*Fundamenta Informaticae 34 (2000) 109–124*
*IOS Press* 

# The AltaRica Formalism for Describing Concurrent Systems

**André Arnold, Gérald Point, Alain Griffault and Antoine Rauzy***
*LaBRI, Université Bordeaux I and CNRS (UMR 5800)*

**Abstract.** The AltaRica formalism is designed for describing complex systems consisting of a number of interacting components. Its semantics is expressed in terms of transition systems so that a system described in this language can be analysed by any technique or tool applicable to transition systems.

The components of a system have two kinds of interactions

- event synchronisation, like in the synchronized product of transition systems of Arnold and Nivat,
- interface coordination: with each component are associated interfaces whose values depend on the state of the component as well as on the values of interfaces of other components of the system.

Another feature of AltaRica is the possibility of defining hierarchical systems: some subsystems can be encapsulated and their mutual interactions as well as their interactions with the rest of the system are supervised by a controller.

**Keywords:** AltaRica, safety critical systems, transition systems, hierarchical systems, synchronization, interface coordination.

## 1. Introduction

In more and more industrial products (aircrafts, cars, nuclear plants, chemical plants, etc.), hardware and software computer systems play a more and more important role, especially for implementing control functions. These systems are safety-critical [Lev95] and consequently must

*Address for correspondence: LaBRI, Université Bordeaux I, 351 cours de la Libération, F-33405 Talence, `Andre.Arnold@labri.u-bordeaux.fr`

be carefully designed and exhaustively checked. Most of this analysis work, or inspection work, must be conducted on a mathematical model of the system under study [Par95].

A characteristic feature of these systems is that they consist of several kinds of components of various nature: physical components (mechanical, electrical, chemical) and software components, all interacting in several and various ways (synchronizations, exchanges of messages, sensors and controllers, etc.). Therefore a mathematical model of these systems must be able to describe the individual components as well as the ways they interact, and to define what the result of these interactions is.

After the seminal work of Petri [Pet62], several such models were proposed in the litterature in the seventies, ranging from programming languages like CSP [Hoa78] to algebraic models like CCS [Mil73, HM85]. Among them, COSY [LTS79, JL92] focuses on the synchronization mechanisms between agents and resources. The synchronization mechanism is also the main feature of the model introduced by Arnold and Nivat [AN82, Arn94], but this model does not distinguish between the natures of the components of the system.

Although the synchronized product of transition systems of Arnold and Nivat is a model for a very wide range of systems [ABC94], it is in some sense too basic or too abstract to be of practical use for engineers. In particular some basic kinds of physical interactions occuring in concrete systems cannot be simply expressed in this model.

Therefore to meet practical needs expressed by engineers involved in the development of critical systems in various industrial branches (avionics, ground transportations, energy) we have extended this basic model in two directions

- by generalizing the notion of a synchronization vector in order to express broadcast communications and by assigning priorities to these vectors,
- by adding constraints that express mutual dependencies on the states of the components [BR94, FV94], provided that there is a way for a component to get information about the state of another component.

Another feature that is really important from a practical point of view is the possibility to define hierarchical systems (i.e. systems built on from subsystems) so that a subsystem can be substituted for an equivalent one without changing the global system.

The AltaRica formalism was designed to express these basic constructs, whose semantics is precisely and unambiguously defined in terms of transition systems, on which it is possible to perform simulation as well as verification.

In the first part of this paper we introduce the AltaRica components in the form of interfaced transition systems. In the second part, we introduce the hierarchical components, called nodes, and we show that they still are interfaced transition systems. Then we introduce a notion of bisimulation that is consistent with the hierarchical constructs of AltaRica. The main features of the formalism will be illustrated by small examples that directly come from more realistic case studies.

# 2. AltaRica components

A component of a complex system has a finite number of *state variables*, say $s_1, s_2, \ldots, s_n$, each $s_i$ ranging over a domain $S_i$ and a finite number of *flow variables*, say $f_1, \ldots, f_m$, each one ranging over a domain $F_i$.

The difference of nature between state variables and flow variables is that the values of the first ones are purely local, as if they were inside a black box, and the environment (i.e., other components of the system) can never access directly to their values. On the contrary the second ones are precisely used to exchange information between the component and its environment.

Very often, the values of the flow and the state variables are not completely independent, and there are strong relationships between these values, so that some elements of the cartesian product $S_1 \times \cdots \times S_n \times F_1 \times \cdots \times F_m$ can never be configurations of the component. For example, a simple electrical switch has a state variable, ranging over a two-element domain, depending on whether the switch is open or closed, and two flow variables indicating the voltage at each of its two connectors. This example also shows that the values of the flow variables may depend on the values of the state variables: If the switch is connecting, the voltages at its two connectors are equal.

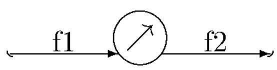



Figure 1: A switch

Therefore, the set of configuration of a component is a subset $C$ of the cartesian product $S_1 \times \cdots \times S_n \times F_1 \times \cdots \times F_m$. The projection of a configuation $c \in C$ is called the *observable part* of this configuration. We will see later the important role that these observable parts play when several components are interacting

An important characteristic feature of the AltaRica model is that the sets of state and flow values may be infinite, but are always discrete. With this respect, our components are not hybrid systems. Of course, it may happen that flows are used to represent continuous quantities like temperatures, pressures, voltages, but since the values of these quantities are accessed only by sensors, we assume that they are discretized or sampled. Returning to the example of the switch, we may consider that the voltage has only two discrete values: high and low. Therefore, the configuration of a component may change only by discrete steps, called *transitions*. These transitions are caused by *events* that are of two kinds.

- local events: they are events that the component knows about, either because they are local to the component, or they are not local, but the component is sensitive to their occurrences,
- an invisible event, denoted by $\epsilon$: the configuration of the component changes because of an external event not perceptible by the component.

For instance, if one pushes on the switch, its state changes, and possibly also the voltages. This is an event that the switch can perceive (it is even one of its roles to perceive this event!). On the other hand, if the switch is a part of an electrical circuit containing a generator, and if the generator fails, the configuration of the switch changes because its two voltages turn down to "low", but there is absolutely no reason for assuming that the switch has perceived the event "power failure": its two voltages have just gone to zero.

Note that whether an event can be perceived by a component or not is by no way a characteristc property of the event but rather one of the component, and indeed it is a part of the description of a component. For instance, let us assume that a component has a flow variable representing a pressure and let us consider the event "the pressure goes over a threshold". This event can or cannot be detected by the component (if it has or not a pressure sensor, for instance) and that can make a huge difference.

This leads us to consider, at an abstract level, that a component is a labelled transition system with some additionnal features

**Definition 2.1. (Interfaced transition systems)** An *interfaced transition system* is a 5-tuple $\mathcal{A} = \langle E, O, C, \pi, T \rangle$ where

- $E = E_+ \cup \{\epsilon\}$ is a set of events,
- $O$ is a set of *observations*,
- $C$ is a set of *configurations*,
- $\pi : C \to O$ is a mapping that with each configuration $c \in C$ associates its observable part $\pi(c) \in O$,
- $T \subseteq C \times E \times C$ is a set of *transitions*, that contains at least $\langle c, \epsilon, c \rangle$ for any $c \in C$.

**Example** The configuration of the switch of the previous example is a triple of Boolean values *on*, $f_1$, $f_2$ indicating whether the switch is connecting or not and whether the two voltages $f_1$ and $f_2$ are high or low. When *on* is true, $f_1$ and $f_2$ must be equal, and when it is false the four pairs of values for $f_1$ and $f_2$ are possible, hence the transition systems has six configurations: two where the switch is connecting and four where it is not. The values of the voltages can be modified (by $\epsilon$-transitions) without changing the state of the switch. On an occurence of the event *push*, the state of the switch changes and there is a transition labelled by this event from any of the two "connected" configurations to the four others, and vice-versa, as shown in the next figure where the observable part of a configuration is typeset in italic.

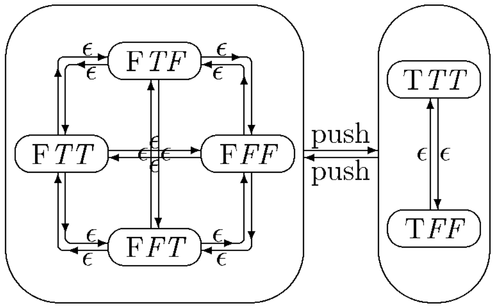



Figure 2: Switch (on,f1,f2)

Another feature we wish to introduce in our model is a notion of *event priority*. In some configurations, one may want to specify that an event is allowed to occur only if there are no other events with higher priority to deal with first. The interest of this notion of event priority will appear later on when we consider systems of interacting components.

**Definition 2.2. (Interfaced transition systems with priorities)** Let $E = E_+ \cup \{\epsilon\}$ be a set of events. A *priority relation* on $E$ is a strict partial ordering $<_E$ of $E$ such that $\forall a \in E_+$, $a \not<_E \epsilon$ and $\epsilon \not<_E a$. This last condition means that the invisible event cannot be related by any priority relation to any other event. This is natural since it is invisible!

An interfaced transition systems with priority is a tuple $\mathcal{B} = \langle E, <_E, O, C, \pi, T\rangle$ where $<_E$ is a priority ordering and $\langle E, O, C, \pi, T\rangle$ is an interfaced transition system.

By resolving priority constraints, one can transform an interfaced transition system with priority into a interfaced transition system without priority.

**Definition 2.3. (Resolution of priority constraints)** Let $\mathcal{B} = \langle E, <_E, O, C, \pi, T\rangle$ be an interfaced transition system with priority, and let $\mathcal{A}$ be the interfaced transition system $\langle E, O, C, \pi, T\rangle$. The interfaced transition system obtained from $\mathcal{B}$ by resolving the priority constraints, denoted by $\mathcal{A} \upharpoonright\!<_E$, is the interfaced transition system $\langle E, O, C, \pi, T \upharpoonright\!<_E\rangle$ where $T \upharpoonright\!<_E$ is the set of all transitions $\langle c, e, c'\rangle \in T$ that satisfy $\forall c'' \in C, \forall e' \in E, \quad \langle c, e', c''\rangle \in T \Rightarrow e \not<_E e'$.

To be sure that $\mathcal{A} \upharpoonright\!<_E$ is actually an interfaced transition system, we have only to check that for any $c \in C$, $\langle c, \epsilon, c\rangle \in T \upharpoonright\!<_E$. This follows from the fact that $\epsilon$ is incomparable (w.r.t. $<_E$ ) with any $e \in E_+$.

One can also remark that if $<_E$ is the empty ordering (i.e., there is no priority between the events), then the resolution of priority constraints is useless: $\mathcal{A} \upharpoonright\!<_E = \mathcal{A}$.

## 3. AltaRica nodes

The AltaRica language allows hierarchical descriptions of systems. Besides the basic components, that are interfaced transition systems, there are other components, called *AltaRica nodes*, consisting of a set of components (interfaced transition systems or other nodes) interacting under the supervision of a controller. We define below the "semantics" of a node (i.e., the way it reacts to some events and how it can potentially exchange information with other components) in terms of interfaced transition systems. It follows that if a component of a node is another node, we can substitute it by an interfaced transition system. Hence, we may assume, in the definition of a node, that all its components are actually interfaced transition systems.

**Definition 3.1. (Altarica nodes)** An *AltaRica node* is a tuple

$$\mathcal{N} = \langle E, <_E, O, \mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_n, V\rangle$$

where:

- $E$ is a set of events partially ordered by $<_E$.
- $O$ is a set of observations.
- For $i = 1 \ldots n$, $\mathcal{A}_i = \langle E_i, O_i, C_i, \pi_i, T_i\rangle$ is an interfaced transition system, called a *sub-node* as a reminder that it can be the interfaced transition system associated with a node.
- $\mathcal{A}_0 = \langle E_0, O_0, C_0, \pi_0, T_0\rangle$ is an interfaced transition system where $E_0 = E$ and $O_0 = O \times O_1 \times \cdots \times O_n$. It is called the *controller* of the node $\mathcal{N}$.
- $V \subseteq E_0^? \times E_1^? \times \ldots E_n^? \times 2^{[0,n+1]}$ is a set of *broadcast synchronization vectors.* For all $i$, $E_i^?$ is the set $E_i \cup \{e? \mid e \in E_i - \{\epsilon\}\}$. Furthermore, we assume that $V$ always contains the vector $\vec{\epsilon} = \langle \epsilon, \ldots, \epsilon, \{0\}\rangle$.

## 3.1. Broadcast synchronization vectors

In several models of communicating systems, an event occuring in one component must be synchronous with some other events occuring in some other components, for instance when one of them is just the instantaneous effect in one component of some event occuring in another one (e.g., push a switch and a bulb lights on), or when a "global" event is by essence a distributed event that occurs in the form of several synchronized "local" events (e.g., a handshake is when two persons mutually shake their hands).

Synchronization vectors were introduced in the Arnold-Nivat model to express all kinds of such synchronizations of events.

However, it turned out that in the systems we have to model, these synchronization vectors are not expressive enough. For instance, let us consider an electric circuit with several lamps. In case of an event "power failure", all the lamps must simultaneously be extinguished. But some of them are already off and cannot execute "light off". Of course, it is possible to model this circuit with synchronization vectors in such a way that in case of power failure only lighted lamps turn off, but at the cost of a considerable increase of the size of the model.

In the AltaRica formalism we allow the modeller to express something like "in case of power failure, all lighting lamps light off" by specifying that if a component is not able to react by the specified event to another event, it is not obliged to do so.

This is done by tagging this event by a question mark in the synchronization vector. For example, if we have a synchronization vector $\langle a, b, c?, d?\rangle$, the possible global events will be $\langle a, b, c, d\rangle$, $\langle a, b, \epsilon, d\rangle$, $\langle a, b, c, \epsilon\rangle$ , and $\langle a, b, \epsilon, \epsilon\rangle$. But, for instance, $\langle a, b, \epsilon, d\rangle$ is forbidden if the third component has the possibility to react by $c$. On the other hand, $\langle a, \epsilon, c, d\rangle$ is always forbidden.

Finally we introduce an additional constraint to such vectors that concerns the number of "reacting" components (i.e., those that actually take part in the interactions by not executing the invisible actions $\epsilon$). In some cases, the interaction expressed by the vector can take place only if there is a minimal number of "participants". For instance consider a lift and its event *request(i)* that memorizes that it has been requested to stop at floor $i$. This can be because this request is made by somebody inside the cabin who wish to get out at floor $i$ (event *out(i)*) or by somebody at floor $i$ who calls the lift (event *call(i)*). Then the vector ⟨*request(i), out(i)?, call(i)?*⟩ with the constraint $\{1, 2\}$ expresses that *request(i)* occurs only when at least one of the two requests *out(i)* or *call(i)* occurs simultaneously. In some other cases it could be a maximal number, and sometimes just a fixed number, like in the rendez-vous "two out of three" expressed by the vector $\langle rv?, rv?, rv?\rangle$ with the constraint denoted by $\{2\}$ to express that two and only two components have to react by $rv$ in this interaction.

To precisely define the set of possible interactions that are described by a broadcast synchronization vector, we introduce the notion of an instance of such a vector.

**Definition 3.2. (Instances of a broadcast synchronization vector)**
Let $v = \langle e'_0, e'_1, \ldots, e'_n, D\rangle \in E^?_0 \times E^?_1 \times \ldots E^?_n \times 2^{[0,n+1]}$. We say that $u = \langle e_0, e_1, \ldots, e_n\rangle \in E_0 \times E_1 \times \ldots E_n$ is an instance of $v$ if

- for any $i = 0, 1, \ldots, n$,
  - if $e'_i \in E_i \subseteq E^?_i$ then $e_i = e'_i$,
  - if $e'_i = b?$ with $b \in E_i - \{\epsilon\}$ then $e_i = b$ or $e_i = \epsilon$.
- the cardinal of the set $\{i \mid 0 \leq i \leq n, e_i \neq \epsilon\}$ is in $D$.

We denote by $I(v)$ the set of instances of $v$ and we order it by $\sqsubseteq$, the product of the orderings $\sqsubseteq_i$ of $E_i$ defined by $e \sqsubseteq e'$ if and only if $e = \epsilon$ or $e = e'$.

Note that $\vec{\epsilon} = \langle \epsilon, \epsilon, \ldots, \epsilon, \{0\}\rangle$ has one and only one instance which is $\langle \epsilon, \epsilon, \ldots, \epsilon\rangle$.

## 3.2. Semantics of a node

The semantics of a node $\mathcal{N}$ is the interfaced transition system associated with it, as defined below.

Let $\mathcal{N} = \langle E, <_E, O, \mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_n, V\rangle$ be a node where $\mathcal{A}_i = \langle E_i, O_i, C_i, \pi_i, T_i\rangle$, for $i = 0, 1, \ldots, n$, and where $E_0 = E$ and $O_0 = O \times O_1 \times \cdots \times O_n$.

We denote by $\phi$, $\phi_1, \ldots, \phi_n$ the projections of $O_0$ on $O$, $O_1, \ldots, O_n$.

The interfaced transition system $\mathcal{A} = \langle E, O, C, \pi, T\rangle$ associated with $\mathcal{N}$ is defined as follows.

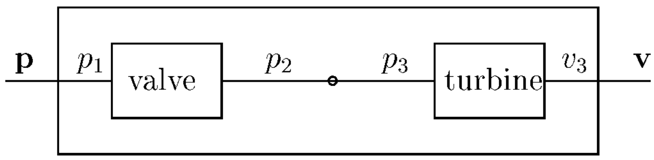
**Configurations** The set of configurations of $\mathcal{A}$ is

$$C = \{(c_0, c_1, \dots, c_n) \in C_0 \times C_1 \times \cdots \times C_n \mid \forall i = 1, \dots, n, \phi_i(\pi_0(c_0)) = \pi_i(c_i)\}.$$

If we interpret the observable part of a configuration as the vector of values of the flow variables, this definition implies that the value of any flow variable of the sub-node $\mathcal{A}_i$ is also the value of a flow variable of the controller. In other words, the controller has access to the flow variables of all the sub-nodes, and its set of configurations allows it to impose some relationship between its own configuration and the configurations of the sub-nodes. Thus, the configurations of the controller specifies a first kind of interaction between the components of a node, that we call *coordination of interfaces.*

**Example** Consider a turbine fed by a highpressure intake line controlled by a valve. The turbine has two states: *run* and *stop*, and two flow variables, $p_3$ equal to 0 or 1, representing the intake water pressure and $v_3$ equal to 0 or 1, representing the current produced by the turbine. The turbine produces current if and only if it is running and is under pressure. Its configurations are thus *(run, p=1, v=1)* and *(stop, p=0, v=0)*. The valve also has two states, *open* and *closed*, and two flow variables $p_1$ and $p_2$ equal to 0 or 1, representing their incoming and outgoing pressures of the valve. If the valve is open then $p_1 = p_2$ and if the valve is closed then $p_2 = 0$. Its configurations are thus *(open, $p_1$=1, $p_2$=1)*, *(closed, $p_1$=1, $p_2$=0)*, *(open, $p_1$=0, $p_2$=0)*, *(closed, $p_1$=0, $p_2$=0)*.

The intake line with its valve and its turbine is described by a node whose sub-nodes are the valve and the turbine and whose controller describes the interactions between the valve and the turbine. It has no state variables. Its flow variables are those of the sub-nodes: $p_1$, $p_2$, $p_3$, $v_3$ and those of the node itself: a variable $p$ representing the pressure upstream of the line, and a variable $v$ representing the current obtained on output. The configurations of the controller are the assignments to these variables that satisfy the constraints $p = p_1$, $p_2 = p_3$, $v_3 = v$ explained by the drawing below.




The configurations of the node are thus the following.

| $p = p_1$ | valve | $p_2 = p_3$ | turbine | $v_3 = v$ |
|---|---|---|---|---|
| 1 | open | 1 | run | 1 |
| 1 | closed | 0 | stop | 0 |
| 0 | open | 0 | stop | 0 |
| 0 | closed | 0 | stop | 0 |

**Observations** The mapping $\pi : C \to O$ is defined by $\pi(c_0, c_1, \dots, c_n) = \phi(\pi_0(c_0))$. In other words, the observable part of the configuration of the node is the restriction to $O$ of the observable part of the configuration of the controller.

**Transitions** We have seen above that the controller has access to the flow variables of the sub-nodes. It follows that if the configuration of a sub-node is modified by a transition, it may happen that the configuration of the controller is simultaneously modified and thus the controller has also simultaneously executed a transition. Conversely, if the configuration of the controller is modified by a transition, then the configurations of some of the sub-nodes may have changed. Therefore a transition of the node consists of a vector of simultaneous transitions of the controller and all the sub-nodes (possibly some of them are the special transition $(c, \epsilon, c)$.) However, due to the constraints between flow values expressed by the set of configurations of the controller, not every such vector is firable. Moreover the set $T$ of executable "global" transitions is further restricted by synchronization constraints expressed by the synchronization vectors.

This set $T$ of transitions is defined in four steps.

First step. Let $W = \bigcup_{v \in V} I(v) \times \{v\}$. Let $T_{\mathcal{C}} \subseteq C \times W \times C$ be defined by

$$\langle (c_0, c_1, \ldots, c_n), (e_0, e_1, \ldots, e_n, v), (c'_0, c'_1, \ldots, c'_n) \rangle \in T_{\mathcal{C}}$$

if and only if for any $i$ between 0 and $n$, $\langle c_i, e_i, c'_i \rangle \in T_i$. Clearly, $\langle c, (\epsilon, \ldots, \epsilon, \vec{\epsilon}), c \rangle \in T_{\mathcal{C}}$, for any $c \in C$.

Second step. We equip $W$ with the strict partial order $<_W$ defined by $(u, v) <_W (u', v')$ if and only if $v = v'$ et $u \sqsubset u'$. Then we set $T_{\mathcal{B}} = T_{\mathcal{C}} \lceil <_W$. This step means that when several instance of the same vector are firable, we choose one where a maximal number of components react. We still have for any $c \in C$, $\langle c, (\epsilon, \ldots, \epsilon, \vec{\epsilon}), c \rangle \in T_{\mathcal{B}}$.

Third step. Let $T_{\mathcal{A}} \subseteq C \times E \times C$ such that $\langle c, e, c' \rangle \in T_{\mathcal{A}}$ if and only if there is $w = (e_0, e_1, \ldots, v) \in W$ such that $e = e_0$ et $\langle c, w, c' \rangle \in T_{\mathcal{B}}$. Still, for any $c \in C$, $\langle c, \epsilon, c \rangle \in T_{\mathcal{A}}$.

The last step is to take $T = T_{\mathcal{A}} \lceil <_E$. This step amounts to the following. When two global transitions are firable in a configuration of a node, the controller has the possibility of selecting one of them according to the priority relation on $E$, of course to the extent that it is involved in these transitions by one of its events.

**Example** Consider a spigot that ought to be closed to prevent the bathtub from overflowing. We suppose therefore that the spigot is in the state *open* and passes to the state *closed* using a transition caused by the event *close-me*. The event "the controller closes the spigot" is noted $f_1$, and will be synchronized with the event *close-me* by means of the synchronization vector $\langle f_1, \textit{close-me} \rangle$. However it might be that the bathtub spigot is stuck in a state *blocked-open* and it is impossible to close it. To avoid an overflow, the controller has another solution: close the incoming feed spigot at the water meter. This event, noted $f_2$, has no effect on the bathtub spigot, giving the synchronization vector $\langle f_2, \epsilon \rangle$.

After the three first transformations shown above, we obtain the following transitions, ignoring those configurations of the controller that are unnecessary to describe here.

- (*open*, $f_1$, *closed*),
- (*open*, $f_2$, *open*),

- (*blocked-open*, $f_2$, *blocked-open*).

In the last two cases the bathtub does not overflow, even if the spigot remains open, since the feed was closed. Nonetheless we can consider that if the bathtub spigot works, it suffices to close it and it is not necessary to close the feed. To express this additional condition, it suffices to give $f_1$ a higher priority than $f_2$. In this case, after the final step of the definition of $T$, there remain only two transitions

- (*open*, $f_1$, *closed*),
- (*blocked-open*, $f_2$, *blocked-open*).

## 3.3. Controller

As we have seen above, the component $\mathcal{A}_0$ of a node controls and coordinates its sub-nodes in three different ways:

- by sharing with each sub-node $\mathcal{A}_i$ the set $O_i$ of observations,
- by interacting synchronously with the sub-nodes as described by the broadcast synchronization vectors,
- by assigning priorities to the events it synchronizes.

This gives the controller a very important role in a node. Some examples were already given. Here we give another example that shows how a controller may reconfigure a system on occurences of failures. It is a typical example of the systems that we have to model, and for which the AltaRica formalism was specially designed.

Let us consider a supervisor $S$ that reads its input flow value $v$ and reports a flow value $w$ that represents a status of the input value, so that $v$ and $w$ are related by a functional relation $w = f(v)$. This supervisor is a complex mechanism that can become faulty by the failure of one of its components and in this case its status value is no longer related to the input value. Therefore, this supervisor has two states $OK$ and $notOK$ and passes from the first to the second one (whatever the flow values are) by a transition caused by the event *failure*. Its configurations are defined by the following constraint: If the state is $OK$ then $w = f(v)$.

As usual in this kind of situation the system contains a second supervisor $S'$ that can be used instead of $S$ in case of failure of $S$. Switching from $S$ to $S'$ is performed by a controller that can perceive the failure of $S$ by an event *S_fails* that is synchronized with the failure of $S$ by the synchronization vector $\langle S\_fails, failure, \epsilon\rangle$. This controller has two states $S\_OK$ and $S\_notOK$ and the event *S_fails* makes it go from the first state to the second one, whatever the flow values are. Its flow variables are $V$ representing the input value to be supervised, $W$ representing the status of the input value, and the flow variables *S.v*, *S.w*, *S'.v*, *S'.w* of the two supervisors $S$ and $S'$.

The configurations of the controller are defined by the following constraints

- $V = S.v = S'.v$ that means that the two supervisors read the same input value.
- If the state of the controller is $S\_OK$ then $W = S.w$. If it is $S\_notOK$ then $W = S'.w$.

It follows that if $S$ is OK, $W = f(V)$. If $S$ is not OK and if $S'$ is OK, still $W = f(V)$. Obviously, there is a problem if the two supervisors are faulty! However if the controller is able to perceive the failure of the second supervisor, it can know about such a situation and be informed that the value of $W$ is then meaningless.

## 4. Bisimulations

In the definition of an AltaRica node, we have emphasized the fact that the states of a sub-node are anonymous: the only information a node has about its sub-nodes is the value of the flows. In this section we formalize this point by defining a notion of bisimulation consistent with the hierarchical construction of nodes: if we substitute for a component of a node (the controller or a sub-node) an equivalent component (with respect to this bisimulation), then the two nodes are still equivalent.

In an interfaced transition system $\mathcal{A} = \langle E, O, C, \pi, T\rangle$, the observation $\pi(c) \in O$ assigned to the configuration $c$ can be seen as a property of this configuration. An *interfaced bisimulation* will be a bisimulation that preserves these properties. Indeed, as explained in [Arn94], it is sufficient to define bisimulation relations that are also transition system homomorphisms.

**Definition 4.1. (Interfaced bisimulation homomorphisms)** Let $\mathcal{A} = \langle E, O, C, \pi, T\rangle$ and $\mathcal{A}' = \langle E, O, C', \pi', T'\rangle$ be two interfaced transition systems having the same sets of events and observations. An interfaced bisimulation homomorphism $h : \mathcal{A} \to \mathcal{A}'$ is a mapping $h : C \to C'$ that satisfies:

**(b0)** $h$ is surjective,
**(b1)** for any $c \in C$, $\pi(c) = \pi'(h(c))$,
**(b2)** for any $\langle c_1, e, c_2\rangle \in T$, $\langle h(c_1), e, h(c_2)\rangle \in T'$,
**(b4)** for any $c_1 \in C, c_2' \in C'$, if $\langle h(c_1), e, c_2'\rangle \in T'$ then there is $c_2 \in C$ such that $h(c_2) = c_2'$ and $\langle c_1, e, c_2\rangle \in T$.

**Proposition 4.1.** *If $h : \mathcal{A} \to \mathcal{A}'$ is an interfaced bisimulation homomorphism from $\mathcal{A}$ to $\mathcal{A}'$ it is also an interfaced bisimulation homomorphism from $\mathcal{A} \upharpoonright <_E$ to $\mathcal{A}' \upharpoonright <_E$ .*

**Proof** Let $\mathcal{A} = \langle E, O, C, \pi, T\rangle$ and $\mathcal{A}' = \langle E, O, C', \pi', T'\rangle$. We have to show

1. $\langle c_1, e, c_2\rangle \in T \upharpoonright <_E \Rightarrow \langle h(c_1), e, h(c_2)\rangle \in T' \upharpoonright <_E$,
2. if $\langle h(c_1), e, c_2'\rangle \in T' \upharpoonright <_E$ then there is $c_2 \in C$ such that $h(c_2) = c_2'$ et $\langle c_1, e, c_2\rangle \in T \upharpoonright <_E$.

1) Let $\langle c_1, e, c_2\rangle \in T \upharpoonright <_E$. Then $\langle c_1, e, c_2\rangle \in T$ hence $\langle h(c_1), e, h(c_2)\rangle \in T'$. Let us assume that $\langle h(c_1), e, h(c_2)\rangle \notin T' \upharpoonright <_E$. By definition, there exists $e' >_E e$ and $c' \in C'$ such that $\langle h(c_1), e, c'\rangle \in T'$. But, by (b4), there is $c \in C$ such that $h(c) = c'$ and $\langle c_1, e', c\rangle \in T$. It follows that $\langle c_1, e, c_2\rangle \notin T \upharpoonright <_E$, a contradiction.

2) Let $\langle h(c_1), e, c_2'\rangle \in T' \upharpoonright <_E$. Then $\langle h(c_1), e, c_2'\rangle \in T'$ and there is $c_2 \in h^{-1}(c_2')$ such that $\langle c_1, e, c_2\rangle \in T$. If $\langle c_1, e, c_2\rangle \notin T \upharpoonright <_E$, there exists $e' >_E e$ and $c \in C$ with $\langle c_1, e', c\rangle \in T$. But then, $\langle h(c_1), e', h(c)\rangle \in T'$,that implies $\langle h(c_1), e', h(c)\rangle \notin T' \upharpoonright <_E$, a contradiction. □

If in a node, we replace the controller $\mathcal{A}_0$, whose semantics is $\mathcal{A}_0$, by a component $\mathcal{N}'_0$ whose semantics $\mathcal{A}'_0$ is the image of $\mathcal{A}_0$ under an interfaced bisimulation homomorphism, the semantics of the node is unchanged up to bisimulation. The same result holds for the sub-nodes.

**Proposition 4.2.** *For $i = 0, 1, \ldots, n$, let $h_i : \mathcal{A}_i = \langle E_i, O_i, C_i, \pi_i, T_i\rangle \to \mathcal{A}'_i = \langle E_i, O_i, C'_i, \pi'_i, T'_i\rangle$ be an interfaced bisimulation homomorphism. Let $\phi_i$ be the projection from $O_0$ on $O_i$. Let $\mathcal{A} = \langle E, O, C, \pi, T\rangle$ be the controlled product $\langle E, <_E, O; \mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_n, V\rangle$ and $\mathcal{A}' = \langle E, O, C', \pi', T'\rangle$ be the controlled product $\langle E, <_E, O; \mathcal{A}'_0, \mathcal{A}'_1, \ldots, \mathcal{A}'_n, V\rangle$. We have $\pi = \phi \circ \pi_0$ and $\pi' = \phi \circ \pi'_0$ where $\phi$ is the projection of $O_0$ on $O$. Let $h = h_0 \times h_1 \times \cdots \times h_n : C \to C'$. Then $h : \mathcal{A} \to \mathcal{A}'$ is an interfaced bisimulation homomorphism.*

**Proof** First of all, let us remark that $h : C \to C'$ is surjective. If $(c'_0, c'_1, \ldots, c'_n) \in C'$, then $\phi_i(\pi'_0(c'_0)) = \pi'_i(c'_i)$. Since the $h_i$ are bisimulations, there exist $c_i$ such that $h_i(c_i) = c'_i$ and $\pi_i(c_i) = \pi'_i(c'_i)$. Thus, $\phi_i(\pi_0(c_0)) = \pi_i(c_i)$ and $(c_0, c_1, \ldots, c_n) \in C$.

It is obvious that $\pi(\langle c_0, c_1, \ldots, c_n\rangle) = \phi(\pi_0(c_0)) = \phi(\pi'_0(h_0(c_0))) = \pi'(h((\langle c_0, c_1, \ldots, c_n\rangle)))$

The proof that $h : \mathcal{A} \to \mathcal{A}'$ satisfies (b3) and (b4) follows the four steps of the construction of $T$ and $T'$.

It is easy to see that $h : \langle E, O, C, \pi, T_{\mathcal{C}}\rangle \to \langle E, O, C', \pi', T'_{\mathcal{C}}\rangle$ satisfies (b3) and (b4). By Proposition 4.1, these properties are satisfied also by $h : \langle E, O, C, \pi, T_{\mathcal{B}}\rangle \to \langle E, O, C', \pi', T'_{\mathcal{B}}\rangle$. For the third step, it is enough to notice: 1) $\langle c, e_0, c'\rangle \in T_{\mathcal{A}} \Rightarrow \langle c, (e_0, e_1, \ldots, e_n, v), c'\rangle \in T_{\mathcal{B}} \Rightarrow \langle h(c), (e_0, e_1, \ldots, e_n, v), h(c')\rangle \in T'_{\mathcal{B}} \Rightarrow \langle h(c), e_0, h(c')\rangle \in T'_{\mathcal{A}}$.
2) $\langle h(c), e_0, c''\rangle \in T'_{\mathcal{A}} \Rightarrow \langle h(c), (e_0, e_1, \ldots, e_n, v), c''\rangle \in T'_{\mathcal{B}}$ and there is $c'$ such that $h(c') = c''$ and $\langle c, (e_0, e_1, \ldots, e_n, v), c'\rangle \in T_{\mathcal{B}}$, hence $\langle c, e_0, c'\rangle \in T_{\mathcal{A}}$. For the last step, we again use Proposition 4.1. $\square$

**Example** A two-way switch has two states: *up* and *down*, and three voltages: $f_1$, $f_2$, and $f_3$. When the switch is *up* we must have $f_1 = f_3$, and $f_1 = f_2$ when it is *down*.

The flow variables can be modified by $\epsilon$-transitions, without changing the state. The event *push* causes a transition from a configuration where the state is *up* to a configuration where the state is *down* and vice-versa (see Figure 4).

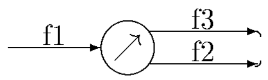



Figure 3: a two-way switch

Now we define a system consisting of two two-way switches $S1$ and $S2$ whose flow variables are renamed as indicated below and connected together as explained in the following figure.

This connection is expressed by a controller with one state and 8 flow variables that has $2^4 = 16$ configurations.

Moreover the event *push* of the controller is synchronized with a *push* in one and only one of the two switches by the vector $\langle$ *push, push?, push?,* $\{2\}\rangle$.

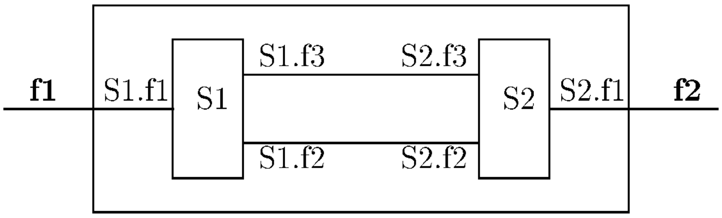



Figure 4. Two two-way switches

The configurations of the node are the following.

| S1 | S2 | f1=S1.f1 | S1.f2=S2.f2 | S1.f3=S2.f3 | S2.f1=f2 |
|---|---|---|---|---|---|
| down | down | $T$ | T | T | $T$ |
| down | down | $T$ | T | F | $T$ |
| down | down | $F$ | F | T | $F$ |
| down | down | $F$ | F | F | $F$ |
| up | down | $T$ | T | T | $T$ |
| up | down | $T$ | F | T | $F$ |
| up | down | $F$ | T | F | $T$ |
| up | down | $F$ | F | F | $F$ |
| down | up | $T$ | T | T | $T$ |
| down | up | $T$ | T | F | $F$ |
| down | up | $F$ | F | T | $T$ |
| down | up | $F$ | F | F | $F$ |
| up | up | $T$ | T | T | $T$ |
| up | up | $T$ | F | T | $T$ |
| up | up | $F$ | T | F | $F$ |
| up | up | $F$ | F | F | $F$ |

Its transitions are given by the following diagram.

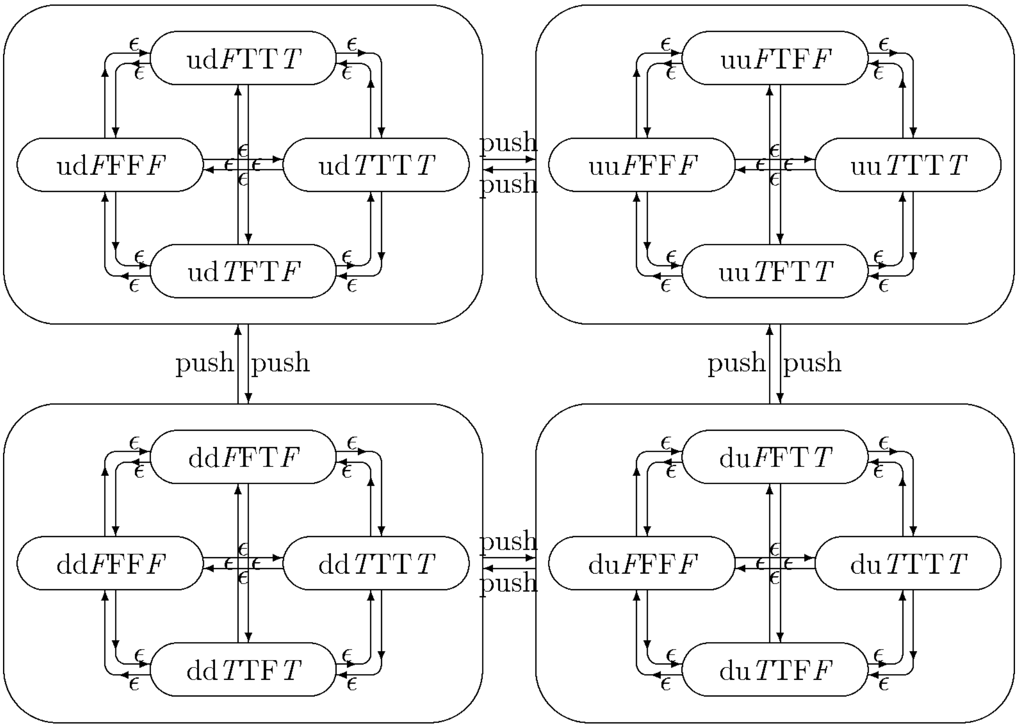



Figure 5: SwitchSystem

It is easy to see that the system of Figure 2 is the homomorphic image of the above transition system under the mapping $h$ defined by the following table, and that this mapping is an interfaced bisimulation homomorphism.

| | |
|---|---|
| dd*T*TT*T*<br>dd*T*TF*T*<br>uu*T*TT*T*<br>uu*T*FT*T* | T*TT* |
| uu*F*TF*F*<br>uu*F*FF*F*<br>dd*F*FT*F*<br>dd*F*FF*F* | T*FF* |
| ud*T*TT*T*<br>du*T*TT*T* | F*TT* |
| ud*F*FF*F*<br>du*F*FF*F* | F*FF* |
| ud*T*FT*F*<br>du*T*TF*F* | F*TF* |
| ud*F*TF*T*<br>du*F*FT*T* | F*FT* |

## 5. Conclusion

Of course, the description of a system with the AltaRica formalism is done by using a concrete syntax [GLP+98]. Actually, this concrete syntax imposes some minor restrictions on the AltaRica components and nodes that can be written with it.

However, one of these restrictions, that concerns the way the transitions are described, is very important from a pragmatic viewpoint. We have not still mentioned it, not to make heavy the above presentation, but all the given examples satisfy this restriction.

Indeed a configuration $C$ is a set of assigments of values to state and flow variables. If we denote by $S$ the set of assigments to state variables and by $F$ the set of assigments to flow variables, $C$ is a subset of $S \times F$. The observable part of a configuration $c$ is its projection $\pi_F(c)$ on $F$. We call its projection $\pi_S(c)$ on $S$ its state part.

The additional constraint is that a transition directly modifies only the state part of a configuration. If the observable part of the configuration is also modified by the transition, it is because the constraints that define the set of configurations of the component may oblige a change of the observable part when the state part changes.

More formally, the set $T$ of transitions of a component whose the set of configurations is $C$ satisfies the following property.

If $c'$ and $c''$ are two configurations of $C$ with the same state part (i.e., $\pi_S(c') = \pi_S(c'')$ )then

- $(c', \epsilon, c'')$ and $(c'', \epsilon, c')$ are in $T$,
- if for some $c \in C$, $(c, a, c')$ is in $T$, then $(c, a, c'')$ is also in $T$.

Indeed this restriction is not a real one: it is always possible to duplicate some flow variables by state variables. Let $C \subseteq S \times F \times F'$. We define $C' \subseteq S \times S' \times F \times F'$, with $S' = F'$ by $(s, s', f, f') \in C'$ if and only if $s' = f'$ and $(s, f, f') \in C$. Therefore, if $c'$ and $c''$ are two configurations of $C'$ that have the same state part (i.e., $\pi_S(c') = \pi_S(c'')$ and $\pi_{S'}(c') = \pi_{S'}(c'')$, we also have $\pi_{F'}(c') = \pi_{F'}(c'')$.

However, it turned out that a clear distinction between the roles of state and flow variables is a valuable guideline when one has to formally describe a complex system.

This concrete syntax is the basis for several tools for handling descriptions in this formalism (simulation, verification, risk analysis) that are currently under development. It is clear that a single formalism without any tool to exploit it is of low interest.

## References


[ABC94] A. Arnold, D. Bégay, and P. Crubillé. *Construction and analysis of transition systems with MEC.* World Scientific, 1994.

[AN82] A. Arnold and M. Nivat. Comportements de processus. In *Les mathématiques de l'Informatique*, pages 257–266. Colloque AFCET, 1982.

[Arn94] A. Arnold. *Finite transition systems.* Prentice Hall, 1994.

[BR94] S. Brlek and A. Rauzy. Synchronization of constrained transition systems. In H. Hong, editor, *Proc, First Int. Symp. on Parallel Symbolic Computation*. World Scientific, 1994.

[FV94] L. Fribourg and M. Veloso Peixoto. Automates concurrents à contraintes. *Technique et Science Informatiques*, 13:837–866,1994.

[GLP+98] A. Griffault, S. Lajeunesse, G. Point, A. Rauzy, J.-P. Signoret, and P. Thomas. The AltaRica language. In *Proc. European Safety and Reliability Assoc. Conf. (ESREL'98)*, 1998.

[HM85] M. Hennessy and R. Milner. Algebraic laws for non-determinism and concurrency. *Journal of the ACM*, 32:137–161, 1985.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Comm. of the ACM*, 21:666–677, 1978.

[JL92] R. Janicki and P. E. Lauer. *Specification and analysis of concurrent systems: The COSY approach.* EATCS Monographs on Theoretical Computer Science. Springer–Verlag, 1992.

[Lev95] N. G. Leveson. *Safeware: System safety and computers.* Addison–Wesley, 1995.

[LTS79] P. E. Lauer, P. R. Torrigiani, and M. W. Shields. Cosy, a system specification language based on paths and processes. *Acta Informatica*, 2:109–158, 1979.

[Mil73] R. Milner. Processes: A mathematical model of computing agents. In H. Rose and J. C. Shepherdson, editors , *Proc. Logic Colloquium*, pages 157–173, 1973.

[Par95] D. L. Parnas. Using mathematical models in the inspection of critical software. In M. G. Hinchey and J. P. Bowen, editors, *Applications of formal methods*, pages 17–31. Prentice Hall, 1995.

[Pet62] C. A. Petri. Fundamentals of a theory of asynchronous information flow. In *Proc. IFIP Congress*, 1962.