# Strategies for Modelling Failure Propagation in Dynamic Systems with AltaRica

Tatiana Prosvirnova, Christel Seguin, Christophe Frazza, Michel Batteux, Xavier de Bossoreille, Frédéric Deschamps, Jean Gauthier, Estelle Saez

# Strategies for modelling failure propagation in dynamic systems with AltaRica *

Tatiana Prosvirnova[1], Christel Seguin[1], Christophe Frazza[2], Michel Batteux[3], Xavier de Bossoreille[4], Frédéric Deschamps[5], Jean Gauthier[6], and Estelle Saez[7]

[1] ONERA/DTIS, Université de Toulouse, Toulouse, France, {Tatiana.Prosvirnova,Christel.Seguin}@onera.fr
[2] SaToDev, 25 rue Marcel Issartier, 33700 Mérignac, France, christophe.frazza@satodev.fr
[3] IRT SystemX, Palaiseau, France, michel.batteux@irt-systemx.fr
[4] APSYS Airbus, 37 Avenue Escadrille Normandie Niemen, 31700 Blagnac, France, xavier.debossoreille@apsys-airbus.com
[5] LGM, Euclide B4 - ZAC St Martin du Touch, 1 Rue Emmanuel Arin, 31300 Toulouse, France, frederic.deschamps@lgm.fr
[6] 5 Dassault Aviation, 54 AV Marcel Dassault, 33700 Mérignac, France, Jean.Gauthier@dassault-aviation.com
[7] IRT Saint-Exupéry, B612, 3 rue Tarfaya, 31405 Toulouse, France, estelle.saez@irt-saintexupery.com

**Abstract.** The AltaRica modelling language has been designed to facilitate failure propagation modelling and safety analyses of complex technical systems. Indeed, it makes it possible to model the functional dynamics (change of control mode, reconfiguration of equipment, etc.) and failures (cascades of failures, hidden failures, etc.) of the systems.
The objective of this article is to provide guides to make the best use of this dynamic modelling capability. We focus on the modelling of potentially problematic dynamic phenomenon - continuous control of a physical process with a feedback loop.
We propose different strategies to model this phenomenon illustrated by a simple example. We discuss the advantages and drawbacks of the proposed solutions.

**Keywords:** Model Based Safety Assessment · dynamic systems · failure propagation models · AltaRica.

## 1 Introduction

The AltaRica language [2] was designed to facilitate failure propagation modelling and safety analyses of complex technical systems. Thus, it makes it possible more particularly to model the functional dynamics (change of control mode, reconfiguration of equipment, etc.) and failures (cascades of failures, hidden failures, etc.) of the systems.

* Supported by French Institutes of Technology Saint Exupéry and SystemX.

The objective of this article is to provide guides to make the best use of this dynamic modelling capability for control systems. The proposed guides were developed by a panel of classical safety and MBSA (Model Based Safety Assessment) experts as part of the S2C (System & Safety Continuity) project of the French Institutes of Technology (FIT) Saint Exupéry and SystemX.

The S2C working group extracted from the participants' MBSA feedback the commonly encountered pitfalls and the modelling strategies adopted to overcome the difficulties. The paper focuses on potentially problematic dynamic phenomenon: continuous control of a physical process with feedback loops.

The modelling guide endeavours to give for this case:

- an example of a very simple system and a failure condition which illustrates the need for modelling;
- the usual modelling errors of this type of system and the resulting malfunctions;
- good modelling practices in the form of AltaRica modelling strategies;
- the results of qualitative analyses (simulations, search for the causes of failure conditions) which provide confidence in the behaviour, modelled according to the strategies;
- the general assumptions under which the modelling strategies are valid.

The modelling of dynamic systems in dependability has mainly been studied to evaluate the usual probabilistic indicators of reliability or safety (see for example [18]). This paper characterises strategies for modelling dynamic systems that also allow the calculation of sequences of events that lead to failure conditions. This type of modelling is in the process of being standardised in aeronautics and the communication also contributes to clarifying the modelling choices made to deal with the aircraft braking system in the future document ED-135 [1].

The remainder of this article is organised as follows. Section 2 describes the case study. Section 3 gives an overview of the related works. Section 4 presents issues raised by failure propagation modelling of systems with control loops and discusses different strategies to solve them. Section 5 concludes this article and gives some perspectives.

## 2 Case study description

In order to illustrate how to deal with failure propagation modelling of systems with control feedback loops, let us consider a simple example illustrated in Fig. 1. In that example, we consider a system composed of an equipment under control and a controller that builds a command from the information provided by a sensor and from the initial order sent, for instance, by an operator. The sensor acquires data of the equipment output and sends it to the controller, which is used to control the equipment. This example is a simplified control loop, and we can easily replace the equipment by a valve or an actuator.

We consider that all the components (operator order, controller, equipment under control and sensor) have two failure modes:

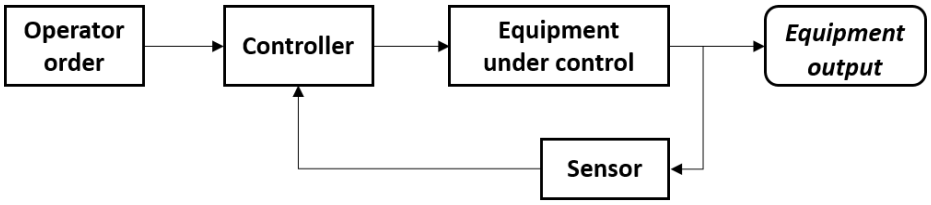


**Fig. 1.** Case study: an equipment under control.

– fail_loss: leads to the loss of the component;
– fail_err: leads to the erroneous behaviour of the component.

From a safety point of view, the evaluated failure conditions are the following:

– FC1: Loss of equipment output;
– FC2: Erroneous equipment output.

The equipment output is monitored by the sensor that sends its acquired information to the controller. The equipment output depends on the equipment input data. The controller computes a re-evaluated order from its two inputs (the operator order and the sensor acquisition information) and controls the equipment based on the order.

The component failures and the corresponding system output and effects are described in table 1.

**Table 1.** Component failure and their safety effects.

| **Component** | **Failure mode** | **Safety effects** |
|---|---|---|
| Order | fail_loss | Leads to the loss of control and the loss of equipment output. |
| | fail_err | Leads to an erroneous command of the equipment and an erroneous equipment output. |
| Equipment | fail_loss | Leads to the loss of equipment output. |
| | fail_err | Leads to an erroneous equipment output. The erroneous data is acquired by the sensor. |
| Sensor | fail_loss | Leads to the loss of the sensor acquisition sent to the controller leading to the loss of the equipment output. |
| | fail_err | Leads to an erroneous information from the sensor acquisition, leading to an erroneous equipment output. |

Equipment Output, on Fig. 1, is a safety artefact, an observer of the failure conditions.

To compute the order and to control the equipment, the controller needs the output of the equipment under control sent by the sensor.

Even before starting the modelling, one can identify that the modelled system is a control loop: the input of the controller depends on the sensor output depending itself on the controller output.

Modelling systems with a control loop using classical Fault Tree approach can lead to circular equations in a fault tree. If the fault tree is structured strictly following the dependencies of the different inputs and outputs of the system, there will be a circular logic in the produced fault tree.

In practice, most of the time, the circular equations in fault trees are solved by the analysts, who make assumptions on the behaviour of the system and adapt a modelling strategy to remove the circular equations from the fault tree. Nevertheless, when a circular equation appears in a fault tree, it is always worth analysing the possible impacts of the simplification performed to solve it.

Modelling systems with a control loop using high level modelling languages supporting MBSA (Model-Based Safety Assessment) approach may lead to similar problems. Different strategies to solve them are discussed in the following sections.

## 3 Related works

### 3.1 Static and dynamic failure propagation models

Failure propagation modelling formalisms can be divided in two categories: combinatorial models (for example, Fault Trees or Reliability Block Diagrams) and discrete-event models (for example, Markov chains or Generalized Stochastic Petri Nets).

Combinatorial models are static models, i.e. all the events are assumed to be independent and may occur in any order. In other words, the order of occurrence of events has no influence on the occurrence of the Failure Conditions. In this case, models are assessed by solving systems of Boolean equations to calculate Minimal Cut Sets (MCS) and probabilistic indicators (for instance, the probability of the Failure Condition). Efficient assessment algorithms have been developed for static models [16], which enable to assess industrial scale models.

Discrete-event models may be static, and in that case the occurrence order of the events has no influence on the resulting state. But it is not always the case. We say that a discrete-event model is dynamic if it exists at least one couple of sequences that are constituted with the same events and result in different states. For dynamic models, the order of occurrence of events is important.

Static discrete-event models can be assessed by generation and solving of Boolean equations. For dynamic discrete-event models, the compilation into Boolean equations is not always possible and may lose information. In that cases, it is possible to generate sequences of events leading to the Failure Conditions. The generation of sequences partially explores the failure scenarios of the model. Note that, the computation time greatly increases compared to the generation of Boolean equations and their assessment.

There are different high-level modelling languages supporting the MBSA approach. Amongst them we can cite AltaRica (AltaRica LaBRI [2], AltaRica DataFlow [7] and AltaRica 3.0 [4]), Figaro [8], SAML [9], HiP-HOPS [12], Component Fault Trees [11], Generalised Stochastic Petri nets with predicates implemented in GRIF [17]. The list is not exhaustive.

Amongst the cited modelling formalisms HiP-HOPS and Components Fault Trees are combinatorial formalisms and enable to create static models. To model failure propagation of systems with control loops using these formalisms the analyst needs to make assumptions on the system behaviour in order to create a static model.

AltaRica, Figaro, SAML and Petri nets are based on discrete-event models and enable to describe static and dynamic models. Continuous control of a physical process with a feedback loop is a dynamic phenomenon. Different strategies can be adopted for failure propagation modelling of systems with control loops. Some of them are presented and discussed using AltaRica DataFlow in the remainder of this article.

### 3.2 AltaRica modelling language

AltaRica is a high level textual formal domain specific modelling language dedicated to Safety Analysis created at the end of nineties [2]. AltaRica is an event-centric language. The behaviour of components is described by means of state machines. The state of a component is represented by variables (the so-called state variables) and their values. The changes of state are possible when, and only when, events occur. The occurrence of an event updates the values of the variables, by the firing of a transition: a triple $< guard, event, action >$, where a guard is a Boolean expression built over the variables and an action is an instruction which modifies the values of state variables. AltaRica distinguishes two types of variables: state variables and flow variables. State variables can be modified only through the firing of transitions. Flow variables are used to model information circulating between nodes of a model. Their values are calculated from the values of state variables thanks to a mechanism described by means of the so-called assertion.

When a transition is fired, first its action is executed to compute the values of state variables, second the assertion is executed to compute the values of flow variables.

The behaviour of components is described inside nodes. Nodes can be assembled into hierarchies, their input and output flows can be connected and their transitions can be synchronised. Nodes can be stored in the libraries of reusable components and are reused by instantiation, like in structured programming languages. AltaRica is an asynchronous language: only one transition can be fired at a time. However, it offers a mechanism to synchronise events. For example, common cause failures, shared repair crews, broadcasts can be represented by means of synchronisations. There are three versions of AltaRica modelling language:

- AltaRica LaBRI, the first version of the language developed by LaBRI [2];
- AltaRica DataFlow, the second version of the language implemented in several industrial tools [7];
- and AltaRica 3.0 implemented in the OpenAltaRica platform by AltaRica Association and IRT SystemX [4].

**Acausal and causal models** The main difference between the three versions of the language is the semantics of the assertion (calculation of values of flow variables).

In the first version of the language, AltaRica LaBRI, the assertion is a set of constraints. There is no input or output variables and in that way, it is possible to represent acausal models. Each time a transition is fired, first the action of the transition is executed, second a set of constraints (the assertion) is resolved to calculate the values of flow variables. Constraints have a big expressive power. However, in general, solving constraints involves multiple computation iterations and may be very resource consuming. A set of constraints may have several acceptable solutions resulting in non-deterministic model. In addition, there may be no solution. In this case, the initial model is incorrect.

To be able to assess industrial scale models, a second version, AltaRica DataFlow, has been created, reducing the expressive power of the assertion [7]. Its semantics is based on Mode Automata [14]. In this version of the language, the assertion is a set of DataFlow assignments. Each flow variable is assigned only once in the model and there is no circular definition. So, it is only possible to represent causal models. The order of the execution of the DataFlow assignments is calculated only once during the compilation of the model. When a transition is fired, first, the values of state variables are calculated; second, the values of the flow variables are calculated by executing the DataFlow assignments only once, which is more efficient than resolving constraints.

The semantics of the third version of the language, AltaRica 3.0, is defined in terms of Guarded Transition Systems [3]. To be able to model easily some kind of looped systems, for instance networks or electrical systems, AltaRica 3.0 introduces the concept of bidirectional assignment. If $x$ and $y$ are variables, $x :=: y$ is a bidirectional assignment, which is equivalent to two assignments: $x := y$ and $y := x$. The assertion is an instruction, where each variable may be defined in several assignments and there may be circular definitions. After each transition firing, first the action of the transition is executed, second, the values of flow variables are calculated by fixpoint solving of the assertion. In general, fixpoint solving of the assertion is more resource consuming than calculation of DataFlow assignments, but less resource consuming than resolving constraints.

Note that loops encountered in communication networks or electrical systems are different from the feedback control loops introduced in Section 2 and fixpoint solving does not resolve the modelling problems. Modelling techniques presented in this article should be used in that case for both AltaRica DataFlow and AltaRica 3.0 modelling languages.

**AltaRica DataFlow** In the remainder of this article we focus on AltaRica DataFlow. It is used as a description language of several industrial modelling tools: Cecilia Workshop (Dassault Aviation, Satodev), Simfia V3 and SimfiaNeo (Apsys). Many industrial scale experiments have been conducted with this version of the language [5, 6, 13]. It has been used to assess the safety of the flight control system in the frame of the certification of the Dassault Aviation Falcon

7X. A set of efficient assessment tools has been developed, including a Fault Tree compiler [14], a generator of critical sequences of events, and a stepwise simulator. The definition of timed and stochastic semantics of AltaRica DataFlow made it possible to develop a compiler to Markov chains [15], a probabilistic model-checker [19] and a Monte Carlo simulator [10].

## 4 Case study modelling and analysis using AltaRica DataFlow

### 4.1 Issues raised by failure propagation modelling of systems with control feedback loops

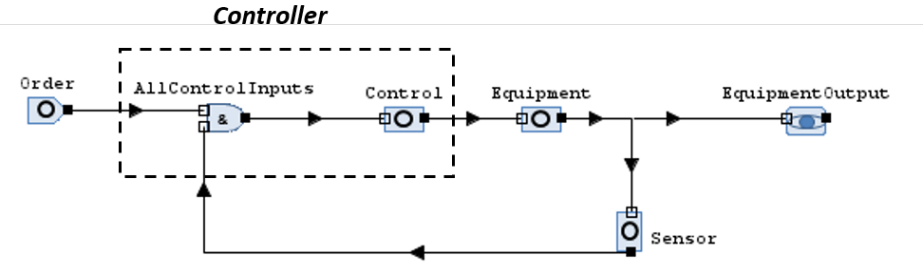


**Fig. 2.** Graphical representation of the AltaRica DataFlow model of the Case study.

Fig. 2 shows a graphical representation of the AltaRica DataFlow model of the case study. The behaviour of each component is represented by an AltaRica node. The node **Sensor** represents the behaviour of the sensor, the node **Order** - the behaviour of the operator order, the node **Equipment** - the behaviour of the equipment under study, nodes **AllControlInputs** and **Control** represents the behaviour of the controller. The node **EquipmentOutput** is a safety artefact, it is an observer on the status of the equipment output and models the Failure Conditions.

The internal state of each node is represented by three values: $\{ok, lost, err\}$, where *ok* represents the nominal behaviour, *lost* represents the loss of the component, *err* represents the erroneous behaviour of the component.

Nodes **Order**, **Control**, **Sensor** and **Equipment** have two events **fail_err** and **fail_loss** representing the failure modes of these components.

The node **AllControlInputs** is a logical node, it does not have any internal state. If at least one of the inputs is erroneous, then the output is erroneous. If not, if at least one of the inputs is lost then the output is lost. Otherwise the output is *ok*.

**Table 2.** AltaRica DataFlow modelling framework.

| **Component** | **Input and output flows** | **State variables** | **Transitions** | **Assertions** |
|---|---|---|---|---|
| Order | Input: N/A<br>Output: O<br>Type: $\{ok, lost, err\}$ | $S \in \{ok, err, lost\}$<br>Initially ok. | `S=ok \|- fail_loss ->`<br>`S:= lost;`<br>`S=ok \|- fail_err ->`<br>`S:= err;` | `O=S;` |
| AllControl-Inputs | Input: I1, I2<br>Output: O | N/A | N/A | `O = case{`<br>`I1=err or`<br>`I2=err :`<br>`err,`<br>`I1=lost`<br>`or I2=lost`<br>`:lost, else`<br>`ok };` |
| Control<br><br>Sensor<br><br>Equipment | Input: I Output: O<br>Type: $\{ok, lost, err\}$ | $S \in \{ok, err, lost\}$<br>Initially ok. | `S=ok \|- fail_loss ->`<br>`S:= lost;`<br>`S=ok \|- fail_err ->`<br>`S:= err;` | `O =case {`<br>`S=ok: I,`<br>`S=lost:`<br>`lost,`<br>`else err };` |
| Equipment-Output | Input: I Output: O<br>Type: $\{ok, lost, err\}$ | N/A | N/A | `O=I;` |

The details of the AltaRica nodes are given in Table 2.

The AltaRica DataFlow model given Fig. 2 is not correct because the assertion of this model is not DataFlow.

We say that, there is a cycle of equations in the assertion if there is a flow variable which depends on itself in the assertion. In other words, there is a circular definition in the assertion. In practice, cycles of equations can be detected during a compilation thanks to the dependency graph of the assertion. If the dependency graph of the assertion has cycles, then there is a cycle in the equations of the assertion.

Indeed, in our example there is a cycle in the equations of the assertion, the output of the node **Equipment** depends on itself.

During the compilation of the model an error is detected (see for example Fig. 3), that should be corrected. There are different strategies to solve this problem. They are presented below.

### 4.2 "Cut the Loop" solution

The simplification or "cut the loop" solution modifies the model in order to solve the equation cycle by "cutting" the control loop in the system, and by ensuring the safety model analysis is still representative of the studied system. Most of the time, it is necessary to add assumptions and explanations in order to achieve

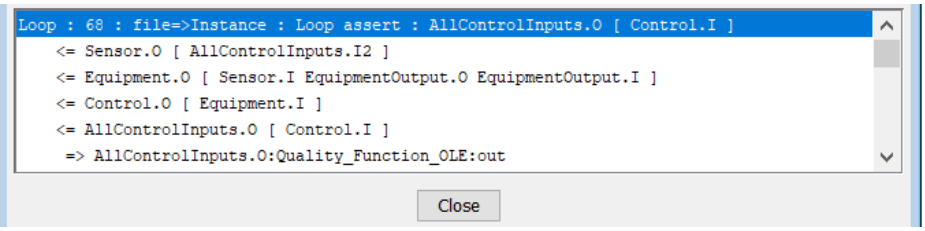


**Fig. 3.** Example of an error: cycle in the assertion.

this goal. For instance, instead of analysing the control loop illustrated Fig. 1 we can choose to perform the analysis on a model with a simplified control loop as illustrated Fig. 4.

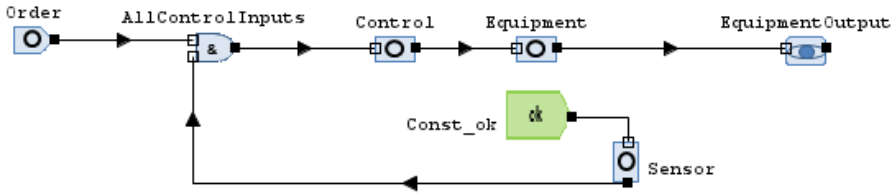


**Fig. 4.** Illustration of the "cut the loop" solution.

In our example, the existing control loop is "cut" using a safety artefact shown in green in Fig.4. The purpose of the "Const_ok" node is to use the same Sensor node than the one previously defined. It is a numerical "cap" that sends an "Ok" input to the Sensor. This is equivalent to use a Sensor node with no Input.

The key point in this solution is to ensure that the analysis performed with the simplified model is as representative as the one performed with the complete model.

**Model validation and analysis** This solution is valid if the simplified model is representative of the studied system despite the simplification. Note that to achieve this goal, in some cases, it may be necessary to provide additional analysis to the model output. In the example, we consider that the safety analyst who decides to cut the loop will check that the cutsets obtained by "cutting the loop" are representative of the control loop. This is the case as shown in Table 3.

In particular, we check that the Sensor failures (fail_loss and fail_err) lead to the Failure Conditions as it is expected (loss of the Sensor leads to the Loss of

**Table 3.** Cutsets for the "Cut the loop" solution.

| Cutsets for FC1: Loss of equipment output | Cutsets for FC2: Erroneous equipment output |
|---|---|
| Control.fail_loss | Control.fail_err |
| Order.fail_loss | Order.fail_err |
| Euipment.fail_loss | Equipment.fail_err |
| Sensor.fail_loss | Sensor.fail_err |

equipment output and erroneous Sensor leads to an erroneous equipment output). In addition, the others components' failures effects are unchanged. Indeed, the failures of the order, control or equipment (fail_loss and fail_err) directly lead to the equipment output corresponding failures. In this example, the Sensor State ("ok" or "failed") has a direct effect. In other words, in case of an erroneous sensor, the Failure Conditions "FC2: Erroneous equipment output" is directly reached. Counterexample: In case of the addition of a consolidation between the two inputs of AllControlInputs (one input ok and the other erroneous leading to the loss of the output), "cutting the loop" solution, proposed in this example, is not valid. It leads to lose the information captured by the Sensor. In that different case, an erroneous order, equipment or control then leads to the loss of the equipment output (FC1) and not to an erroneous equipment output (FC2). It is still possible to "Cut the loop" by linking directly the AllControlInputs to the EquipmentOutput and by cutting the loop right before the Control node, nevertheless the resulting model is very far from the initial system.

**Advantages and drawbacks of the approach** The interest of this approach is to solve the equation cycle in the assertion by using a static modelling. Compared to dynamic modelling, static modelling enables shorter cutset computation time. In addition, it is possible to generate Boolean equations from static models without loss of information. In that case, this modelling choice can allow solving huge industrial scale models. Eventually, in addition, for static models the probabilities computation is straightforward.

Nevertheless, the simplification of the model leads to a model closer to what the safety specialist has in mind (and could write down using Fault Tree Analysis approach) than to the initial system description. Modellers can choose to make the model look like the system, for instance by adding some graphical artefacts or "empty" links. In that case, they introduce an artificial consistency between the safety and the system models that may lead to misunderstandings and future mistakes.

Only the output just before the "loop cut" is affected by all the failure modes. Consequently, this approach is only valid (in terms of resulting cutset) when the control loop has only one output (here **EquipmentOutput**). When the control loop has several outputs (to the FCs or to the other parts of the model) some information may be missing. In our example, if Control output is an input for a monitoring positioned after the sensor, this new node does not see the impact of

the failures of equipment. This approach is efficient when the loop can be "cut" before nodes (the sensor here) that only affect the system when they fail (State ok or failed here). Its use is limited when the components are involved in the functional description of the system (for instance in the monitoring).

### 4.3 The "Dirac" solution

The "Dirac" solution introduces a safety artefact to handle the equation cycle. It allows the modelling of all dependencies between the output and input flows by introducing a state variable. In order to solve the equation cycle in the example illustrated in Fig. 5, we introduce this safety artefact through the modelling unit named "FeedbackDelay".

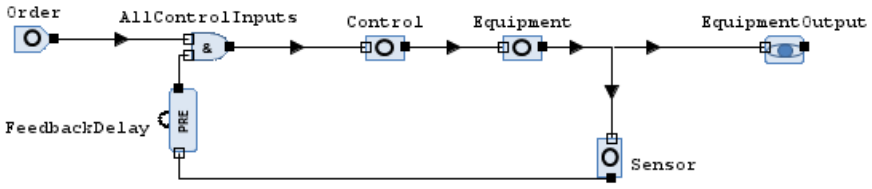


**Fig. 5.** Illustration of the "Dirac" solution.

The "FeedbackDelay" modelling unit contains:

- A state variable `prev_val` (previous value) that is initially ok;
- Two flow variables: `I` and `O`;
- An assertion: `O = prev_val`;
- An immediate deterministic event called "update", associated with the probability distribution Dirac(0);
- A transition `not (I = prev_val) |- update -> prev_val:=I;`, which allows to remove the direct flow dependency between the output value (`O`) of the node and its input value (`I`) and to introduce a dependency between the input `I` and the state variable `prev_val`.

The defined transition can be read: when the condition (input value `I` is different from the state value `prev_val`), the deterministic event `update` is instantaneously triggered (because it follows a Dirac(0) law). As a result, `prev_val` is assigned to the current value of `I`. Because of the assertion the output `O` takes immediately the same value, resulting in the propagation of the failure mode.

The introduction of a state variable set to "ok" initialises the problem to be solved when no failure is triggered. This solves the equation cycle for the initial state. At this stage it is interesting to note that the state variable introduces a "memory" effect on the transition. Indeed, the state value of `prev_val` will

change only when the transition conditions are fulfilled. This is why the modelling artefact we have presented is often called a "Delay". It does not refer to quantitative time (e.g. measured in second) but to sequential time, i.e., the order in which the different updates happen.

**Model validation and analysis** The minimal cutsets calculated for the model given in Fig. 5 are the same as the ones obtained for the previous solution given in Table 3.

The "Dirac" solution does not require specific validation, except for the local validation of the dedicated component. The stepwise simulator can be used to validate the model behaviour. In addition, we can also outline that special care shall be taken when several immediate (Dirac(0)) transitions are introduced in the model that can be enabled at the same time.

**Advantages and drawbacks of the approach** The proposed model is very close to the studied system. It allows a close representation of the system control. Consequently, it will be easier to validate the model with system engineers. It will also be easier to use this model to communicate to others or to capture the system behaviour.

Introducing a deterministic transition may lead to have a dynamic model. When this is the case, the tool solver will generate all the possible sequences of failures leading to the top events while for a static model it would be sufficient to generate all the combinations of failures (i.e. cutsets) or to solve directly a Boolean equation. Consequently, the computation time becomes more important than for a static model. At worst, for very huge industrial scale systems this computation time can be a blocking point. In addition, when there are several deterministic transitions, their synchronisation and priority of triggering need to be handled. This adds complexity to the model.

### 4.4 The "double flow" solution

The "double flow" solution relies on the addition of artificial flows to deal with the dependencies in the model. As shown in Fig. 6, the dependencies between the variables are modelled through two different paths.

Firstly, failure modes of all components are "collected" by the flows from Order to Sensor (underneath path). In this underneath path, the output of AllControlInputs does not depend on the Sensor output. Then, **AllControInputs** gets a second output. Each **AllControInputs** output is related to an input (see Table 4).

**Model validation and analysis** The validation of this approach is the same as the one discussed for the "cut the loop" approach. It is needed to demonstrate that the model is representative of the modelled system. Additional analyses may be required to justify this choice. The cutsets are as expected (the same as given in Table 3) and validate the model outputs.

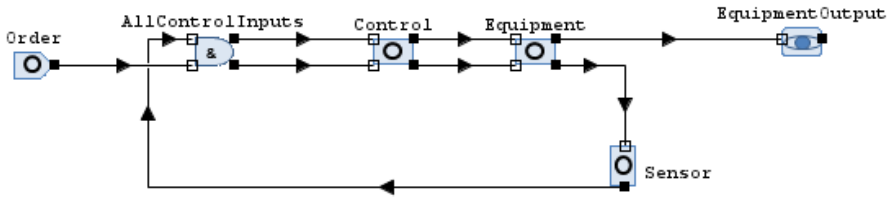


**Fig. 6.** Illustration of the "double flow" solution.

**Table 4.** AltaRica DataFlow modelling framework for the "double flow" solution.

| **Component** | **Input and output flows** | **State variables** | **Transitions** | **Assertions** |
|---|---|---|---|---|
| AllControl-Inputs | Input: I1, I2<br>Output: O1, O2 | N/A | N/A | `O1 = I1;`<br><br>`O2 = I2;` |
| Control | Input: I1, I2<br>Output: O1, O2<br><br>Type: $\{ok, lost, err\}$ | $S \in \{ok, err, lost\}$<br><br>Initially ok. | `S=ok \|- fail_loss -> S:= lost;`<br><br>`S=ok \|- fail_err -> S:= err;` | `O1 =case { S=ok: I1, S=lost: lost, else err };`<br><br>`O2 =case { S=ok: I2, S=lost: lost, else err };` |
| Equipment | Input: I1, I2<br>Output: O1, O2<br><br>Type: $\{ok, lost, err\}$ | $S \in \{ok, err, lost\}$<br><br>Initially ok. | `S=ok \|- fail_loss -> S:= lost;`<br><br>`S=ok \|- fail_err -> S:= err;` | `O1 =case { S=ok: I1, S=lost: lost, else err`<br>`O2 =case { S=ok: I2, S=lost: lost, else err };` |

**Advantages and drawbacks of the approach** The interest of this approach is to solve the equation cycle using a static modelling. This choice can thus reduce the model computation time. In addition, the probabilities computation is straightforward. This approach is usable in case of control loops with several outputs (when several downstream components depend on the control loop output).

This approach is the one requiring the more safety artefacts, making the model and the justifications heavier. As a consequence, it is mostly used for local loops, with few involved components.

### 4.5 Summary

We presented three different solutions that can be used to solve an equation cycle in the assertion of AltaRica DataFlow models. All of them have their advantages and drawbacks.

The "Cut the loop" solution is simple and conserves static modelling. But it is not always possible to use and needs additional validation by the analyst.

The "Dirac" solution is always possible. The model can be validated using stepwise simulation. The model in most of the cases is dynamic which may greatly increase computation time for large scale models. In addition, when several immediate transitions are used in the model, this greatly increases the model complexity and its validation.

The "Double flow" solution adds some artificial flows. But the model stays static and conserves the efficiency of calculations. However, additional validation of the model should be provided by the analyst.

## 5 Conclusion and perspectives

In this article we presented different strategies that can be used to represent control loops with AltaRica DataFlow modelling language. We show that there is no best solution to solve the problem. All the proposed solutions have their advantages and drawbacks.

Our future works will focus on the identification of other types of modelling problems raised in the domain of dynamic failure propagation modelling and on the definition of modelling strategies for these problems.

## References

1. Eurocae ed-135 guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment
2. Arnold, A., Griffault, A., Point, G., Rauzy, A.: The altarica language and its semantics. Fundamenta Informaticae **34**, 109–124 (2000)
3. Batteux, M., Prosvirnova, T., Rauzy, A.: Altarica 3.0 assertions: the why and the wherefore. Journal of Risk and Reliability (2017), article accepted
4. Batteux, M., Prosvirnova, T., Rauzy, A.: Altarica 3.0 in 10 modeling patterns. International Journal of Critical Computer-Based Systems **9**(1–2), 133–165 (2018). https://doi.org/10.1504/IJCCBS.2019.098809
5. Bernard, R., Aubert, J.J., Bieber, P., Merlini, C., Metge, S.: Experiments in model-based safety analysis: flight controls. In: Faure, J.M. (ed.) Proceedings of IFAC workshop on Dependable Control of Discrete Systems. pp. 43–48. Curran Associates, Inc., Cachan, France (June 2007), iSBN 9781617389948

6. Bieber, P., Blanquart, J.P., Durrieu, G., Lesens, D., Lucotte, J., Tardy, F., Turin, M., Seguin, C., Conquet, E.: Integration of formal fault analysis in assert: Case studies and lessons learnt. In: Proceedings of 4th European Congress Embedded Real Time Software, ERTS 2008. SIA (electronic proceedings), Toulouse, France (January 2008), code R-2008-01-2B04
7. Boiteau, M., Dutuit, Y., Rauzy, A., Signoret, J.P.: The altarica data-flow language in use: Assessment of production availability of a multistates system. Reliability Engineering and System Safety **91**, 747–755 (2006)
8. Bouissou, M., Bouhadana, H., Bannelier, M., Villatte, N.: Knowledge modelling and reliability processing: presentation of the figaro modelling language and associated tools. In: Proceedings of Safecomp'91 (1991)
9. Güdemann, M., Ortmeier, F.: A framework for qualitative and quantitative model-based safety analysis. In: Proceedings of 12th High Assurance System Engineering Symposium. pp. 132–141 (2010)
10. Khuu, M.: Contribution à l'accélération de la simulation stochastique sur des modèles AltaRica Data Flow. Thèse de doctorat, Université de la Méditerranée (Aix-Marseille II) (2008)
11. Mohrle, F., Zeller, M., Hofig, K., Rothfelder, M., Liggesmeyer, P.: Automated compositional safety analysis using component fault trees. In: 2015 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). pp. 152–159. IEEE (2015)
12. Papadopoulos, Y., Walker, M., Parker, D., Rude, E., Hamann, R., Uhlig, A., Gratz, U., Lien, R.: Engineering failure analysis and design optimization with hip-hops. Engineering Failure Analysis **18**, 590–608 (2011)
13. Quayzin, X., Arbaretier, E.: Performance modeling of a surveillance mission. In: Proceedings of the Annual Reliability and Maintainability Symposium, RAMS'2009. pp. 206–211. IEEE, Fort Worth, Texas, USA (January 2009), iSBN 978-1-4244-2508-2
14. Rauzy, A.: Mode automata and their compilation into fault trees. Reliability Engineering and System Safety **78**, 1–12 (2002)
15. Rauzy, A.: An experimental study on iterative methods to compute transient solutions of large markov models. Reliability Engineering & System Safety **86**(1), 105–115 (2004)
16. Rauzy, A.: Probabilistic Safety Analysis with XFTA. AltaRica Association, Les Essarts le Roi, France (2020)
17. Signoret, J.P., Dutuit, Y., Cacheux, P.J., Folleau, C., Collas, S., Thomas, P.: Make your petri nets understandable: Reliability block diagrams driven petri nets. Reliability Engineering & System Safety **113**, 61–75 (2013). https://doi.org/https://doi.org/10.1016/j.ress.2012.12.008
18. Signoret, J.P., Leroy, A.: Reliability assessment of safety and production systems : analysis, modelling, calculations and case studies / Jean-Pierre Signoret and Alain Leroy. Springer Series in Reliability Engineering Ser, Springer, Cham, Switzerland (2021)
19. Teichteil-Königbuch, F., Infantes, G., Seguin, C.: Epoch probabilistic model-checking. In: Model Based Safety Assessment Workshop. Toulouse, France (2011)