Integration of F.M.E.A and SA-RT: a way to enhance dependability characteristics of critical systems?
A. Khalfallah, S. Bahsoun, H. Basson, & J.C. Derniame

CRIN, BP239, 54 506 Vandoeuvre, France
IUT du Velizy, 78 129 Velizy, France
Université du Littoral, IUT de Calais, 3 rue Louis David, 62226 Calais, France

ABSTRACT

This paper investigates critical systems from the software engineering perspective. It particularly focuses on analysing the system models building activities defined as Software Process Modeling and tries to lead to a response to problems such as:

- Is it possible to enhance the dependability characteristics of the system by using a development environment where constraints about the development activities can be expressed and enacted?
- Which operation could be automated in a critical system development environment leading to a better dependability.

Early activities in the system life cycle related to dependability (such as preliminary hazard analysis) are carried on by methods that exhibit two inconveniences: informality and lack of integration. For example, in the Failure Modes and Effects Analysis (F.M.E.A) this drawbacks are present: The formalism of F.M.E.A is limited to a tabular representation of the results from the analysis process and how the method relates to other activities of the life cycle is not clear.

We show that the use of a software process modeling formalism to describe the methods can alleviate - at least part of - these disadvantages. For this, we present the models for SART and FMEA methods, using the MASP/DL software process modeling formalism, which has been developed in the ALF project. Then merging of the two models achieves the integration between methods; this consists of identifying the links between the objects of the two models, establishing the global orderings of operations to maintain the coherence of artefacts and identifying the causes to effects relationships between operations and modeling them through control mechanisms.

INTRODUCTION

Many authors have pointed out the need to analyse the safety requirements for both software and hardware aspects of critical systems since the earliest stages of the development [1,2,3,4,5]. The specifier must determine which services are expected by the users and which qualities of services must be provided in term of dependability, reliability, safety. The approaches to realize this study can
Software Quality Management

-roughly- be classified in two broad categories: formal methods and structured methods. The former class is characterised by its proof capabilities while the latter offers ease of use and readability qualities. Examples of the former are presented in [4,6,7,8], examples of the latter are the so called inductive and deductive methods [1,9,10,11,12].

In this paper we focus on two structured methods which are used at the same stage of the system life cycle, but which have different objectives. The studied methods are Failure Mode and Effect Analysis (F.M.E.A.) [12] and Structure Analysis for Real Time (S.A.R.T.))[16]. They exhibit two well-known inconveniences of structured methods:

- Informality: There is no clear formalisms and models with well defined semantics underlying the methods. The concepts manipulated are defined by reference to examples and particular situations. How to carry the analysis process is also described through general guidelines and rules of the thumb.

- Lack of integration: Which manifests itself by the absence of connection between the method's scope and other system development activities. The method doesn't make clear how it can use results from other activities such as functional specification, which activities must precede and follow the method and so on...

This paper aims at showing that the use of a software process modeling formalism to describe the methods can alleviate at least part of these disadvantages.

The software process is defined as "...the technical and management framework established for applying tools, methods and people to the software task" [13] and the Software process model as the manner in which the different tools and software objects should be coordinated to support a software process [14].

The paper is structured as follows: in section 2 the two methods SA-RT and F.M.E.A are introduced. Section 3 will introduce the software process modeling language called MASP/DL we have been developing in the context of the ALF project. In section 4 we will develop a process model for SA-RT and another one for F.M.E.A. In these models some semantics of the artefacts produced by the methods will be expressed by representing object structures and relationships, allowed operations on objects and integrity constraints. Integration between the methods is achieved by merging the two process models. Finally section 5 will discuss the two following points:

-Which operations could be automated in a critical systems development environment?

-Is it possible to enhance the dependability characteristics of the system by using an ALF based environment where the models can be enacted ?

F.M.E.A AND SA-RT METHODS

The choice of F.M.E.A and SA-RT to illustrate the contribution of process modeling to the conception of safety critical systems is motivated by:

- The two methods present different levels in the formality of the definition of their concepts. One may qualify SA-RT as being "semi-formal" because even though the constructs it offers to modelize the system under development have no formal semantic associated to them, their definition is
precise enough to allow for simulation of the final system model. Whereas for F.M.E.A there is no model underlying it. This difference will permit the comparison of the contribution of process modeling in the two cases.

- The two methods apply to the same phase of the life cycle, i.e., the functional specification phase. On that account, the relationships between the two methods have a rich semantic which modelization is of interest.

- The two methods address disjoint objectives: SA-RT is concerned with functional requirements analysis and specification. The sole non-functional requirements SA-RT is concerned with are the expression of real-time constraints. In the other hand, F.M.E.A do not include functional specification analysis. This difference of goals entitle us to foretell that it is desirable to integrate the two methods.

Failure Modes and Effects Analysis method

The F.M.E.A method is intended for a qualitative analysis of the Dependability -and more especially the safety- characteristics of a system [11,12]. It is particularly well suited when the system to be conceived is unfamiliar or uses new technologies. When safety is of major concern the method is sometimes called F.M.E.C.A -Failure, Modes, Effects and Criticality Analysis. The presentation of an analysis method should include a description of the underlying model it proposes to represent the system under development and the description of the approach for the model construction activity. We will present in the following this two aspects.

The undelving model of F.M.E.A As noted above F.M.E.A is characterized by its strong informal aspect so one cannot properly talk of an underlying model of F.M.E.A. The analysis process recommended by the method results in the identification of instances of a set of concepts representing the specification of the dependability and safety requirements. These concepts are presented in a tabular form as in figure 1. However different authors may retain different sets of relevant concepts, in [15] a slightly different table is used.

<table>
<thead>
<tr>
<th>Element:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sub-system:</td>
</tr>
<tr>
<td>Function:</td>
</tr>
<tr>
<td>Failure mode</td>
</tr>
<tr>
<td>Local</td>
</tr>
</tbody>
</table>

Figure 1: Analysis results of F.M.E.A (from [12])

The concepts employed are only defined by a textual, informal description. For example in [15], the failure mode is presented as "...the manner in which a component fails." And in [12] it is defined as the answer to the question: how a functionality of the system can be affected? although it is more operational, this second definition still leaves too much room for interpretation at the time of the analysis. The absence of well defined semantics for the concepts makes the systematization of the analysis process a very hard -if not impossible- task.
The approach of F.M.E.A According to [12] the main steps of the analysis are:
- Decompose the system into subsystems and identify the functionalities of each sub-system and the flows between the subsystems
- Identify the hardware components of each subsystem and the environmental constraints on them.
- Identify failure modes by the conjoint analysis of the functional structure and the hardware structure.
- Identify the operators tasks
- Identify failure modes caused by operator's mistakes
- Identify critical phases for each failure mode, describe their consequences. Associate to the consequences a gravity coefficient \( P_i \) and associate to the failure mode a probability of occurrence \( \lambda_i \). The criticality factor \( C_i = \lambda_i * P_i \) can then be used to classify the failure modes by higher priority.
- Eventually, for each failure mode:
  * Define whether a corrective action and an alarm to trigger it are necessary.
  * Identify the failure modes associated to the same alarm and verify that they require the same corrective action
  * Identify how to identify the failure.
  * Identify what reaction are inappropriate and what are their consequences.
  * Identify which parameters would allow the maintenance staff to foresee the failure.

The description of approach that has been briefly introduced above presents many drawbacks:
- It let the reader imply that the steps occurs sequentially by not describing more precisely the control that governs the chaining of steps.
- The distinction between the steps to be executed and the objects operated upon is not made. The two notions are intertwined in the description, consequently, it is difficult to establish what has to be done and what are the expected results.
- Some objects are mentioned with no description of how their are to be obtained.

It is common to find similar flaws in many textual descriptions of methods and office procedure manuals internal to companies. The use of a natural language to describe a method will most of the time result in ambiguities and incompleteness.

The SA-RT method
The SA-RT method [16] permits the requirements analysis and functional specification of a system. It is issued from the structured analysis approach of DeMarco [17]. As it provides a formalism for the expression of control, it is more suited to real-time systems. In the following we will briefly introduce the formalism and the approach of SA-RT

The formalism of SA-RT SA-RT offers two models: the requirements model and the architecture model. We will restrict this study to the requirements model. This model is composed of four constituents:
- A process model based on Data Flow Diagrams (DFD) hierarchically structured.
- A control model based on control signals and finite state machines.
Managing Quality Systems

- A specification of the real-time constraints at the interface of the system.
- A data dictionary.

The concepts appearing in the model are precisely but informally defined in [16], we expect the process model of the method to make the definitions gain in formality.

The approach of SA-RT

The definition of an approach for a structured analysis method necessarily reflects the preferences of a group of users. The approach proposed in the following have no pretenses to universality but corresponds to our proper choices. In a sense the process model of a method captures a local expertise of use of this method.

The main component of the requirements model is the process model. It is from this model that the users can start apprehending the system. The building of this model is the central activity that orchestrate the other activities. The first task of this activity is the definition of the boundaries of the system. That is the construction of the root diagram of the DFD hierarchy. In order to do that, one may start with a draft top-level diagram where the two first levels are gathered. This draft top level will undergo a sequence of construction/validation steps until it is accepted by the protagonists of the development. Once the top-level is approved, it is separated in two levels, the signals it uses must be defined in the data dictionary and the time constraints have to be specified. After that, the refinement of the first level processes may start. This activity takes place by successive iterations on two or three levels. At the end of each iteration the obtained DFDs must be validated and the signals they use declared in the dictionary.

THE MASP/DL LANGUAGE

A software process model describes general features of a class of processes but not those features which are specific for each process. Well known ancestors of software process models are the waterfall model, and the spiral model. Software process models describe which activities have to be performed by which people, who is allowed to access defined categories of documents, when activities are to be performed, which tools will be used... A formalism suited for describing software process models is called a software process modeling language.

The software process modeling approach pursued in ALF is based on executable process modeling language. By executing software process models it is possible to ensure that the actual software process is coformable to the model. The basic structure element of our process modeling language is the MASP, which salient features are:

- **Object model**: The object model defines object types and object relations. It is based on the typed entity-relationship diagrams of PCTE [20].

- **Operator types**: An operator type consists of a name, a signature, a pre- and post conditions. The signature specifies the types of input and output parameters which are elements of the object model. pre- and post conditions are predicates expressed on objects states . The precondition must evaluate to true before execution of the operator. The postcondition is assumed to be true after execution of the operator.

- **Rules**: Rules allow the triggering of operations whenever predefined conditions occurs.
- **Orderings**: Orderings describe constraints on the order in which operations may be executed. A path expression like formalism is used for the specification of orderings.

- **Characteristics**: Characteristics are predicates specifying integrity constraints. Thus they must be true at each state of the software process. Whenever the violation of a characteristic is detected actions are taken for correcting the situation and having the characteristic to be true again. A detailed description of ALF and the MASP/DL language can be found in [21,22]

**DEFINITION OF PROCESS MODELS FOR F.M.E.A AND SA-RT**

Our aim here is not to exhibit complete models of the two methods but rather to evaluate the contribution of process modeling to the construction of critical systems. In particular we which to show that the description, by the intermediary of the MASP/DL language of a method allow to reduce the drawbacks due to it's informal aspect. In effect, it is only by making explicit the artefacts produced by a method and relating them to a precise semantic, and by precising the operations that lead to them that we can systemize the application of the methods and improve their automatisation. As the use of the process model of a method proceed, the concepts that appear in the method's formalism will get more and more precisely defined and the approach to build the system model will be described more deeply. The process model will furnish a more and more coherent and complete reference framework in which the creative activities of the method's users may take place. This framework will help to ease these activities.

There are three important facets in the construction of a method's process model: Identification of the objects produced by the method, identification of the approach to build these objects and implementation of this knowledge in a process modeling language.

Starting from the expected outcome of the method as the initial artefact, we will look for the operations that produce it, by successive decompositions, considering the method as the most abstract operation. As the exhaustivity is not part of our goals, we will push the decomposition to only two levels.

**Process model of F.M.E.A**

The first operation we have is then the one representing the whole method. F.M.E.A operation has as input the hierarchical structure of the system and the functional structure of each subsystem and it outputs a result table as defined in figure 1. The hierarchic and functional structure objects are defined by the ERA schema of figure 2.

The elicitation of the objects hierarchic and functional structure is considered to be extern to the method because this activity is complex enough to require a method for itself, and for more these objects are part of the results provided by SA-RT, we'll get back to this point in section 5 about method integration. The results table is an aggregation of the different objects obtained during the analysis, its definition consists in the definition of its components which will come next.

At this first level of decomposition we can distinguish three activities: The analysis of a subsystem, the analysis of the oprators tasks and the analysis of failure modes.
Analysis of a subsystem. This operation supply as a result the list of the failure modes associated to a subsystem. This list is the outcome of an inductive approach: Starting from one of the functionalities offered by the subsystem, one have to look for possible faulty behaviours of this function and search for the hardware or software components of the subsystem that might cause this faulty behaviour. The objects related to this operation are then: the functionalities and the components of a subsystem as input and the failure mode list as output. The meaning of function in this context is linked to the determination of the different ways it may fail. Thus a function will be defined by the specification of its abnormal behaviours as shown in figure 3.

One of the activities that is part of the analysis of a subsystem is induced by the specification link of figure 3 and will consist in the specification of the list of faulty behaviours for each function of the subsystem. Each definition of an abnormal behaviour entails the identification of a failure mode for the component that implement the function. Figure 3 shows these relationships between the objects we have identified.

From the structure of the function object we can deduce the following operations, controls and integrity constraints:
466 Software Quality Management

- Specification of a faulty behaviour
- Identification of a failure mode
- The specification of a faulty behaviour should trigger the identification of a failure mode
- Each function must be associated to a list of failure modes and must be implemented by a component.

Analysis of the operator's tasks The second source of potential failures originates from operator's mistakes. The analysis of operator tasks will allow to anticipate them. This analysis starts by the description of the tasks realised by the operator. The description will be used for the search of the possible mistakes the operator might make. The definition of the task must then be made with respect to this use. Figure 5 shows the ERA schema defining the task object.

In the ERA schema of figure 4 a task is structured in a set of elementary actions. Elementary actions are ordered by sequencing constraints and have acceptance test to validate them when they are finished. The search of failure modes is realised by the search of sequencing constraints on elementary actions violations and by validation tests violation. Clearly these are not the only sources of errors, the role of the other link is to take into account failures from other origines. The repeated use of the process model should allow for the refinement of this link.

From the schema of figure 4 we can decompose the operator tasks analysis to the following operations:
- Definition of a task structure
- Definition of sequencing constraints associated to an elementary action
- Definition of the validation test at the end of an elementary action
- Definition of a failure mode

From the schema of figure 4 we can decompose the operator tasks analysis to the following operations:

From the schema of figure 4 we can decompose the operator tasks analysis to the following operations:

- The definition of an order relation must trigger the definition of the associated failure mode.
- similarly, the definition of a validation constraint must trigger the definition of the associated failure mode.

Analysis of a failure mode Once the failure modes identified, they must be sorted according to their criticity. Criticity is defined with respect to the failure mode probability and consequences gravity. A failure mode may be associated
to an alarm triggering a corrective action, the occurrence of the failure starts the alarm. The ERA schema of figure 5 shows the structure of a failure mode.

Figure 5: structure of the failure mode object

To the object model of figure 5 are associated the three operations:
- Identification of a failure consequence.
- Definition of alarms
- Definition of a corrective action
and the following integrity constraints:
- Every failure mode with which criticality (probability * gravity) is above a certain level must be associated to an alarm.
- If two failure modes trigger the same alarm then, the fact that they require the same corrective action must be validated.

Process model of SA-RT

The artefacts produced by SA-RT are the requirements model and the architecture model. We have restricted ourselves to the requirements model. This model is made of the hierarchy of DFDs, the data dictionary and the time constraints, as shown in figure 6.

Following the approach sketched in section 2.2 and form the schema of figure 6, we can found out the following operations:
- Construction of the top level
- Validation of the top level
- Development of the model.
We will show the decomposition of the first operation.

Construction of the top level. The aim of this operation is the development of a draft top level from which the context diagram and the level one diagram will be generated. It is an essential step in the course of requirements analysis because it fixes the boundaries of the system for the rest of the development. The concepts involved in the definition of the top level object are presented in figure 7.
The functionalities of the system are represented at the highest level of abstraction by process objects. Processes communicate by signals with other processes and with the environment which is described by terminators.
The main activities performed during the build of the top level are then:
- Identification of processes
- Identification of terminators
- Identification of signals.

Once the top level is finished it must be separated into the context and level one diagrams. This operation is automatic (i.e. it doesn't require any interaction from a human user). To this fourth operation corresponds a control which states that the separation operation has to be triggered each time the build top level operation is exited.

INTEGRATION OF THE MODELS

The integration of a method into a system life cycle must answer questions like: Which activities precede, are concurrent to and follow it? What artefacts does the method import from other methods? How the objects produced by the method are to be used?...Actual development environments offer point to point integration and impose a given set of method (generally requirements specification and conception methods). This scheme of integration is too rigid because it doesn't cater for method's evolution expression [17], it doesn't allow the representation of the knowledge of a method accumulated by a group of users.

Figure 7: Structure of the Top level object.

In the following we will be concerned with the integration of F.M.E.A with SA-RT, and specifically we will describe the relationships between the object models of the two methods, the global ordering of operations and the causality links between activities triggering.
Object models relationships

The first two objects we will bring together are the hierarchical structure and functional structure of a system defined for F.M.E.A (figure 2) and the DFD hierarchy of SA-RT (figure 6). It appears that the context diagram corresponds to the system object of F.M.E.A, a subsystem in turn can be assimilated to a hierarchy of DFDs which root is a given process representing the subsystem. Figure 8 shows these relations.

![Diagram](image)

Figure 8: Integration of the hierarchical structure and the DFD hierarchy object models

The process object of SA-RT defines a logical entity which represents the transformation of the input signals into output signals, the function object of F.M.E.A also designates an entity that receives inputs and transmits outputs. At first sight the two objects appear similar, nevertheless there exists a difference: In the case of SA-RT, a process depicts an immediate logical transformation of logical input signals into logical output signals whereas for F.M.E.A a function may portray a physical component and the input/output may correspond to a flow of material. For SA-RT, the projection of the functional structure into a physical one is made at the architecture model development time. This separation between the logical and physical view of a system is one of the strong points of SA-RT, therefore it is advisable to adapt F.M.E.A accordingly. The subsystem analysis operation of F.M.E.A is the object of this adaptation: there are now two subsystem analysis activities, one at the logical level and one at the physical level. The ERA schema of figure 3 is consequently splitted in two parts as shown in figure 9.

For F.M.E.A the description of a faulty behavior is achieved using a natural language so one may ask: can we use the SA-RT formalism to specify erroneous comportment of a process? The use of "Structured English" (PSPECs) of SA-RT is certainly feasible but wouldn't bring much more. On the other hand the use of finite state machines may be more interesting since it lends itself to analysis. Faulty behaviour than, can be specified by unacceptable state spaces. Levison[8] describes how system safety can be analysed and validated when a state transition based formalism is used.

Finally the time constraint object associated to the signal object of figure 6 is a new source of failure modes. In deed [19] characterizes a real-time system by "The correctness of a real-time system depends not only on the logical results of computation, but also on the time at which the results are produced".
470 Software Quality Management

So we have to provide a new operation in the context of F.M.E.A which corresponds to the analysis of the time constraints. For this operation a failure mode is structured as in figure 10.

Control integration
The integration of the object models revealed the need for substituting one operation by two others and lead to the identification of a new one. The new operations are:
- Logical subsystem analysis
- Physical subsystem analysis
- Time constraints analysis

From the above operations and preceding operations we can deduce the following controls:
- The validation of the top-level should trigger the analysis of the time constraints.
- The processes produced at the end of each iteration of the model refinement must be linked to a subsystem.
- Whenever a subsystem specification with SA-RT is finished, the F.M.E.A operation of logical analysis must be triggered for this subsystem.

The orderings constraints on operations are:
- At the first level F.M.E.A and SA-RT may proceed in parallel.
- At the second level there is an iteration of construction and validation of the top level followed by a model development operation in parallel with repeated sequences of subsystems, operator tasks, and failure modes analyses.

Method integration to be complete must be supported by integration at the tools level. This will involve mechanisms of translation of the data manipulated by

![Diagram of control integration](Image)

**Figure 9:** Adaptation of the Subsystem analysis operation object model. (a) At the logical level a process is associated to a list of potential failure modes (b) When the architectural model is refined the potential failure modes are filtered to retain the ones that concern the component selected to implement the process.
the tools according to the ERA schemas that has been defined and invocation of tools functionalities according to the control we have defined [23].

CONCLUSIONS

Our aims in this paper were to show that the use of process modeling may alleviate the drawbacks due to informal aspects of analysis methods. The contributions of process modeling are in: formalisation of concepts, automatisation and systematisation of the methods, systems dependability improvement. In the following we will assess the contribution of the MASP/DL models to each aspect.

Concepts formalisation The semantics of concepts appearing in a method are expressed by ERA schemas for its static aspects and by characteristics and operator models for its dynamic aspects. The definition of a method process models oblige users to identify the key concepts of the method and their relationships which in turn allow the recognition of operations on objects, preconditions and characteristics.

The identification of concepts instances is still a human process but now users can refer to the ERA schemas to check that the instance fits the model. Moreover, operators definition render explicit deduction links between objects. However the definition of ERA schema where the semantic of concepts are expressed is a difficult task because it must capture all the different semantics of an object and this completeness is hard to achieve. This may result in models that are too contraindient. There must be a right compromise between discipline and creativity in the process model of a method.

Methods integration It's probably for this aspect that process modeling provides the major contribution. Difficulties of methods integration arise from the fact that each method have it's own formalism. So comparisons cannot easily be made and relations are difficult to identify. Therefore process modeling languages can be seen as the common formalism to which methods formalisms are translated.

Two methods will be fully integrated when all the relevant relations between their concepts are identified. This can only be done by adapting the integrated model in use to the new expressed needs and iterating it.

Automatisation and systematisation of a method The control mechanisms of MASP/DL (i.e. ordering and rules) are the means for methods systematisation.
At any moment there is a restricted set of actions a method user can take. The user is subordinated to the approach defined by the control aspects of the model. But again this may result in models that are too contraind, process modeling must be aware of the necessity of compromise between discipline and creativity.

Actually the automatisation of methods takes the form of editing and document generation facilities and some validation capabilities (ie. balancing checks for SA-RT). To improve the automatisation of a method we shouldn’t expect a small number of high granularity tools e.g. the equivalent of compilers or linkers tools but rather search for a large number of small tools.

Specification activities requires a lot of interaction. It is only by recursively decomposing this activities that we can reach less abstract processes that do not require human interaction and might be totally automated. The MASP structuring capabilities encourage the definition of software process at different levels of abstraction and the assistance mechanisms support automatisation by taking initiatives, see [20] for more information on MASPs guidance and assistance mechanisms.

**Systems dependability improvement** The contribution of an ALF based SDE to system dependability improvement is directly related to the enforcement of discipline capabilities of this environment. Examples of what this environment can guarantee are:
- activities are undertaken by qualified persons only, this allow to ensure that all the tricky task with major implication on system safety are carried on by people with adequate competence.
- all instances of a concept are the submitted to a set of activities, for example all the instances of the Failure mode object of SA-RT are submitted to the failure mode analysis operator. This allow to ensure a form of completeness in the analysis process.

**REFERENCES**

23. O. Perrin and N. Boujdilida "Towards a Model for Persistent Data Integration" LNCS 685. June 93.