PRIDE: An Integrated Software Development Environment for Dependable Systems

Silvia Mazzini
Intecs,
Via Umberto Forti Trav. A, n.5
Polo di Attività Montacchiello, Loc. Ospedaletto
I-56121 Pisa, Italy
Silvia.Mazzini@pisa.intecs.it

Diego Latella, Daniela Viva
ISTI-CNR
Area della Ricerca di Pisa
Via G. Moruzzi, 1 - I56124 Pisa, ITALY
Daniela.Viva@isti.cnr.it
Diego.Latella@isti.cnr.it

Abstract
This paper presents the PRIDE research project, developed by Intecs and co-funded by the Italian Space Agency. The project aims at providing an integrated software development environment for dependable systems based on UML, which integrates design activities with selected methods for formal verification and validation (V&V) and quantitative dependability attributes assessment, supported by appropriate tools. The PRIDE environment is built as an extension of the already existing environment for the modeling of Hard Real Time systems, HRT-UML, based on the Unified Modeling Language (UML) notation.

The outcome is an advanced and integrated methodological solution for the design of complex embedded real-time systems and for their evaluation and verification, according to rigorous techniques based on formal theories, such as formal verification, model based dependability evaluation and schedulability analysis.

1. Introduction

The quality of a complex system design is strongly dependent on the quality of the process in which it is inserted and on the tools supporting such a process; in particular an effective design process requires an early validation of the concepts and architectural choices.

Design for dependability is actually a challenging field where the ongoing research initiatives are trying to set new patterns. In fact the growing of the system complexity has shown the limit of the traditional validation practices typically used in industry to deal with these systems’ dependability requirements.

The approach is to assist the designer by offering an appropriate design methodology, and a support toolset, that integrates a consistent set of dependability validation techniques based on formal means, well established quantitative dependability attributes assessment techniques, automatic translations from the design to the mathematical models and the analysis tools that are most appropriate for the different kinds of validation. Following this approach, the designer can early and easily validate the design since the bulk of the validation technicalities are hidden.

The design methodology is based on the Unified Modeling Language (UML) notation, which offers profiling extension mechanisms, making it an open and extensible framework for the design of complex system. Since UML today does not provide elements that are...
suitable to model dependability aspects, an appropriate extension has been defined for this purpose. Dependability evaluation in PRIDE is driven by the two following approaches:

- **Formal Verification** of system behavioral properties;
- **Quantitative Validation** of system dependability attributes.

Having both formal and quantitative aspects of the critical part of the system within the very same notation sets a big step toward filling the gap that usually separates formal verification from quantitative validation.

### 2 Formal Verification

Proved functional correctness is of paramount importance in the design and development of critical systems. By proved functional correctness we mean that not only confidence on functional correctness of a system is built by means of best practice in its development, but that, moreover, system correctness w.r.t. crucial functional properties is proved using sound mathematical and logical techniques, possibly supported by reliable automatic support tools.

Proved functional correctness is essential for those complex systems which support human activities which are crucial for the well functioning of our societies, and for the safety of living-beings and their environment. Examples of such systems are energy-production plants and automotive or aerospace control systems.

Formal Methods provide the techniques, methodologies and tools for producing such proofs and consequently for designing proved correct systems. Of course, the use of Formal Methods introduces non-trivial costs, in terms of additional training, specific tool support, formal specification development time, and related verification effort. Such costs can be justified only when assessed in relation to the criticality of the system components to which Formal Methods are applied. Consequently, the first activity to be performed in the development of any critical system is the identification of the most critical subcomponents as candidates for the use of formal modeling and verification techniques.

In the context of the PRIDE Project, a specific technology has been chosen as the main support technology for the application of Formal Methods, namely Model-Checking [CGP00].

Model-Checking techniques allow for the automatic verification that a specific model of the overall behavior of a system (critical component) satisfies a set of requirements. The model is specified by means of a formal language, which can be textual or graphical. Typically, behaviors are modeled as state-transition systems. Each requirement is usually specified as a temporal logic formula [Eme90].

A peculiarity of the model-checking approach, when compared with other techniques (like e.g. semi-automatic theorem proving), is the user-friendliness of the tools, which support the verification activity. They fall in the "push-button" category: once a model has been developed and a temporal logic formula characterizing a critical desired property of the model has been formulated, the verification of whether the model satisfies the formula is performed by the tool in a completely automatic way, i.e. without any further action required to the user. If the formula is not satisfied, the tool usually provides a counter-example in the form of a sequence of computation steps, which bring to a violation of the desired property.
Such marked user-friendliness is the main reason why model-checkers are so successful in the context of formal validation and is the main motivation why we chose a model-checking approach in the context of PRIDE.

Moreover, we increased user-friendliness by providing the PRIDE user with an UML-native notation for building its models, namely Statechart Diagrams. Similarly, we defined a simple linear time temporal logic for specifying correctness requirements, which builds upon the OCL, which the typical UML user should be familiar with.

Our target verification tool was SPIN [Hol03]. SPIN is a model-checker, which combines efficient algorithms for the validation of complex models with an easy to use user interface. The latter includes a graphical animation system for Message Sequence Charts used to report counter-examples. SPIN is capable of easily performing exhaustive verifications of systems with tens of million states. In exhaustive verifications each and every state of the model is visited, as well as all possible computation paths, if necessary, in order to check the requirement. This is possible due to advanced compression techniques used for representing the model state space.

Moreover, when the state space is too large for an exhaustive analysis, SPIN can perform a random analysis of the model—the so called bit-state analysis—which can guarantee high coverage for state spaces the sizes of which can grow up to the number of single bits available in the actual RAM of the machine where the verification is performed. Moreover PROMELA, the modeling language used in SPIN, is very similar to the C language, which makes it easy to learn also in industrial environments, where indeed the tool is widely used nowadays.

2.1 From UML to a PROMELA model

A necessary requirement for the development of any formal method support tool is that the notation of interest has a formal semantics. In the case of PRIDE we based our work on the proposal presented in [LMM99] for the formal operational semantics of UML statecharts. Essentially, the semantics of a statechart is a transition system each state of which corresponds to one state of the underlying state-machine and the current content of its input-queue. A generic transition of such a transition system corresponds to a STEP transition of the statechart. We extended the semantics proposed in the above mentioned work in several ways in order to cover all aspects of the subset of UML statecharts used in PRIDE. In particular we added the possibility of using object state variables and we allowed a system model to be composed by a collection of statecharts, each associated to a specific class (and modeling the behavior of the generic object of such a class) and provided with its input queue.

A translation has been formally defined which takes a model consisting of a collection of statecharts as input and generates a PROMELA model to be used as an input for SPIN.

The translation is based on previous work of Latella at al. [LMM00] and extends it in order to cover collections of statecharts and use of state variables. In particular, a distinct PROMELA process is generated for each statechart, and input-queues are mapped into PROMELA FIFO queues. The proof of correctness of the translation follows the same principles as those used in [LMM00], with the necessary extensions. Essentially, a formal relation is established between, on the one end, the states of the transition system associated to the collection of statecharts by its formal semantics, and, on the other end, the states of the PROMELA code. Then for each computation of the (collection of)
statechart(s) leading to a certain state \( S \) of the transition system it is shown that a computation exists in the execution of the PROMELA code leading to a state corresponding to \( S \) according to the formal relation mentioned above, and vice-versa.

Moreover, a simple temporal logic has been defined by customizing that used by SPIN with features specific to UML statecharts. In particular, OCL-based atomic predicates have been embedded in the logic. The requirements can be expressed by this logic are those typical of linear-time temporal logics (i.e. safety, liveness and fairness properties). For instance, the requirement that whenever the value of variable \( x \) of object \( Y \) is \( v \), object \( Y \) will eventually reach state \( Z \) will be formalized as

\[
[](Y.x==v \implies <>Y.OclInState(Z))
\]

where \( [] \) and \( <> \) are the usual 'always' and 'eventually' operators of linear time temporal logic; informally \( []f \) (resp. \( <>f \)) means that \( f \) holds in every state (resp. in some state) and \( f1 U f2 \) means that there exists a state in which \( f2 \) holds and \( f1 \) must hold in all states up to such a state.

The PRIDE environment provides all the tools necessary for the definition and maintenance of the collection of statecharts associated to the class diagrams for a system model and an easy to use interface for the definition of the system requirements using the above sketched temporal logic. Moreover, the interface provides support for the definition of the run-time configuration parameters necessary for the verification, e.g. the total amount of RAM available for the verification and the specific verification algorithm one is interested in (e.g. exhaustive, bit-state, etc.). PRIDE automatically translates the statechart(s) and the requirements into PROMELA and proper logic formulas and run SPIN on them, with proper run time setting, reporting to the user the result of the verification.

The main advantage of the approach is that all the details of the PROMELA modeling language are hidden to the UML modeler who can focus on her/his statechart model, still using the power of SPIN. Another advantage comes from the particular approach we have chosen, namely a transformational one; by developing a translation from statecharts to PROMELA we do not need to define and implement an ad hoc model-checker for UML statecharts and in principle we could define a similar translation from other UML diagrams to PROMELA, thus re-using SPIN technology. Of course, the price we had to pay for such an advantage is efficiency; as a parallel research activity, we also developed an ad hoc model-checker prototype directly and specifically for UML statecharts and, although the performance of the PRIDE/SPIN model-checker was good for all the case studies we developed, as expected, we also had indications that better performance results can be achieved using ad hoc techniques. As usual, the final assessment can only be given after extensive experimentation and as a trade-off between efficiency vs. generality.

3 Quantitative Validation

Quantitative Validation of System Dependability Attributes aims at building a dependability model based on Stochastic Activity Networks (SAN) [MMS85], which concentrates on the structure of the system and takes information from the structural UML diagrams. A dependability model is a mathematical model that characterizes a system with a certain level of abstraction. The following general parts are included in a dependability model: the fault activation processes which model the fault occurrence in system elements and result in basic events, the propagation processes which model the consequences of basic events.
and result in derived failure events, and the \textit{repair processes} which model how basic or derived events are removed from the system. When redundancy is employed in the system for fault tolerance purposes, the model has also to capture the dynamics of such redundant structures. A dependability model is strictly tied to the measure of system behavior to be computed. According to that measure, in the development of the model, some of the processes or events just mentioned may be neglected.

The proposed approach allows for a less detailed but system wide representation of the dependability characteristics of the systems and offers a significant advantage in terms of controlling the size of the models. Furthermore, it represents a means to analyzed dependability measures of a system while it is still being designed. This way, preliminary evaluations of the system dependability during the early phases of the design can be provided, which show very helpful in identifying possible deficiencies in the design and as guidelines for design refinements.

SAN models for quantitative evaluation of dependability measures are analyzed using Mobius [D&al00]. Mobius is a modeling software tool developed by the University of Illinois. It provides a powerful automatic infrastructure to support multiple interacting formalisms and solvers, and is extensible in that new formalisms and solvers can be added to the tool without changing those already existing.

3.1 From UML to a SAN model

The quantitative transformation focuses on the early, system-level dependability modeling based on high-level structural descriptions. The transformation maps UML structural diagrams to SAN models, following an approach already proposed in [BMM99a], [BMM99b], [BDLPP99].

In defining the transformation, the UML structural diagrams considered are all those available in the standard UML Notation: use case, class, object, component, and deployment diagrams. Moreover, statechart diagrams are taken into account, to deal with the management of redundant resources. In general, nowadays tools for UML system design don't support all these diagrams; therefore it could be necessary to revise the sources of the dependability related information if the specific used tool does not support all the mentioned diagrams.

The transformation is defined in two steps. The first one has the fundamental task of extracting the relevant dependability information from the mass of information available in the UML description. In this step, an Intermediate Model (IM) is built, in which we can fix the set of basic and derived failure events, the fault activation, propagation and repair processes. In a sense, the dependability model is built in this step. Then, in the next step, the translation from the IM to the SAN model is defined.

The IM is a hyper graph, where each node represents an element described in the UML diagrams, and each hyper arc represents a relation between elements. During this first step of the transformation, UML model elements (objects, nodes, packages, and components) are mapped to simple elements, while redundancy structures are mapped to composite elements of the IM. Hardware and software elements, with or without internal states, are distinguished by the type of the element (the distinction is important from the point of view of errors and error propagation). A special node, called SYS, is introduced to represent the components of the system whose dependability attributes are the object of the evaluation. In particular, a SYS node may represent the entire system, or any UML
entity about which the designer is interested in estimating the dependability figures. UML relations (associations, compositions, dependencies, deployments) are mapped to “uses service of” and “composed of” relations of the IM. Failure of a composite element, i.e. a redundancy structure, is not just an OR relation of the failures of the simple elements belonging to the given composite one. Usually, a fault tree can be used to describe the effects of separate and common mode failures of the elements. In our approach, it is derived by analyzing the behavioral description of the redundancy structure, which is provided by the designer in the form of an annotated UML statechart diagram.

The second step of our transformation builds the SAN dependability model, by generating a SAN for each element of the IM. The SANs include well-defined set of places and transitions that are interfaces towards other SANs of the model, through a mechanism said "common places". According to the above described structure and parameters of the IM, the target SAN model consists of the following types of SANs:

- Basic SANs, which represent the internal state of the UML element corresponding to each node of the IM, model the failure processes, and the repair processes. Two different kinds of repair are modeled as a consequence of transient and permanent faults.
- Propagation SANs, which are generated by examining the “uses the service of” hyper-arcs of the IM. They link basic SANs modeling the propagation of a failure occurring in a node to other node(s) with consequent corruption of their internal state.

Properly translating the nodes and hyper-arcs of the IM model in the corresponding SANs, and composing all together the resulting SANs, the overall SAN model is obtained. Then, the dependability measure to be evaluated, given as attributes of the SYS node of the IM, is translated according to the syntax of the Mobius tool. The results, obtained by processing the dependability measure selected, are managed by the designer as part of the documentation and may be compared with the results obtained in successive refinements of the analysis.

4 The PRIDE UML Profile

Since the standard UML notation does not cover all aspects required for the analysis and evaluation of dependability aspects during system design, a PRIDE extension profile is defined for qualitative and quantitative analysis, based on the standard UML mechanisms, such as stereotypes, tagged values and constraints.

4.1 UML profiling for qualitative analysis

In the context of PRIDE the model of system or subsystem behavior is represented by a collection of UML Statecharts asynchronously communicating via their input queues. Each Statechart models the behavior of the object to which is uniquely associated.

The specific class of Statecharts, which we consider in PRIDE, is obtained from UML Statecharts by means of constraints on the specific elements of the notation to use and the extensions of specific features.

In the following we provide the list of both the constraints and the extensions:

- **Creation and Deletion:** system is modeled by a static set of Statecharts, no creation destruction of behaviors is allowed.
- **States:** shallow history, deep history states as well as action states, choices and junction are not allowed.
- **Events:** events are restricted to signal and (asynchronous) call events, synchronous calls should be modeled by explicit wait states.
- **Transitions:** completion transitions are not allowed.
- **States internal actions:** they are not allowed.
- **Data values and variables:** the following basic data types can be used in the statecharts models: bool, byte, short, int where byte, short and int are subsets of the integers.
- **General correctness requirements** relevant to qualitative analysis are expressed as formulas of Temporal Logics, they are concerned mainly with linear time temporal logics.

### 4.2 UML profiling for quantitative analysis

The quantitative analysis profile provides the UML extensions enabling the designer to provide dependability related parameters, to select desired dependability measures, to specify fault occurrence structures to be included in the system, and the transformation tool to construct the dependability model, i.e. define the basic and derived events, propagation, failure and repair processes.

A summary of UML extensions for quantitative analysis is provided in the following:

- **Assignment of dependability related parameters to UML elements:** software (UML Classes/Objects, Components) and hardware elements (UML Nodes) are distinguished with **state full** and **stateless** stereotypes according to the presence or not of internal state. The dependability related parameters (such as fault occurrence, error latency, repair delay, etc.) are expressed as tagged values applied to stereotypes. Stereotyped relations indicating error propagation paths are assigned propagation-related parameters.

- **Identifying redundancy (fault tolerance) structures:** redundancy structures are composed of objects stereotyped as follows:
  - **Redundancy manager**
  - **Variant**
  - **Adjudicator**

- **Identifying states and events in state charts of redundancy managers:** in order to derive the non-trivial relations in redundancy structures, the state charts of the redundancy manager have to be analyzed. This analysis is supported by stereotyping states and events in the state chart as follows:
  - **Stereotype Failure** of a state indicated that it is an explicit failure state
  - **Stereotype Failure Notification** of an event indicates that it is an explicit failure notification towards the client(s)
  - **Stereotype Response** of an event indicates that it is a normal response of the object toward the client(s).

- **Definition of the reward structure:** a failure condition is a boolean expression on the set of objects collaborating in a use case: it determines the partition between proper and improper states of the objects implementing a system service.

- **System dependability requirements:** the measure to evaluate (reliability, availability, etc.) is given in UML as a structured tag value, associated to the project.

### 5 The PRIDE Transformation Framework

PRIDE contributes to the OMG Model Driven Architecture (MDA) approach to software development: automatic translations are applied to a model in order to generate new models that enable verification and validation of functional and non functional
requirements; this models are specific for the V&V platform, we call them V&V Platform Specific Models. The MDA applied pattern is a new pattern where a model transformation is not applied serially to derive a PSM that can be further detailed to produce an implementation, but finalized to a specific development process activity.

Figure 1 describes the transformational framework. Tools support is provided so that when the user requests a qualitative or quantitative analysis, automatically the following sequence is performed:

- ask the user for the tool required run time settings
- save the appropriate model in XML format,
- apply the transformation on the model, by processing the XMI files against XSL templates; the result of such processing are the PROMELA or SUN files.
- run the appropriate tool on the resulting files, with the user selected settings,
- report the result of the verification to the user.

6 Building on top of HRT-UML

The PRIDE design methodology, and the support toolset, is based on the extension of the HRT-UML method, developed by Intecs in the context of a previous research project, co-funded by ASI. HRT-UML is a customized version of UML aimed at providing a comprehensive solution to the modeling of Hard Real Time systems, in the context of a previous research project, co-funded by ASI [[DMDL02], [M&al02], [DMD03],

HRT-UML incorporates and improves the principles of HRT-HOOD, a method developed in the early 90’ for mission critical aerospace applications, recognized in his “niche” domain as the reference methodology for the development of time-critical missions.

7 The HRT-UML Core Method and its Extensions

The core of the HRT-UML method and toolset is an UML extension profile, which we call HRT-UML Core. The HRT-UML Core offers the basis for ad-hoc profiles concerning hard-real time (HRT profile) and dependable system (PRIDE, or Dependability profile).
These two profiles enable the user to deal with the main concepts arising from this kind of systems, concepts that can be easily managed and traced during the development process, from the design until the verification or evaluation phase. They are orthogonal, in the sense that there is no concept of one profile that brings limitation on the other one. Given an entity in the tool, properties (stereotype in UML) of one or both the profile can be assigned to it independently.

As shown in Figure 2, the HRT-UML Core method may host any further extension, in addition to the identified ones.

![HRT-UML Core](image)

**Figure 2 – The HRT-UML Core and its Extensions**

8 Conclusions

Dependability modeling and evaluation is a mature technology that needs to be integrated in the design activities in order to cost-effectively impact the development process. The PRIDE philosophy is to help out the designer by offering an UML based methodology that provide translations from the UML notation to the mathematical models suitable for different validation types. Accordingly the designer will be able to validate specific design dependability features, since automatic translations into the individual mathematical models for the validation deal with technicalities.

An integrated software development environment for complex embedded real-time systems supporting appropriate design methods is progressively defined and complemented with sophisticated validation techniques, such as formal verification, model based dependability evaluation and schedulability analysis.

9 Acknowledgments

The authors wish to thank for their contribution and support, in terms of suggestions and criticisms, all the people involved in discussions during the course of the project. In particular, they wish to thank E. Russo (ASI), A. Di Cesare (ASI), G. Holzmann (NASA), A. Bondavalli (Univ. Firenze), F. Di Giandomenico (ISTI-CNR), S. Gnesi (ISTI-CNR), M. Massink (ISTI-CNR), Franco Mazzanti (ISTI-CNR), and the Intecs team.

10 References


