(p_i)[u_i(p_i) : \bullet\, u_i(p_i)]$ and hence, $\Sigma, m \models u_i(p_i) = \bullet\, u_i(p_i)$.
Consequently, $\sigma_m(u_i(p_i)) = \sigma_{m-1}(u_i(p_i))$ and, by (3), $\rho_i(u_i(p_i)) = \rho_{i-1}(u_i(p_i))$.

(b) Suppose $\Phi_i(p_i)$ is not satisfied in ρ_{i-1}. Since $\Sigma, m \models \Phi_i(p_i)$ and since $u_{i+1}, \ldots, u_k$ occur only under the scope of $\bullet$ in $\Phi(p_i)$, by (3), it follows that $\Phi_i(p_i)$ is satisfied in ρ_i.