# Architecture patterns for safe design•

C. Kehren*, C. Seguin*, P. Bieber*, C. Castel*, C. Bougnol¤, J.-P. Heckmann¤, S. Metge¤

* ONERA, 2 av. E. Belin, B.P. 4025, 31055 Toulouse Cedex, France
{name}@cert.fr

¤ AIRBUS, 316 route de Bayonne, 31300 Toulouse, France
{name}@airbus.com

## ABSTRACT

The design, prototyping and analysis of complex systems architectures are often very difficult because of their important size. Our modelling experience of several aircraft systems in AltaRica allowed us to exhibit component assemblies whose aim is to ensure the safety of the architectures. The reuse of these assemblies, made generic, that we call Safety Architecture Patterns, simplifies these different tasks. In this paper, we report how SAP allow to obtain a more synthetic view of a system and to exhibit its satisfied properties on an A320-like electrical system case study.

## INTRODUCTION

Recently, AIRBUS and ONERA have launched an R&T study on architecture patterns to enhance safe design activities. The purpose of this study is to provide methods and computer based tools to assist the modelling and the assessment of safety architectures of all kinds of complex systems (e.g. software, mechanical). The basic idea is to encode experts' know-how into formal model libraries of typical safety micro-architectures.

Those micro-architectures models exhibit elements of interest for a safer design: structural features (e.g. redundancy), good use condition and induced safety properties. They are indeed abstract views of the system safety elements and will be called Safety Architecture Patterns (SAP). Sets of SAP are identified both in literature and in Airbus practice. We propose now to encode them into formal notations such as the AltaRica [1] language. This language was developed at Bordeaux University and is supported by toolsets including simulation, fault tree generation and model-checking capabilities. We have been inspired by computer science studies where design patterns have been introduced to ease software development process by allowing the reuse of mature application templates. This approach has been used, as an experiment, to model Airbus electrical and hydraulic generation and distribution systems.

The paper has the following structure. The first section describes the SAP concepts. Then we present basic operations that can be used to manipulate SAP. The last section is the application of the SAP approach on an A320-like electrical system and the presentation of the results we obtained.

## PRESENTATION OF SAFETY ARCHITECTURE PATTERNS

Safety Architecture Patterns are system abstractions that highlight useful attributes from the safety point of view. In a first part, we present the attributes we selected. Then we illustrate this generic definition with a concrete example of SAP. Finally, we give a flavour of the formalisation developed to support the concept by computer based tools. This formalisation is an extension of a formal language called AltaRica to linear temporal logic operators. Consequently, attributes of a SAP are declared using this formalism. Analyses can then be easily made with AltaRica tools or verification tools like SMV [2] for instance.

### ATTRIBUTES

SAP describes pieces of architecture commonly used in safety critical systems. There exist a wide set of these micro-architectures. So, we first tried to find out their important properties (attributes) for safety analyses. Then we proposed a formal structure to store them in libraries and use them in a systematic way with formal tools. Finally, we added an informal structure to ease SAP library browsing.

Indeed, the design of system architectures requires the combination of several patterns. Browsing a pattern library gathering various types of patterns makes it possible to simplify this design. In order to facilitate the comprehension of the structure of such a catalogue of patterns it is necessary to declare each SAP well. This declaration will be made using attributes in order to organize their relations into categories. The classification we chose is inspired by Zimmer's design pattern classification [3]. It proposes three categories of relations between a pattern A and a pattern B: A uses B in its solution, A is similar to B and A can be associated with B.
It allows, at the time of the search for a pattern allowing the resolution of a precise problem, to find the family of pattern proposing adapted solutions. Then we choose the pattern which meets the requirements. From the catalogue and classification we think of being able to automate the

• Published in the *Complex and Safe Systems Engineering 2004* international conference proceedings

instantiation of SAP in a concrete architecture and their detection.

As for design patterns, we will see that attributes are associated with each SAP. Some attributes are informal; for instance, the *problem* attribute is a text describing the issue that this SAP solves. Other attributes are described with formal notations. For instance, we used an AltaRica like notation to describe the SAP *architecture* attributes.

Let us now clarify the detail of each attribute. The SAP declaration is the following:

```
sap <name>
```

The informal attributes are declared in the `header` section.

```
 header
  problem: <string>
```

The "Problem" attribute makes it possible to describe the context in which the pattern applies, the problem which is exposed.

```
  solution: <string>
```

"Solution" specifies how the pattern solves the arising problem.

```
  links: [<relation><sap_name>]*
 redaeh
```

"Links" enumerates the possible relations with other SAP.

The second part of the attributes makes it possible to formalize the solution brought by the SAP and is declared in the `node` section:

```
 node
```

The architecture section is composed of the variables (called flows) declaration and the hierarchy (i.e. sub-nodes used in the SAP). Flows are input, output or local variables that model both functional and dysfunctional parameters. Sub-nodes are instances of more elementary SAP.

```
  flow
   <declare_input>*;
   <declare_output>*;
   <declare_local>*;

  sub
   <declare_sap>*;
```

To ease the validation of the composition of SAP we chose to distinguish the assumptions on the environment from those required by the SAP itself and called "body" hypothesis. The interest is to clarify the significant requirements for the proof and to allocate the responsibilities.

```
 assert
  body
   [<assumption_name>: <formula>]*;

  environment
   [<assumption_name>: <formula>]*;
```

Finally, under those assumptions and this architecture the SAP guarantees a set of safety properties.

```
guaranteed
 [<guaranteed_property_name>: <formula>]*;
edon
pas
```

EXAMPLE

We will now present an example of SAP. We will take as example a cold redundancy and will follow the previously defined SAP declaration procedure. This SAP uses the elementary *block* pattern in its solution. We will thus start by introducing this last briefly.

The first part of the declaration relates to the informal attributes of the pattern.

```
sap block
 header
  problem: "the pattern is elementary
   consequently it does not bring a direct
   solution to an arising problem"
  solution: "none"
  links: none
```

The second part of the declaration relates to the formal attributes. We start with the architecture declaration. A *block* has the following variables: an input `i`, an activation order `a`, a resource `r` and a failure f. It fulfils a function `o` (see Figure 2). All these variables are Boolean.

```
node block
 flow
  i, a, r are Boolean inputs;
  f, o are a Boolean outputs;

 assert
  body
   output_law: To provide an output (i.e.
    o = true) a component needs to be
    activated (a = true), to have resource
    (r = true)), to have an input (i =
    true) and of course not to be in a
    failure state (f = false);
```

There is no environment assumption for the *block*.

```
  environment
   void;
```

The guaranteed property is:

```
  guaranteed
   output_provision: The block always
    provides output if it is activated, has
    resource, has something in input and is
    not in a failure state;

edon
```

We can now introduce the cold redundancy.

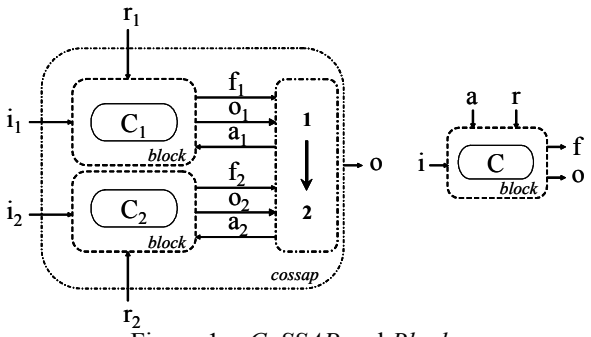

Figure 1 – *CoSSAP* and *Block*

The cold redundancy pattern is made of two elementary *block* patterns activated by a controller (on the right side of the *CoSSAP* in the above figure).

```
sap cossap
 header
  problem: "the arising problem is one
   failure tolerance"
  solution: "the proposed solution is the
   function passive duplication"
  links: use block
 redaeh

node cossap
 flow
  i1, i2, r1, r2: bool in;
  o, f1, f2: bool: out;
  o1, o2, a1, a2: bool: local;
```

The *CoSSAP* pattern inherits the body hypothesis of the *blocks* it is composed of.

```
 sub
  B1 and B2 are instances of block;
```

The body hypotheses of the *CoSSAP* are:

```
 assert
  body
   input_connection: B1.i=i1 and B2.i=i2
    and B1.r=r1 and B2.r=r2;

   output_connection: f1=B1.f and f2=B2.f;

   internal_connection: B1.a=a1 and B2.a=a2
    and o1=B1.o and o2=B2.o;

   output_law: The output of the SAP is
    always equal to the output of B1 or B2
    which compose it;

   B1_activation: If B1 is not in a failed
    state then it is activated;

   B2_activation: The failure of B1
    involves the activation of B2;
```

The environment assumption relates to resource:

```
  environment
   B1_resourse: Activation of B1 and no
    failure of B1 always imply resource for
    B1;

   B2_resourse: Activation of B2 and no
    failure of B2 always imply resource for
    B2;

   B1_input: Input from B1 is always true;

   B2_input: Input from B2 is always true;
```

Under those assumptions and this architecture the SAP guarantees that the *CoSSAP* is fault tolerant:

```
  guaranteed
   output_provision: The function realized
    by the SAP is one failure tolerant;
edon
```

## FORMALIZATION

SAP structure is inspired by AltaRica hierarchy concept. Now we present the language used to formalize the SAP assertions.

These assertions are used for several purposes. For instance, they establish relations between flows of the component and so, describe how component outputs are determined by component inputs. They can also describe more high level properties such as "any component shutdown request must end up being satisfied". This property deals with the dynamic aspect of the SAP we declared. Linear Temporal Logic [7], fragment of CTL* and SMV model-checker specification logic, makes it possible to handle this type of dynamic properties easily.

As a result, we chose to write the hypothesis and properties using LTL temporal operators G, F and X (representing "globally", "finally", and "next" respectively). Traces explaining the meaning of these temporal operators are represented on the figure hereunder.

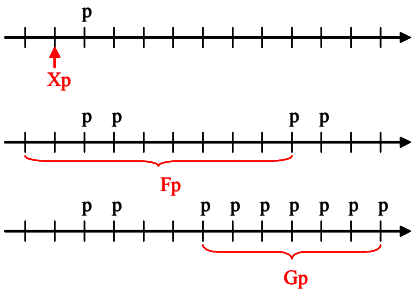

Figure 2 – LTL Temporal Operators

The use of these temporal operators will allow us to make a distinction between temporary breakdowns and permanent ones.

These concepts are illustrated in the following example of the cold redundancy safety pattern.

```
node block
 flow
  i, a, r: bool in;
  o, f: bool out;
assert
  body
   output_law: G(i and a and r and not(f)
    -> o);
  environment
   void;
  guaranteed
   output_provision: G(i and a and r and
    not(f)-> o);
```

```
edon

node cossap
 flow
  i1, i2, r1, r2: bool in;
  o, f1, f2: bool: out;
  o1, o2, a1, a2: bool: local;
 sub
  B1, B2: block;
 assert
  body
   input_connection: B1.i=i1 and B2.i=i2
and
    B1.r=r1 and B2.r=r2;
   output_connection: f1=B1.f and f2=B2.f;
   internal_connection: B1.a=a1 and B2.a=a2
and
    o1=B1.o and o2=B2.o;
   B1B2_merge: G((o1 or o2) <-> o);
   B1_activation: G(not(B1.f) -> B1.a);
   B2_activation: G(B1.f -> X B2.a);
  environment
   B1_resource: G((B1.a and not(B1.f)) ->
    B1.r);
   B2_resource: G((B2.a and not(B2.f)) ->
    B2.r);
   B1_input: i1 = true;
   B2_input: i2 = true;
  guaranteed
   output_provision: G(not(B1.f and X B2.f)
    -> F o);
edon
```

It is worth noting that the formalization is sound only if we are able to prove that body and environment assumptions ensure the guaranteed properties. We used the capabilities of McMillan's model-checker to demonstrate it.

## SAP OPERATIONS

The main goal of the architecture patterns is to capitalize the experts' solutions in the field of safety. The re-use of these recurring and mature solutions will allow saving time in the systems architectures design and analysis phases. Indeed, SAPs have pre-proven properties which will facilitate the analysis of the system. Their use also makes it possible to obtain a synthetic view of the system and to exhibit the properties satisfied by the model for a faster comparison with the requirements contained in the safety documents. These uses are possible thanks to two basic SAP operations: SAP composition and SAP recognition. In this section, we present the mechanisms of these basic operations. We will show how these operations are used to structure the safety assessment of a complex system in the next section.

### SAP COMPOSITION.

*Composition* of SAP1 and SAP2 consists in linking outputs of SAP1 to inputs of SAP2 and checking the following conditions:

- Plugged input/output have the same type of value;

- SAP1 provides SAP2 with the expected services: assumed environment properties of SAP2, that are related to SAP2 inputs connected to SAP1 outputs, are fulfilled by SAP1 guaranteed properties under SAP1 environment conditions.

Thus the validity of a SAP composition is established not only by type checking the involved interfaces, but requires also discharging proof obligations. Such proofs were achieved in our framework thanks to modular proof means of Cadence SMV. As counterpart, the composition preserved guaranteed properties.

SAP composition enables fast prototyping of safe system architectures. On one hand, each SAP gathers a subset of components. So SAP composition provides more quickly architecture than composition of elementary components. On the other hand, each SAP is also characterized by a set of requirements: derived safety requirements for the interfaced components (environment), intrinsic (body) properties, and requirement fulfilled (guaranteed) under assumptions on the environment. So each SAP allows highlighting the safety requirement allocations inside the architecture. This enables to drive the design by safety requirements.

Moreover, when an architecture is depicted by compositions of SAP, a global safety requirement can be assessed according to two strategies:

- The assessment is based on the body features of each SAP. This strategy is always applicable.

- A proof establishes that the global requirement only results of environment and guaranteed properties. This strategy may fail. Indeed, guaranteed properties are abstractions of body properties and can omit important details when trying to prove a specific global safety requirement. However, each time this strategy is successful, it provides an evidence of safety architecture robustness: it is a guarantee that sufficient details of importance are covered by the requirement allocation.

### SAP RECOGNITION

*Structural mapping* of SAPi with an architecture ARCH consists in recognizing only the architectural part of a SAP. This means:

- Mapping each basic block of SAPi with a subset of ARCH components.

- Linking interfaces and internal flows of SAPi with flows of the selected subset of components. Each link states how a SAPi flow is defined in function of ARCH flows. Simplest links are identity of flows.

*Refinement* of SAPi in architecture ARCH with respect to the structural mapping of SAPi with ARCH consists in checking that all ARCH behaviours are similar to SAP behaviours modulo correspondences between flows stated in the structural mapping. Refinement checking is implemented in SMV Cadence model-checker, for instance.

*Properties mapping* of SAPi in architecture ARCH with respect to the structural mapping of SAPi with ARCH consists in deriving, from SAP properties, concrete requirements that are applicable to ARCH. The derivation is done by substituting names of SAP flows by their definitions in the structural mapping.

It is worth noting that refinement checking is a strong checking. As a consequence, when it has been proved that a SAPi satisfies a requirement "prop" expressed as an LTL formula, and also proved that ARCH refines SAPi then we can conclude, without anymore proof, that ARCH satisfies the concrete mapping of "prop" as illustrated in the figure below.

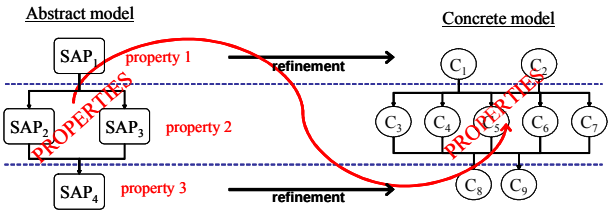


Figure 3 − Property preservation by refinement

Classical design processes recommend to first perform an abstract model and then to refine it into a concrete model. Nevertheless, it is also possible to rebuild *a posteriori* the abstract model and then, refinement techniques can be used to validate the abstraction. In such case, we get back to the previous case where proved properties on the abstract model are preserved on the concrete model, modulo the properties mapping.

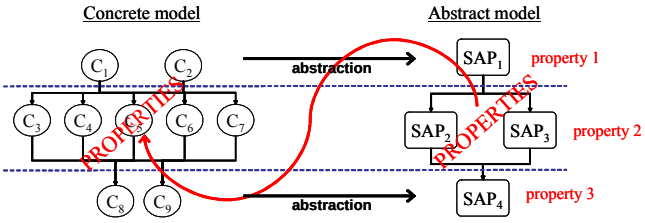


Figure 4 − Property preservation by abstraction

Thus, the combination of SAP composition and SAP refinement/abstraction steps enables various demonstration strategies illustrated in the next section.

## A320-LIKE ELECTRICAL SYSTEM CASE STUDY

### CASE STUDY PRESENTATION

The electrical system we modelled [9] contains approximately 70 components: sources, contactors, circuit breakers, transformer-rectifiers, inverters, bus bars… This model is derived from an A320 electrical system.

The architecture is made up of 2 main electrical sources (one source per engine) and of an emergency source. The two main sources are in cold redundancy, this redundancy being itself in cold redundancy with the Auxiliary Power Unit. In case of total electrical loss, an emergency source fed by a Ram Air Turbine provides voltage to the critical bus bars.

The power supply of the electric bars is made using three independent distribution lines named `side1`, `side2` and `ess`. Transformers/Rectifiers (TR) are used to convert AC into DC. For safety considerations, we distinguish nominal bars, whose loss is not considered catastrophic, from the "essential" bars which feed critical components.

The overall system layout is shown in the figure hereunder.

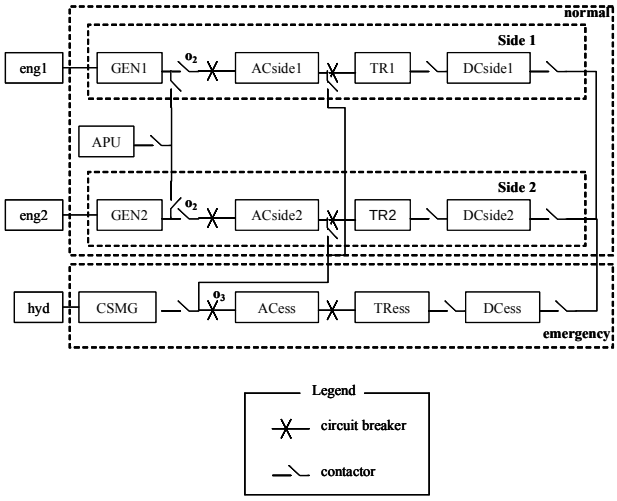


Figure 5 − Electrical System Model

### USE OF SAP

We will now show in detail the validation of this A320-like electrical model thanks to safety architecture pattern recognition. The aim, as seen earlier, is to check that this architecture satisfies a set of safety requirements listed in the Airbus documents. The Symbolic Model Verifier has been used to check the validity of a set of properties formalizing some of these requirements [9]. In the following, we will only consider those concerning the loss of the electrical bars.

The requirements concerning the electrical bars can be simplified as follows:

− The loss of an electric bar must be caused by at least one failure.
− The loss of two bars must be caused by at least two failures.

The preceding requirements were translated into Linear Temporal Logic. Those concerning the loss of a bar were translated in the following way:

```
G(upto_0_failure) -> G(bus_bars_ok)
```

The translation of the requirements relating to the loss of two bars is:

```
G(upto_1_failure) -> GF(ACside1_ACside2_ok)
G(upto_1_failure) -> GF(ACside1_ACess_ok)
G(upto_1_failure) -> GF(ACside2_ACess_ok)
G(upto_1_failure) -> GF(DCside1_DCside2_ok)
G(upto_1_failure) -> GF(DCside1_DCess_ok)
G(upto_1_failure) -> GF(DCside2_DCess_ok)
```

The use of the temporal operator F (finally) is imposed by the reconfiguration time necessary in case of a failure of a component in the system.

Let us now take the example of the cold redundancy composed of the two principal electrical sources.

IDENTIFICATION PROCESS

We consider that basic components of the detailed architecture are similar to the basic SAP block. For instance, a generator (GEN) is a block where input, activation and resource are always available. A contactor (CT) is a block where resource is always available. We will now try to identify the piece of architecture composed of a GEN and its associated CT with a *block* pattern.

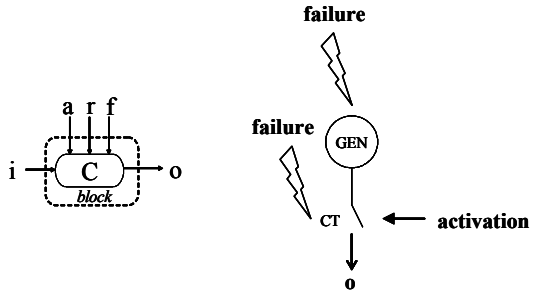


Figure 6 − *Block vs* Concrete Assembly

We first have to associate to the SAP variables the variables or combination of variables of the concrete assembly.

| SAP | | ELEC |
|---|---|---|
| *input variables* | | |
| i | → | *true* |
| a | → | CT.activation |
| r | → | *true* |
| f | → | GEN.failure or CT.failure |
| *output variable* | | |
| o | → | o |

Table 1 − Variables Identification

To simplify, we chose to set the input and resource variables to `true`. Under those equivalence classes, we demonstrated that `ELEC` was a refinement of the `SAP` specification (i.e. considering the same input values, the SAP model and an instantiation provide the same output values). According to the identification process described in the preceding section, we still have to check that the environment assumptions hold on the concrete assembly. This verification is simple since there is no environment assumption in the *block* specification. Consequently, it is possible to carry out this substitution. The same process is then repeated for the second *block* pattern concerning the second generator and its contactor.

Finally, by property mapping, the two instantiated *block* guaranteed properties are:

```
G((CT.activation and not(GEN.failure or
CT.failure)) -> F o)
```

The assembly, we identified with a *block*, includes the activation of the electrical source (i.e. the contactor). Moreover, we know that it is impossible to feed one side of the system with two sources simultaneously. Thus, we will identify the combination of these two *blocks* with a cold redundancy. The composition of these *blocks* into a *CoSSAP* must now be validated.

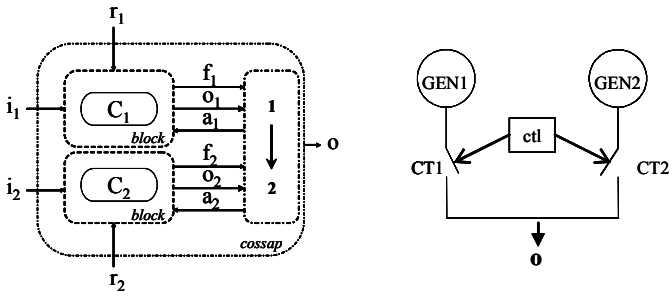


Figure 7 − *CoSSAP vs* Concrete Assembly

We showed that the assemblies (GEN1, CT1) and (GEN2, CT2) were refinements of *blocks*. To be able to substitute the whole piece of concrete architecture with a *CoSSAP* we must demonstrate that the electrical controller is a refinement of the *CoSSAP* controller. This is done the same way as for the *block*.

We still have to check that the environment properties hold on the concrete assembly. The instantiated *CoSSAP* environment properties are:

```
G((CT1.activation and not(GEN1.failure or
 CT1.failure)) -> true);

G((CT2.activation and not(GEN2.failure or
 CT2.failure)) -> true);
```

These two properties are tautologies. As a result, we can substitute the piece of architecture of the above figure with a *CoSSAP*. The resulting guaranteed property by *CoSSAP* located in the area #1 in the overall system layout of Figure 8 is:

```
(1) G(not((GEN1.failure or CT1.failure) and
 X(GEN2.failure or CT2.failure)) -> F o1);
```

By repeating this process we obtain the following formulae for the *CoSSAP* located in the area #2. These *CoSSAP* are inputs for AC and DC side1 and side2 (i.e. non essential) bus bars.

```
(2) G(not(((GEN1.failure or CT1.failure)
 and X(GEN2.failure or CT2.failure)) and
 XX(APU.failure or CT3.failure)) -> F o2);
```

It is worth noting that in this *CoSSAP* instance activation of the second block (the one related with the APU) is delayed by one time unit. As a result, we notice that the output is tolerant to 2 faults and thus satisfies the requirement.

Concerning the area #3 we can demonstrate the following:

```
(3) G(not((((GEN1.failure or CT1.failure)
 and X(GEN2.failure or CT2.failure)) and
 XX(APU.failure or CT3.failure)) and
 XXX(CSMG.failure or CSMG.failure))-> F o3);
```

which means that *CoSSAP* in this area are tolerant to 3 faults. This is satisfactory with respect to the requirements.

The result of these consecutive substitutions is shown on Figure 8. We notice the reduced complexity of the model which results in a better comprehension of the several possible safety reconfigurations.

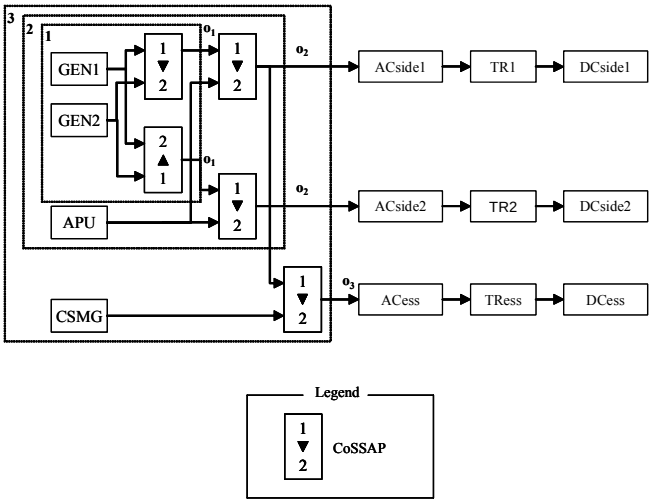


Figure 8 − SAP-based Model

## CONCLUSION

As a conclusion, these first experiments showed that SAP based models enable capitalization of the experts' know-how and the possibility to reuse validated patterns in new architectures. They also provide a more concise view of the whole system and ease the exhibition of the satisfied properties. As a benefit, we expect this approach will offer a better and wider understanding of system safety and will save time during the preliminary safety assessment phase. Another benefit of SAP is rapid prototyping of system models. Indeed, using an SAP library has shown that the design of system architectures is made easily and quickly. This allows the creation of several prototypes. The system designer can compare those competitive solutions of architecture in terms of feasibility or cost for instance. Moreover, safety requirements are automatically assessed thanks to the available verification technologies. We used Cecilia OCAS graphical simulation and fault tree generation tool coupled with SMV model-checker. Next step in the design process is to shift from a SAP based model to a more concrete one. Pre-proved properties of the SAP can be used to make easier the safety requirements allocation from high-level system safety objectives. Furthermore, we are studying a manner of automating the identification and substitution process of SAP within an architecture.

## REFERENCES

[1] A. ARNOLD, G. POINT, A. GRIFFAULT, A. RAUZY. *The AltaRica formalism for describing concurrent systems.* Fundamenta Informaticae n°40, p109-124, 2000

[2] K.L. MacMILLAN – *Symbolic Model Checking*. Kluwer Academic Publishers, 1993, ISBN 0-7923-9380-5

[3] W. ZIMMER – *Relationships between design patterns*. Pattern Languages of Program Design, p345-364, Addison-Wesley, 1994, ISBN 0-201-60734-4

[4] GFI Consulting – *Cecilia Workshop*. http://www.gfi-oasys.com/french/cecilia/o_presentation.htm

[5] A. RAUZY – *Mode automata and their compilation into fault trees*. Reliability Engineering and System Safety, 2002

[6] G. POINT, A. RAUZY – *AltaRica: constraint automata as a description language.* Journal Européen des Systèmes Automatisés n°33, p1033-1052, 1999 http://iml.univ-mrs.fr/~arauzy/publis/publis.html

[7] Z. MANNA, A. PNUELI – *The temporal logic of reactive and concurrent systems.* Springer-Verlag, New-York, 1992, ISBN 0-387-97664-7

[8] G. POINT – *Contribution à l'unification des méthodes formelles et de la sûreté de fonctionnement.* University of Bordeaux Ph.D. thesis, 2000

[9] C. KEHREN et al. – *Advanced multi-system simulation capabilities with AltaRica*. International System Safety Conference '04, Providence, RI, USA.