System Fault Tolerance Specification: Proposal of a Method Combining Semi-formal and Formal Approaches

Giovanna Dondossola (contact author), Oliver Botti

ENEL R&D Department
Via Volta 1, Cologno Monzese 20093 Milan, Italy

E-mail: dondossola@pea.enel.it
Phone: +39 2 72245478
Fax. +39 2 72245465

Abstract. The topic of the present work is the specification of system Fault Tolerance (FT). FT is considered a valid technique for increasing the dependability of critical automation systems by adding them the ability to operate in presence of faults. Two basic considerations stimulated the development of the present work. Firstly although a considerable amount of concepts and theory have been published around FT, a full-organized method supporting their application to the FT needs of a specific system is still missing. Furthermore, the availability of a methodology oriented to the specification of system FT is especially useful in view of integrating available FT software layers according to specific system needs. Goal of the present work is therefore to develop a methodology for the FT specification, to be used as a tool supporting the configuration of the tailorable FT software layer, which is currently under development within the TIRAN Project[1]. The presented approach to the FT specification is based on a combined use of two general-purpose specification methods: the UML (Unified Modeling Language) graphical method and the TRIO (Tempo Reale ImplicitO) temporal logic. The main novelty of the proposed method consists in the identification and organization of a sequence of specification steps, which drive the industrial user in collecting and analyzing system dependability requirements and then in designing FT solutions, possibly tailoring already existing and configurable FT mechanisms.

1. Introduction

Purpose of this work is the construction of a methodological scheme in support to the development of *dependable systems*. Deterministic aspects of dependability properties, which concern how the system faces the possibility of faults independently by their occurrence probabilities, are specifically addressed. With respect to high-

[1] The TIRAN (TaIlorable fault toleRANce frameworks for embedded applications) Esprit Project is partially funded by the IT Programme of the Commission of the European Communities as project n° 28620. The partners of the TIRAN Project are ENEL-R&D (Italy), SIEMENS (Germany), TXT Informatica (Italy), EONIC Systems (Belgium), Katholic University of Leuven (Belgium) and University of Turin (Italy).

T. Maibaum (Ed.): FASE 2000, LNCS 1783, pp. 82-96, 2000.

level dependability means like fault prevention, fault removal, fault forecasting and fault tolerance the scheme focuses on *fault tolerance (FT) techniques* which increase system reliability, availability and integrity by preventing system faults from producing system failures [Lapr95].
Each system stating a set of FT requirements has to be realised by adopting a specific FT solution. More or less explicitly high level FT requirements express constraints on the FT strategy to be adopted and/or about suitable configurations/compositions of FT steps and their related mechanisms.
As an example, let us consider the following high level FT requirement:
"When a hardware fault occurs and remains unrepaired for at least δ time units, the system must be able to find the damaged part and put it off-line."
The above requirement expresses a FT strategy addressing hardware faults composed by FT steps such as error detection (*"When a hardware fault occurs…"*), fault diagnosis (*"…the system must be able to find the damaged part…"*) and system reconfiguration (*"…and put it off-line."*). The triggering of the FT strategy is due to output from the error detection step and conditioned by the duration of fault permanence in the unrepaired state (*"…and remains unrepaired for at least δ time units…"*), therefore requiring some time-out mechanism.

The proposed methodological scheme concentrates on the first steps in the development of a FT solution, which concern *high level FT requirement specification and how they constrain the FT solution*. It supports the collection and organization of FT requirements into a semi-formal model and their formal specification and analysis. The work is part of the TIRAN Esprit Project whose main objective is the development of a tailorable software framework providing a set of FT mechanisms amenable for real-time and distributed automation systems [Bott99]. In the context of the TIRAN project, the main goal of this work is the definition of a methodological support addressed to both

- *application designers*, in capturing system-specific high level FT requirements
- *users of the TIRAN framework*, in tailoring its use to the FT needs of the particular system.

The methodological scheme is based on a systematic organization of well-known dependability concepts (due to Laprie [Lapr95], [Lapr98] and recent R&D experiences [EFTOS97] about FT flexible solutions. The novelty of the approach consists of the definition of a methodology organized into a sequence of steps, which drive the user in first collecting and analyzing his/her FT requirements and then composing his/her FT solutions.
The whole scheme, overviewed in section 2, is composed by four distinct specification supports which make use of informal, semi-formal and formal techniques at the aim of producing, for a given system, a certifiable specification of its FT requirements. The present paper summarizes the first three steps of the scheme fully presented in [TIRAN99]:

1. in the first step, described in section 3, the FT specification is based on the UML [UML97a] [UML97b] semi-formal approach, as supported by the tool Rational Rose [Rose98]
2. the second step, described in section 4, exploits the TRIO formal language [Ciap97a] to express the requirements related to FT specification in a formal way
3. the third step exploits the formal techniques made available by TRIO to perform formal analyses on the FT requirements.

The scheme has been applied to model the FT requirements of an ENEL system, the Primary Substation Automation System (PSAS) which consists of different modules managing an electric substation[2]. The proposed application combines protection, command and control, monitoring and supervision capabilities and is a good representative for most of the dependability requirements of the energy field, in terms of integrity, security, availability and EMI immunity. It also demands for distributed and heterogeneous platforms with High Performance Computing capabilities. However, due to space reasons, only one example of the application of the formal part of the FT scheme to the PSAS system has been reported in sections 4 and 5.

2. Scheme Overview

In order to allow its wide exploitation, the scheme has been developed by separating the use of semi-formal techniques for specifying FT requirements from the use of fully formal methods for performing formal V&V activities on FT requirements.
To non-formal specifiers the scheme proposes a model for the specification of FT requirements based on the Unified Modeling Language or UML [UML97a] [UML97b], in its version supported by the tool Rational Rose [Rose98]. As UML is

- *an object-oriented graphical language* ⇒ therefore highly expressive
- *an OMG standard* ⇒ a pre-requisite to its industrial spreading
- *supported by visual modeling tools* ⇒ therefore an "easy to learn" technology
- *strongly emerging* in the market of semi-formal methods for system design

it showed to be an adequate mean for communicating the basics of the generic scheme for the specification of FT requirements, as well as for applying the scheme on specific FT systems.
The UML-based support to FT specification represents an entry-level use of the scheme, which requires a very low overhead, paid back by significant improvements in the specification inter-operability. It is centered on

- *System composition*, introducing system components/functions and their attributes
- *Fault/Error/Failure* (FEF) classes characterized by hierarchical relationships, attributes and associations with system components/functions
- *FT step* classes characterized by hierarchical relationships, attributes and associations with fault classes.

As a second step the scheme proposes to formalize the specification of FT requirements, thus endowing FT requirements with all the specification and V&V benefits recognized to formal methods [Rush95] [NASA95] [MOD91]. A specific formal method called TRIO is used to specify FT requirements. TRIO (Tempo Reale Implicito) [Ciap97a] is a product of ENEL research conceived for the specification of real time systems, also experimented by industries like Ansaldo, Volvo and Sextant.

[2] A joint project with the ENEL Distribution Division is ongoing to renew the existing PSAS. One of the goals of this joint project is to improve the efficiency and quality of service, by increasing the system dependability and by reducing its costs. Target plants are about 2000 existing and new Primary Substations (PS).

The TRIO formalization is finalized to produce a fully formal version of the FT requirement specification expressed in TRIO forms[3].
The third step of the scheme makes use of TRIO formal techniques to perform V&V activities on the FT requirement specification. Specifically two levels of support to V&V are provided:

- Level A: static analysis dealing with the V&V of the fault tolerance strategic plan
- Level B: dynamic analysis dealing with the V&V of system behaviors.

The fourth, final step of the scheme establishes generic links between system-specific FT requirements and their related FT solutions. A central issue towards FT design is the identification of appropriate FT mechanisms. There are several mechanisms supporting fault tolerance. Each mechanism may be used to realize several types of FT steps and may contribute fulfilling several system-specific FT requirements. FT mechanisms may be explicitly referred in the FT requirements of a specific system, but it may also be the case that the requirements do not refer to them. Some FT mechanisms could have alternative configurations, which need to be set in the FT design specification. The following design activities have to be considered by the fourth step of the methodology:

- choice of FT mechanisms: a set of FT mechanisms fulfilling the requirements has to be identified depending on the specified FT steps and design constraints
- choice of the target platform
- verification of design constraints: time/space/redundancy constraints stated in the requirements have to be satisfied by the correspondent bounds associated with the selected FT mechanisms and platform.

The instantiation of this step of the scheme on a specific application represents a support to a system-specific tailoring of the TIRAN framework driven by FT requirements. However, as this step has not been developed in detail yet, it will not be further described in the rest of the paper.

3. UML-Based Support to FT Specification

The first step of the methodological scheme is expressed in the UML notation. This means that UML models, to be intended as meta-models to be instantiated by each application of the methodology to a specific FT system, capture methodological guidelines. The UML methodological step consists in a set of class hierarchies distributed into a structure of **Packages**[4]. Each Package encapsulates a set of inner packages/classes and their relationships, visualized by means of **Package/Class Diagrams**.[5]

---

[3] The possibility of supporting the non-formal specification of FT requirements by adopting the TRIO graphical editing capabilities [Bert97] has been investigated, this choice allowing the automatic integration of the semi-formal and formal steps of the scheme. However, in order to favor a wide usability of the first, semi-formal step of the methodology in industrial contexts, the adoption of the UML standard notation has been preferred.

[4] All the words referring to the UML terminology are written in **bold** face.

[5] Due to space constraints, this section visualizes only a simple example of the UML diagrams built by using the tool Rose but visualizes just one UML diagram. The application of the

The top-level **Package** is named *Methodology*[6] and includes three **Packages**, namely
1. *System Model* addressing system requirements relevant to FT specification
2. *FEF (Fault Error Failure) Model* supporting the description of fault/error/failure classes and modeling the *FEF chain*
3. *FT Strategy Model* concerning the identification of FT steps.

According to the object-oriented approach, using the UML scheme for specifying FT requirements of a given system means
- to select sub-parts of the scheme models which are relevant for the specific system
- to assign values to class attributes from the scheme models
- to refine/specialize sub-parts of models from the scheme.

3.1 System Model

The Package *System Model* includes four inner Packages supporting
- the definition of the conceptual structure of an FT system, i.e. the identification of (classes of) its components (Package *Composition*)
- the association of functions to system components (Package *Functions*)
- the association of real time requirements to system components and/or functions (Package *Time Requirements*)
- the association of dependability attributes to system components and/or functions (Package *Dependability*).

The model of a specific system is obtained specializing classes provided in the *System Model* and representing system-specific components, functions, time constraints, attributes and associations.

The Package *Composition* concerns system structure. The definition of system composition supports system partitioning in fault confinement area, which establish independence among faults at the aim of avoiding fault propagation. As the system structure is typically system-specific, it will be mainly defined at application level. The methodological level may provide generic structures for categories of systems. By addressing in particular the category of automation systems, the very basic meta-structure in Fig.1 is made available defining
- the decomposition of the class *Automated System* into *Automation System* and *Plant* (the field) as a UML aggregation relationship
- the decomposition of the classes *Automation System* and *Plant* in *Automation Component* and *Plant Component*, respectively
- the interface between *Automation Component* and *Plant Component* as a UML association called *commands*
- the class *Automation Operator* (which may be *local* or *remote*) associated with an *Automation Component* by the association *interfaces.*

As far as system functions are concerned, the methodology provides a very simple meta-model, which associates system functions to system components and operators.[7] Communication functions are characterised by the following attributes:

UML scheme to the PSAS system has been omitted here and the interested reader may refer to Chapter 7 of [TIRAN99] where a step by step instantiation of the scheme is reported.

[6] All the words referring to element identifiers of the UML model are written in *italic* face.

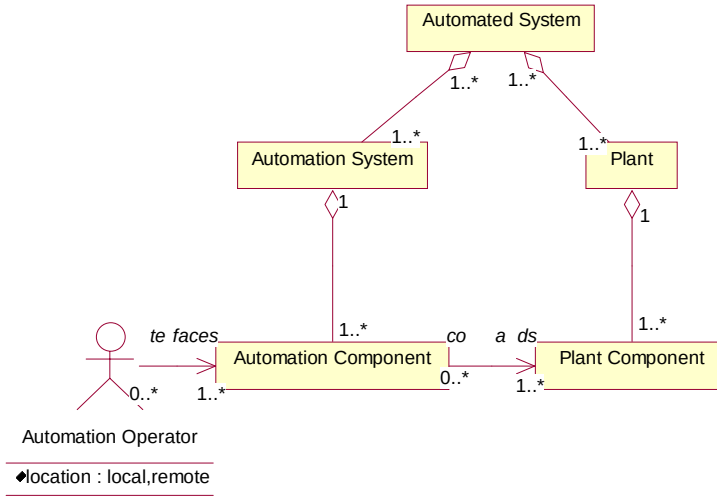

Fig. 1.: the Main Class Diagram in the Package Composition.

- *Transmission*: the mode in which data are transmitted
- *Channel*: the type of physical support to communication
- *Bandwidth*: the information quantum supported by the channel in the time unit
- *Location* that distinguishes communications internal to a component from communications towards the plant, remote systems/operators, local components/operators.

The Package *Time Requirements* identifies two attributes, which are considered particularly relevant for the FT specification of real time systems:

- the **attribute** *cycle time* (Tc) refers to the interaction between the system/component and its external environment (the plant)
- the **attribute** *execution time* (Texec[f_i]) which is the (maximum) response time required by a function f_i
- cycle and execution times are related by Tc < Min (Texec[f_i]).

In the case of automation systems the values of these attributes are tightly coupled with the real time constraints stated by the plant process on the automation system and their definition greatly influences the instantiation of the appropriate FT strategy with its associated mechanisms.

The methodology considers dependability attributes whose quantitative or qualitative estimates may be provided. Both repairable and non-repairable dependable systems are characterized by the following attributes:

- *criticality*: a qualitative estimate of the risk consequent to a failure of a given function/component/system. Its type is an enumerative that identifies criticality levels. A three-level criticality scale may be adequate for most critical domains

[7] A more detailed model for the Package *Functions* could be provided by identifying a set of generic classes of automation functions such as monitoring, command and control.

- *complexity*: a qualitative estimate of the complexity degree of the system. Its type is an enumerative, which identifies complexity levels. Complexity levels have to be determined on a strictly system-specific base
- *MTTF (MeanTimeToFailure):* the expectation of the mean time to failure.

In repairable systems/components three other dependability attributes may be added:

- *MTTR (MeanTimeToRepair)*: the expectation of the time to restoration
- *MTBF (MeanTimeBetweenFailures):* the expectation of the time between two failures
- *Availability:* the probability that the system is ready for specified usage.

Dependability attributes may be defined at system/component/function level.

3.2 Fault Error Failure Model

The FaultErrorFailure (FEF) model captures general concepts on the fault theory partially expressed in the literature and partially derived from previous experience. Three different Packages (*Fault Model, Error Model, Failure Model*) compose the FEF model, each package characterizing a different view of a fault evolution, from its appearance to its recovery and/or repair.

Faults are error causes that usually affect a system component but then may propagate to other components through their interactions.

The Package Fault Model contains a class diagram representing the fault classification due to Laprie [Laprie98][8] as a fault hierarchy. The root class *Fault* of the inheritance tree describes a generic fault in terms of the following attributes:

- *fault rate:* frequency of a fault occurrence
- *location:* faults may affect three logical parts of a system component, namely memories (MEM), elaboration units (ELAB) and communication units (COM)
- *latency:* the length of time between the occurrence of a fault and the appearance of the corresponding error.

The first level of the inheritance tree distinguishes *Physical Fault* from *Design Fault.*

Physical faults may be either *Permanent Fault* or *Temporaneous Fault.*

Permanent physical faults are specialised by the following sub-classes:

- *DevPerm Fault:* internal, permanent faults due to the development phase
- *OpInt Fault:* internal, operational faults that have their origin within hardware components and are constantly active
- *OpEx Fault:* external, operational faults induced on the system by the physical environment

Temporaneous physical faults are specialised by the following sub-classes:

- *DevTemp Fault:* internal, temporary faults due to the development phase
- *Intermittent Fault:* internal physical defects that become active depending on a particular pointwise condition
- *Transient Fault:* faults induced by environmental phenomena

[8] Only those classes that are relevant with respect to the FT scope have been included.

Design faults are specialised into

- *Systematic Fault:* accidental, permanent faults (flawed algorithms) that systematically turn into the same errors in the presence of the same input conditions and initial states
- *Intentional Fault:* intentional, though not malicious, faults (basically compromises introduced at design time).

Faults are related to system components by a multiple association named *location*.

Errors are deviations from the correct state of the system. They are caused by faults affecting system components and are related to some functions of the faulty component. The model assumes that errors affect locally the functions of the faulty component before propagating to another interacting component.

The Package Error Model contains a class diagram representing an error hierarchy that is especially useful for the associations between FT mechanisms and the kind of errors they are able to manage.

The root class *Error* introduces the following basic error attributes:

- *latency*: the length of time between the occurrence of an error and the eventual appearance of the corresponding failure
- *PE:* an estimate of the Probability of the Error.

The first level of the error hierarchy identifies three error sub-classes, namely *Processing Error, Communication Error* and *Memory Error*. A Processing Error may be sub-classified by one of [*Runtime Error, Late Processing Error, Memory Violation Error, Corrupted Processing Error*]. A Communication Error may be sub-classified as one of [*Late Communication Error, Corrupted Communication Error, Disordered communication Error*]. Additional attributes are introduced locally to error sub-classes. In particular a *BER* (Bit Error Rate) attribute is associated to the class *Corrupted Communication Error.*

Errors are related to system functions by a multiple association named *location*.

Failures are deviations of the service delivered by the system from fulfilling its intended function. The root class *Failure* of the inheritance tree in the Package Failure Model is characterized by the following basic attributes:

- *PF:* an estimate of the Probability of the Failure
- *criticality:* consequences on the environment or failure criticality level. The failure criticality level of a specific class of failure influences the specification of its related failure mode assumption.

The different failure mode assumptions generate the following failure classes [Cri91][9]:

- *Omission Failure*: it occurs when an agreed reply to a well-defined request is missing. The request appears to be ignored
- *Timing Failure*: it occurs when the service is supplied, though outside the real-time interval agreed upon in the specification. Sub-classes are *Early Timing Failure* and *Late Timing (or Performance) Failure*
- *Response Failure*: it occurs when the system provides a wrong response. Sub-classes are *Value Failure* (when the system supplies an incorrect output) and *State-transition Failure* (when the system executes an incorrect state transition)
- *Crash Failure*: it occurs when the system continuously exhibits omission failures. Sub-classes are *Pause-crash Failure* (the system restarts in the state it had before

[9] The failure classification is an updated version of that reported in [TIRAN99].

its crash), *Halting-crash Failure* (the system never restart), *Amnesia-crash Failure* (a restarted system re-initializes itself wiping out the state it had before its crash) and *Partial Amnesia-crash Failure* (some part of a system's state is re-initialized while the rest is restored to its value before the crash).

The cause-effect relationships in the chain fault → error → failure → fault are captured by a UML class diagram named *FEF Chain*. The cause-effect association between a fault type and its provoked error type considers the value of the attribute *location* in the causing fault. Thus, for example, a fault localized on the memory of a system component may propagate as a memory error on some function of that component. An error may propagate into a failure and a failure may propagate into a fault located anywhere on a component related to the original faulty component.

3.3 FT Strategy Model

The UML model of the fault tolerance strategy proposed by the methodology includes the classification of FT steps due to Laprie [Laprie95]. Some extensions to that classification have been introduced in order to distinguish some FT steps concerning faults from steps belonging to the error processing family.

Any number of FT steps classified as *fault processing*, *error processing* and *fault treatment* may carry out fault tolerance.

Fault Processing aims at avoiding systematically the propagation of fault effects and includes two specialized sub-steps, namely

- *Fault Masking*: masks the fault using the available redundancy to enable the delivery of an error-free service. It does not assume error detection
- *Fault Containment*: prevents propagation of fault effects by means of either spatial (*Permanent Containment*) or temporal redundancy (*Temporary Containment*). Spatial redundancy may be on information (*Local Containment*), hardware or software (*Global Containment*) units

Error processing aims at removing errors from the computational state (if possible, before failure occurrence) and includes the following FT steps:

- *Error detection*: identifies states as being erroneous
- *Error diagnosis*: assesses the damages caused by error propagation before detection
- *Error isolation*: isolates the erroneous component from the other part of the system to prevent error propagation
- *Error recovery*: performs recovery after detection and includes
  - *Compensation*: recovery is performed using the present (erroneous, internal) state that contains enough redundancy to enable the delivery of an error-free service. Requires error detection
  - *Forward recovery*: recovers to a future state
  - *Backward recovery*: recovers to a past state

Fault treatment aims at assuring that the system fails according to the stated failure modes and at preventing faults from being re-activated and includes the following:

- *Failure handling*: performs a set of actions required to fulfil a given failure mode
- *Fault compensation*: after fault containment, it allows the system to provide a response to compensate for output of the faulty subsystem

- *Fault repair*: performs repairing actions
- *Fault diagnosis*: identifies the cause(s) of error(s)
- *Fault passivation*: removes faulty components and includes *Reconfiguration*, i.e. modifies the structure of the system such that non-failed components fulfil the system function, possibly at a degraded level.

FT steps are related to fault/error classes by a multiple association named *addresses*.

4. TRIO-Based Support to FT Specification

The structure of a system FT specification in the TRIO language may be obtained by translating in TRIO the system-specific Rose-UML diagrams. However the FT formalization includes some parts which have not a correspondence with the semi-formal FT specification but whose introduction V&V purposes have motivated. Formal methods are typically endowed with automatic analysis capabilities requiring that the formalization be adequately instrumented for performing that type of analysis[10].

TRIO is a temporal logic language that supports a linear notion of time: the *Time Domain* is a numeric set equipped with a total order relation and the usual arithmetic relations and operators (it can be the set of integer, rational, or real numbers, or any interval thereof). TRIO formulae are constructed in the classical inductive way, starting from terms and atomic formulas. Besides the usual prepositional operators (*&* for *and*, | for *or*, ~ for *not*) and quantifiers (*all*, *ex*), TRIO formulae may be composed by using a single basic modal operator called *Dist* that relates the *current time* (which is left implicit in the formula) to another time instant. Thus the formula *Dist (F, t)*, where *F* is a formula and *t* a term indicating a time distance, specifies that *F* holds at a time instant at *t* time units from the current instant. For convenience, TRIO items (*variables*, *predicates*, and *functions*) are distinguished into time-independent (TI) ones, i.e., whose value does not change during system evolution and time-dependent (TD) ones, i.e., those whose value may change during system evolution. Several *derived temporal operators* (ex. *Alw(F), Som(F), Becomes(F), Futr(F,t), NextTime(F,t)*) can be defined from the basic *Dist* operator through prepositional composition and first order quantification on variables representing a time distance.
TRIO provides object-oriented concepts and constructs, which support writing modular reusable specifications of complex systems. Among the most important o-o features are the ability to partition the universe of objects into classes, to introduce inheritance relations among classes, and to exploit mechanisms such as genericity to support the reuse of specification modules and their incremental development.
Classes denote collections of objects that satisfy a set of axioms. They can be either *simple* or *structured* –the latter term denoting classes obtained by composing simpler ones. A simple class is defined through a set of axioms premised by a declaration of all items that are referred therein. Some of such items (declared *visible*) are in the

[10] The fact that some specification parts are missed in the correspondent semi-formal specification does not mean in general that the semi-formal language is unable to express the concepts but that that kind of information is relevant for analysis purposes not considered during the semi-formal specification.

interface of the class, i.e. they may be referenced from outside it in the context of a complex class that includes a module that belongs to that class.

As an example let us consider the following two textual FT requirements of the ENEL PSAS system, labeled FR2 and FR19[11]:

FR2: "Corruption on input/output, elaboration, memory and internal communications, caused by any first fault (transient, intermittent or permanent) must be tolerated

a) allowing to preserve a working state acceptable and coherent with the history of the system
b) avoiding or handling any loss of control
c) avoiding to transmit wrong output to the plant, the operator, the remote systems."

FR19: "Proper evolution according to the system history needs to be guaranteed:

a) the evolution must be guaranteed between acceptable states
b) leaving an acceptable state must be allowed only towards an other acceptable state - e.g. by using mechanisms which maintain the current state (judged correct) till a confirmation of correctness of the next state (e.g. by using a Stable Memory technique [Deco98])
c) the transitions between two subsequent acceptable states must be non interruptible and without uncertainties - e.g. by an atomic action
d) evolution must be coherent among the different system components - e.g. by synchronizing the state transition of all the components within a single atomic action."

The combination of FR2a with FR19 demands for a fault masking technique applied to the PSAS state. In the TRIO formalization reported below such requirements have been fulfilled by a fault masking mechanism called Stable Memory, which integrates temporal and spatial (triple modular) redundancy to stabilize the PSAS state.

For the purpose of providing an initial PSAS formal specification, the Rose-UML diagrams have been manually translated in the TRIO language. However, such translation could be made automatic by completely defining the translation rules mapping Rose-UML $\rightarrow$ TRIO[12]. The construction and refinement of the PSAS initial TRIO specification for V&V purposes has been supported by the TRIO Editor.

The above PSAS FT requirements have been formalized by two TRIO classes, namely *PSAS* and *PSAS_Stable_Memory*. The class PSAS is characterized by the TI value *cycle_time*, the TD values *stable_state* and *new_state* and by the TD propositions *begin_cycle*, *end_cycle* and *confirm_state*. The PSAS cycle time is assigned to 3 time units by the axiom *set_cycle_time*. The PSAS FT behavior is abstractly defined by the last five axioms allowing to cyclically producing a new state (axioms *cyclic_behavior*, *produce_new_state* and *maintain_new_state*) which is considered stable under confirmation (axioms *maintain_stable_state* and *change_stable_state*). The confirmation of the new state, provided by the *PSAS_Stable_Memory*, is conditioned by its triple repetition, as expressed by the axiom *stable_state.*

---

[11] PSAS textual requirements, extracted from documentation produced by ENEL technicians for a call for tender, have been grouped into 4 categories labeled SR (System Requirements), DR (Dependability Requirements), FR (FT Requirements) and TR (Time Requirements).

[12] The VDM (Vienna Development Method) formal method has already adopted such approach by providing a new tool from IFAD called Rose-VDM^{++} Link. It supports round trip engineering between UML and VDM^{++} through an automatic bi-directional coupling of Rational Rose and the IFAD VDM Tools (see the IFAD WWW Page at http://www.ifad.dk).

```
class PSAS
    visible        end_cycle, new_state, confirm_state;
    temporal domain    integer;
    types          state_values = {state0,state1,state2,state3,state4,state5,none};
                   cycle_values = 1 .. 15;
    TI Items
      values       cycle_time : cycle_values;
    TD Items
      values       new-state : state_values;
                   stable_state : state_values;
      propositions     begin_cycle; end_cycle; confirm_state;
    vars           s1, s2 : state_values;
                   tc : cycle_values;
    axioms
    set_cycle_time:        cycle_time = 3;
    initialisation:        stable_state = state0 & new_state = none & begin-cycle;
    cyclic_behaviour:      Alw(all tc (cycle-time = tc →
                   (begin-cycle ↔ Dist(end_cycle & Dist(begin_cycle,1),tc))));
    produce_new_state:  Alw(all s1 (end_cycle & stable_state = s1 →
                                ex s2 (s2 <> s1 & Becomes(new_state = s2))));
    maintain_new_state: Alw(new_state = none ↔ ~end_cycle);
    maintain_stable_state:      Alw(all s1 (~confirm_state & stable_state = s1 ↔
                                    Dist(stable_state = s1, -1)));
    change_stable_state: Alw(all s1 (confirm_state & new_state = s1 ↔
                                Becomes(stable_state = s1)));
end   PSAS
```

```
class PSAS_Stable_Memory
    visible        end_cycle, new_state, confirm_state;
    temporal domain    integer;
    types          state_values = {state0,state1,state2,state3,state4,state5,none};
    TD Items
    values         new-state : state_values;
    propositions       end_cycle; confirm_state;
      vars         s1, s2, s3: state_values;
                   d1, d2: integer;
      axioms
      stable_state:        Alw(confirm_state ↔ all s1, s2, s3, dist1, dist2
                                (Becomes(new_state = s1) &
                                NextTime(Becomes(new_state = s2) &
                                NextTime(Becomes(new_state = s3),d2),d1) &
                                s1 = s2 & s2 = s3))
end   PSAS_Stable_Memory
```

5. TRIO-Based Support to FT V&V

Formal specifications may be analyzed by different Verification and Validation (V&V) techniques. Several formal V&V techniques have been developed

characterized by different degrees of formality [Dond00]. By applying formal V&V techniques for analysing FT properties of dependable systems we may distinguish two main levels of formal support, namely *static analysis* (level A) and *dynamic analysis* (level B).

The static analysis of a FT specification allows the extraction of distinct views of the FT static model captured by the FT formalization. Static analysis may be referred either to the whole system or to specific functions/components. The following ones are some examples of static analysis:

- analysis of relevant fault/error classes
- analysis of FEF chains
- analysis of the defined FT strategy, e.g. fault/error classes (not) addressed, masked fault classes, contained fault classes, detected error classes, isolated error classes, recovered error classes, handled failure classes.

The dynamic analysis of a FT specification implies a detailed analysis of FT behaviours, sometimes requiring the formal specification to be refined and enriched with related functional requirements. Examples of dynamic types of analysis are:

- FT consistency analysis (satisfiability proofs), e.g. generation of system behaviors and check of the absence of contradictions in the FT specification
- FT adequacy analysis (truth/false proofs) checking specific FT conditions
- proof of system FT properties
- automatic generation of fault injection cases for the conformance testing of the final system.

By considering the PSAS system the following analysis could be addressed:

A1 Which fault classes are associated to which system components?
A2 Which system functions are affected by corrupted-communication errors?
A3 Which are the FT steps addressing PSAS permanent faults?
B1 Find PSAS histories stabilizing a new state.
B2 Find PSAS histories which revoke a new unstable state.

The TRIO method is endowed with tools supporting three basic V&V techniques, i.e. model generation, property poof and test case generation[13], which are suitable for performing both levels of FT analysis. In order to perform V&V activities, the following proof settings must be provided to the TRIO tool:

a) load of the formal specification
b) selection of the axiom set to be considered
c) selection of the V&V technique to be applied
d) selection of the Temporal Domain and the Evaluation Instant.

Let us consider the case B1 and assume that the following values are given to the required settings above:

a) TRIO classes PSAS and PSAS_Stable_Memory reported in section 4
b) all axioms
c) model generation
d) Temporal Domain = 1 .. 12; Evaluation Time = 1.

The TRIO tool builds all possible histories satisfying the chosen set of axioms, i.e. sequences of time dependent literals (positive and negative items) located on the time

[13] In particular such techniques are supported by a new version of TRIO, which will be available at the end of the FAST Esprit project No. 25581, integrating the TRIO model generation within the NP-tools, by Prover Technology.

axis. The output of the model generation process is a graph representing for each relevant item its truth-value on the temporal domain. A tabular version of the B1 results are visualized below, where at each time instant only positive items are given.

Time Domain	**|= Model**
1	begin_cycle, stable_state = *state0*, new_state = *none*
2	stable_state = *state0*, new_state = *none*
3	stable_state = *state0*, new_state = *none*
4	end_cycle, new_state = *state1*, stable_state = *state0*
5	begin_cycle, stable_state = *state0*, new_state = *none*
6	stable_state = *state0*, new_state = *none*
7	stable_state = *state0*, new_state = *none*
8	end_cycle, new_state = *state1*, stable_state = *state0*
9	begin_cycle, stable_state = *state0*, new_state = *none*
10	stable_state = *state0*, new_state = *none*
11	stable_state = *state0*, new_state = *none*
12	end_cycle, new_state = *state1*, stable_state = *state1*

6. Conclusions and Future Usage

The paper has presented a methodological approach to the specification and analysis of high level fault tolerance requirements, which makes use of two general purpose methods, namely UML and TRIO.
A main global result of the methodology is the systematization of FT concepts and their relations aimed at defining articulated FT strategies, eventually focussed on specific system components and/or functions. On one side, the UML-based support represents an easy way to collect and organize FT requirements. On the other side the TRIO-based support provides formal techniques to validate the specification of such requirements.
The experimentation of the methodological approach to the ENEL PSAS system has allowed assessing the original textual requirements, which are now traceable on both (UML and TRIO) model items.

The methodological scheme will be experienced by ENEL within the TIRAN project itself over the PSAS pilot application. An evaluation of this experience will be provided, together with the methodology itself, to the future users of the TIRAN framework. It will represent a guided support for the specification and analysis of FT requirements as well as for framework configuration and verification.

References

[Bott99] O. Botti, V. De Florio, G. Deconinck, F. Cassinari, S. Donatelli, A. Bobbio, A. Klein, H. Kufner, R. Lauwereins, E. Thurner, E. Verhulst, *TIRAN: flexible and portable fault tolerance solutions for cost effective dependable applications*, in: Proc. of 5th Int. Conf. Europar'99 - Parallel Processing, Toulouse, F, Aug. 1999, LNCS, No.1685, Springer-Verlag.

[Ciap97a] E. Ciapessoni, A. Coen-Porosini, E. Crivelli, D. Mandrioli, P. Mirandola, A. Morzenti, *From formal models to formally-based methods: an industrial experience*, submitted to Transaction on Software Engineering and Methodologies, 1997.

[Cri91] F. Cristian, *Understanding fault-tolerant distributed systems*, in Communications of the ACM, 34(2): 56-78, February 1991.

[EFTOS97] G. Deconinck, T. Varvarigou, O. Botti, V. De Florio, A. Kontizas, M. Truyens, W. Rosseel, R. Lauwereins, F. Cassinari, S. Graeber, and U. Knaak. (Reusable software solutions for more fault-tolerant) Industrial embedded HPC applications. *Supercomputer*, XIII(69):23-44, 1997.

[Deco98] G. Deconinck, O. Botti, F. Cassinari, V. De Florio, R. Lauwereins, *Stable Memory in Substation Automation: a Case Study*, in: IEEE Digest of Papers of the 28th Annual Int. Symp. on Fault-Tolerant Computing (FTCS-28), Munich, Germany, Jun. 1998.

[Dond00] G. Dondossola, *A Scheme for Formal Methods Assessment in the context of developing Certifiable Control Systems*, paper to be published.

[Lapr95] J.-C. Laprie, "Dependability — Its Attributes, Impairments and Means", Section II.A from B. Randell, J.-C. Laprie, H. Kopetz, B. Littlewood (Eds.), "ESPRIT Basic Research Series: Predictably Dependable Computing Systems" Springer-Verlag, Berlin Heidelberg New York, 1995, pp. 3-18.

[Lapr98] J.-C. Laprie, "Dependability of computer systems: from concepts to limits", in Proc. of IFIP International Workshop on Dependable Computing and Its Applications (DCIA98), Johannesburg (South Africa), Jan. 12-14 1998.

[MOD91] UK Ministry of Defence: *Interim Defence Standard 00-55: The procurement of safety critical software in defence equipment*, Part 1, Issue 1: Requirements; Part 2, Issue 1: Guidance, April 1991.

[NASA95] Formal Methods Specification and Verification Guidebook for Software and Computer Systems, Volume I: Planning and Technology Insertion, NASA-GB-002-95.

[Rose98] "Rational Rose 98i Using Rose", Rev. 6.0, December 1998, (Software Release 98i).

[Rush95] J. Rushby, *Formal Methods and their Role in the Certification of Critical Systems*, SRI Technical Report CSL-95-1, March 1995 (300 pages). This is a shorter (50 pages) and less technical treatment of the material in [Rushby 93]. It will become a chapter in the FAA Digital Systems Validation Handbook (a guide to assist FAA Certification Specialists with advanced technology issues).

[TIRAN99] *D1-1: Requirements specification – Version V2*, TIRAN Project deliverable, Oct. 1999, (confidential).

[UML97a] "UML Notation Guide", version 1.1 September 1997.

[UML97b] "UML Semantics", version 1.1 September 1997.