HIERARCHICAL, OBJECT-ORIENTED MODELING OF FAULT-TOLERANT COMPUTER SYSTEMS

Juan A. Carrasco*
Departament d'Enginyería Electrònica, UPC
Diagonal 647, plta. 9
08028-Barcelona, Spain

Abstract

A hierarchical, object-oriented modeling language for the specification of dependability models for complex fault-tolerant computer systems is overviewed. The language incorporates the hierarchical notions of cluster, operational mode and configuration and borrows from object-oriented programming the concepts of class, parameterization, and instantiation. These features together result in a highly expressive environment allowing the concise specification of sophisticated dependability models for complex systems. In addition, the language supports the declaration of symmetries that systems may exhibit at levels higher than the component level. These symmetries can be used to automatically generate lumped state-level models of significantly reduced size in relation to the state-level models which would be generated from a flat, component-level description of the system.

1. INTRODUCTION

A fault-tolerant computer system can often be conceptualized as made up of components changing their state as a result of failure, recovery, and repair processes. Dependencies among components often arise in real fault-tolerant computer systems due to failure propagation, limited recovery efficiency, reconfigurations, and repair. Stochastic processes allow to consider all these important details. Recovery processes are typically several orders of magnitude faster than both failure and repair processes and can be modelled by using instantaneous coverage probabilities [1], which can be obtained by statistical analysis [2] of experimental data collected from fault-injection experiments. Typically, failure and repair times are assumed to have exponential distributions so that the stochastic behavior of the system can be described by a continuous time Markov chain (CTMC) having failure and repair transitions. In general, dependencies among the behavior of the components of the system are such that the CTMC has to be generated and solved using general-purpose, state-level methods. Except for systems with an small number of components, the CTMC has a size (number of states) such that its direct specification is unpractical and automatic generation from a higher level specification is required.

Available model specification methodologies show a tradeoff between modeling power and user-friendliness. We have in one hand very general specification methodologies like Stochastic Petri nets [3, 4] and Production rules [5], which allow the specification of arbitrary CTMC dependability models but require complex and thus error-prone descriptions for large systems. Another approach [6, 7] is the use of a special-purpose language with an implicit modeling point of view and special constructs easing the specification of often-encountered complex dependencies. In addition to being more user-friendly, the use of a high-level modeling language has the advantage of conveying semantic knowledge which can be exploited during model solution [8, 9, 10].

This paper overviews a hierarchical, object-oriented modeling language for fault-tolerant computer systems which takes the SAVE modeling language [6, 7] as starting point, but introduces many extensions (clusters, operational modes, configurations) enhancing significantly its modeling power and user-friendliness and supporting automatic generation of state-level models of reduced size when, as it is often the case, the system has symmetries at levels higher than the component level. The latter is important since it is the size of the CTMC what ultimately limits the application of numerical solution methods. The rest of the paper is organized as follows. Section 2 overviews the language using a example of moderate complexity (a detailed description of the language syntax and semantics and additional examples can be found in [11]). Section 3 discusses the representation of the states of the lumped CTMC's . Section 4 concludes the paper.

2. LANGUAGE OVERVIEW

2.1. Modeling framework

In our language, a fault-tolerant computer system is conceptualized as made up of components and repair teams. Component states are modified by failure and repair processes. Failure processes only affect unfailed components and are associated with particular components of the system but, in general, can be propagated to other components. Repair processes are performed by repair teams. The system is either operational or down, as determined by the unfailed/failed condition of the components of the system through a coherent structure function [12]. Lack of coverage can be modeled in our language by failure propagation. This approach is less restrictive than it seems at first glance. For instance, the inefficiency of system-level recovery procedures can be modelled by introducing a "recovery" component without failure processes which is required to be unfailed for the system to be operational, and propagating uncovered failures to the "recovery" component. The repair of the "recovery" component would model the restart action usually undertaken after that type of system failures.

*This work was supported by the ESPRIT project of the Commission of the European Communities no. 1609 "SMART: System Measurement and Architecture Techniques".

Lack of coverage taking down only part of the system can be modeled similarly.

A component can be operational, failed, or dormant. As in the SAVE modeling language [6, 7], the failed state can be refined by failed modes, which are determined at the time the failure of the component occurs. In addition, in our language, the operational state can be refined by operational modes. Operational modes provide the basis for modeling differences in the failure processes affecting a component which may result from the configuration in which the system is working, and are an useful generalization expanding significantly the modeling power of the language. We also introduce in our language the concept of cluster. Conceptually, a cluster is a collection of components which at a certain abstraction level can be seen as a whole. Clusters can have operational modes and configurations. Configurations are also defined at the system level. A configuration defines a mapping of components and clusters instantiated at a given level into operational modes. This allows the concise specification of complex configuration strategies by exploiting the hierarchical structure of the system.

The behavior of components and clusters is described in parametrizable classes which can be instantiated. A collection of objects can be instantiated with the same name. The behavior of the objects with the same name is undistinguishable and this information is exploited for the generation of lumped models whose states are defined by factorizing instances with the same name and in the same state. By using clusters, symmetries that the system may have at levels higher than the component level (which would be disregarded if a flat, component-level description were used) can be made explicit and exploited.

2.2. An example

The constructs of the language will be illustrated through an example of moderate complexity. The example is a distributed fault-tolerant database system with 4 sites. Each site contains two processors and two databases, keeping two copies of the data. Access to data is performed through two front-ends. The sites and the front-ends are connected by two local-area networks (LAN's) as shown in Figure 1. The system is operational if each site has at least one unfailed processor and one unfailed database, and at least one front-end and the LAN to which the front-end is connected are unfailed. When the system is down all the components are dormant. In addition, when a front-end is failed the corresponding LAN is dormant and viceversa. Dormant components do not fail.

Two failed modes are considered for the processors: one requiring repair (hard failure) and another requiring only restart (soft failure). The processor soft failure rate is higher when the site has only one operational processor, since in this configuration the processor has a higher load. Databases have also two failed modes: one requiring repair (hard failure) and another requiring only data recovery (soft failure). In order to keep the copies of the data consistent, write accesses are performed in parallel to all the operational databases of a site. A processor failure contaminates one (and only one) operational database with a probability which depends on the type of failure (hard or soft) of the processor. A contaminated database is in soft failed mode. Data recovery for databases in soft failure can be done in two modes. The first mode (restoring) is possible if the the other database and at least one processor of the site are operational and the site is accessible from at least one operational front-end. When restoring is not possible the database has to be recovered by a more time-consuming reloading operation. Processor restarts also require the site to be available from at least one front-end.

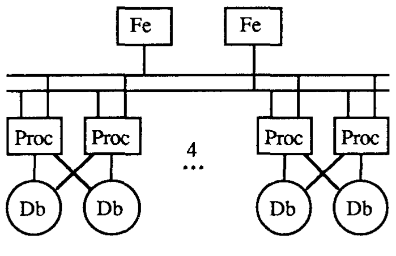

Figure 1: A distributed fault-tolerant database system.

Two field engineers repair databases, front-ends, LAN's, and processors, with the higher priority given to databases, next to front-ends and LAN's, and next to processors. Processor restarts and database recoveries are carried out by an unlimited number of operators.

2.3. Model constructs

In our language, a model is described by a set of parameters and system, cluster class, component class, and repair team descriptions. The PARAMETERS construct for the distributed fault-tolerant database system is given below. Each parameter can be assigned a default value to be used for CTMC generation if a value is not specified for it.

```
PARAMETERS
   fefr:   1/3600   /* front-end failure rate */
   lfr:    1/20000  /* LAN failure rate */
   phfr:   1/800    /* processor hard failure rate */
   pdsfr:  1/200    /* proc. soft f. rate in duplex */
   pssfr:  1/100    /* proc. soft f. rate in simplex */
   dbhfr:  1/3600   /* database hard failure rate */
   dbsfr:  1/1200   /* database soft failure rate */
   hcov:            /* cov. to proc. hard failures */
   scov:            /* cov. to proc. soft failures */
   ferr:   1/10     /* front-end repair rate */
   lrr:    1/20     /* LAN repair rate */
   prepr:  1/5      /* processor repair rate */
   presr:  1/0.05   /* processor restart rate */
   dbrepr: 1/20     /* database repair rate */
   dbresr: 1/0.2    /* database restore rate */
   dbrelr: 1        /* database reload rate */
```

In our language, a system is described hierarchically as a set of parameterized instantiations of cluster and component classes. This is done by MADE OF constructs included in the SYSTEM and CLUSTER CLASS constructs. In addition, the SYSTEM construct includes an optional list of resource attributes, either an operational or a down expression, and configurations. Resource attributes are logical variables summarizing the availability of unfailed resources instantiated at the system level and are defined by logical expressions with atoms of the forms *component*[*integer*] and *cluster.res-att*[*integer*], where *component* (*cluster*) is a component (cluster) class or name instantiated at the current (system) level and *res-att* is a resource attribute of the cluster class. These atoms are true when at least *integer* instances are unfailed or have the resource attribute true, respectively. Previously defined resource attributes can also be used. Resource attributes can be used in the construction of the operational/down expression and other expressions included in the description of configurations.

The conditions under which the system is operational are described by either an operational expression or a down expression. Both are logical expressions with the same syntax as resource attribute expressions. The operational expression evaluates to true when the system is operational and its atoms have the same semantics as in resource attribute expressions. A down expression evaluates to true when the system is down and its atoms have reversed semantics (at least *integer* instances are failed or have the resource attribute false, respectively). Resource attribute, operational, and down expressions have to be built using only the logical *and*, *or* operators. This restriction is imposed to guarantee that the structure function of the system is coherent [12].

A configuration is described by a list of directives mapping unfailed component instances and cluster instances into operational modes. Configurations may have requirements and are tried in the order they appear so that, in a given state, the first one whose requirements are met is used. Thus, semantically, a list of configurations defines a configuration strategy at a given level. Unconfigured (unmapped) component and cluster instances become dormant. A dormant cluster has all its clusters and unfailed components dormant. This implicit behavior is followed by many systems and, thus, turns out to be useful.

The SYSTEM construct for the distributed fault-tolerant database system is:

```
SYSTEM
    MADE OF
        2 Channel of ChannelC
        4 Site of SiteC
    OPERATIONAL IF: Channel.Up[1] and Site.Up[4]
    CONFIGURATION
        CLUSTERS: Channel.Up, Site.Up
            operational: all
```

specifying that the system includes two instances with name Channel of the cluster class ChannelC and four instances with name Site of the cluster class SiteC, and that the system is operational if at least one cluster Channel and all clusters Site have their resource attribute Up true. A configuration mapping all Up clusters into operational is included. This is necessary because, by default, unmapped clusters are dormant. The keyword operational is used because the cluster classes do not have operational modes.

A cluster is a set of components seen as a complex entity which can be configured and can configure the components included in it. The cluster concept is hierarchical, i.e., a cluster can be defined in terms of lower level clusters. As components, clusters are seen as instances of classes. The use of clusters makes the specification of the model more concise and allows to exploit symmetries which the system may exhibit at levels higher than the component level. This is the case in the distributed fault-tolerant database system and two cluster classes are included in the specification of the model. The cluster class ChannelC includes one front-end and the LAN connected to it and is Up when both components are unfailed. The cluster class SiteC includes the processors and databases of a site. The description of SiteC is:

```
CLUSTER CLASS: SiteC
    MADE OF
        2 Proc of ProcC
        2 Db of DbC
    RESOURCE ATTRIBUTES
        Up: Proc[1] and Db[1]
    CONFIGURATION
        REQUIREMENTS: Proc[2]
        COMPONENTS: Proc
            Duplex: 2
        COMPONENTS: Db
            operational: all
    CONFIGURATION
        COMPONENTS: Proc
            Simplex: 1
        COMPONENTS: Db
            operational: all
```

The resource attribute Up is true when at least one processor and one database are unfailed. It is possible, in general, to define several resource attributes for a cluster class. The cluster class SiteC has two configurations. The first one is used when all its processors are unfailed. The second one is used when only one processor is unfailed. The first configuration maps the two unfailed processors into the operational mode Duplex and all unfailed databases into operational. The second configuration maps the unfailed processor into the operational mode Simplex and all unfailed databases into operational. Cluster or component classes not having operational modes have to be mapped into operational if they are not to become dormant. When the cluster class has operational modes each configuration has to be associated to an operational mode by using the construct CONFIGURATION FOR: *op-mode.*

The behavior of components is also described in classes. The description of a component class includes, in its more general form, lists of parameters, operational modes and failed modes. and descriptions of failure modes and repair modes. Parameters are very useful in practice. For instance, the behavior of front-ends and LAN's, which is qualitatively identical, is described in the example by the following component class with parameters defining the failure and repair rates:

```
COMPONENT CLASS: SimpleC
    PARAMETERS: failr, repr
    FAILURE MODE
        RATE: failr
    REPAIR MODE
        RATE: repr
        REPAIR TEAM: Fieldengineers
```

The values of the parameters are defined when the components are instantiated. Thus, a front-end would be instantiated as SimpleC(fefr, ferr).

The general form of a component class construct will be illustrated by the description of the component class ProcC of the distributed fault-tolerant database:

```
COMPONENT CLASS: ProcC
    OPERATIONAL MODES: Duplex, Simplex
    FAILED MODES: Hf, Sf
    FAILURE MODE
        FAILED MODE: Hf
        RATE: phfr
        PROPAGATION MODE
            PROBABILITY: 1-hcov
            PROPAGATION EVENT
                COMPONENTS: Db
                NUMBER: 1
                FAILED MODE: Sf
    FAILURE MODE FROM: Duplex
        FAILED MODE: Sf
        RATE: pdsfr
        PROPAGATION MODE
            PROBABILITY: 1-scov
            PROPAGATION EVENT
                COMPONENTS: Db
                NUMBER: 1
                FAILED MODE: Sf
    FAILURE MODE FROM: Simplex
        FAILED MODE: Sf
        RATE: pssfr
        PROPAGATION MODE
            PROBABILITY: 1-scov
            PROPAGATION EVENT
                COMPONENTS: Db
                NUMBER: 1
                FAILED MODE: Sf
    REPAIR MODE FROM: Hf
        RATE: prepr
        REPAIR TEAM: Fieldengineers
    REPAIR MODE FROM: Sf
        DEPENDS UPON: /Channel[1]
        RATE: presr
        REPAIR TEAM: Operators
```

The component class `ProcC` has three failure modes. The first models hard failures and its active in any operational mode. The second and third failure modes model soft failures in, respectively, the `Duplex` and `Simplex` operational modes. In general, a list of unfailed component states, including operational modes and the keywords `operational` and `dormant`, can be given as the argument of the `FAILURE MODE FROM` construct. The description of a failure mode includes the mode in which the component is failed, which can only be omitted if the component class does not have failed modes, the rate of the event (for each component instance) and, optionally, a list of propagation modes.

Propagation modes are exclusive events (at most one of them can occur) and their description includes the probability of the mode and a list of propagation events. Propagation events are not exclusive (any combination of them can in general occur). A propagation event propagates the failure of the component to a given number of operational instances of a set of components of a given class. The set is specified using a special construct called component selector which allows navigation on the instantiation tree of the system (starting from the level at which the component is instantiated) using a UNIX-like syntax. The selection of the set of components to which the failure can be propagated can be refined by specifying a list of operational modes in which the components can be affected. The description of a propagation event includes also the mode in which the components are affected and the probability of the event. Probabilities of propagation modes and events can be omitted with a default value of 1.

The component class `ProcC` has two repair modes. The first one is used for hard failures and is scheduled to the repair team `Fieldengineers`; the second one is used for soft failures, is scheduled to the repair team `Operators`, and can only be undertaken if at least one channel is operational (note that this implies that at least one processor and the other database of the site are also operational).

Repair strategies are specified in `REPAIR TEAM` constructs which are illustrated by the repair team `Fieldengineers` of the distributed fault-tolerant database system:

```
REPAIR TEAM: Fieldengineers
    NUMBER: 2
    STRATEGY: priority
        DbC: 1
        SimpleC: 2
        ProcC: 3
```

The team has two repairmen and uses a preemptive priority strategy, in which failed components scheduled to the team are selected according to given priorities, with the selection among components with the same priority being at random when there are more failed components than remaining repairmen. Priorities can be assigned to component with given names as well as classes and can be made dependent of the mode in which the components are failed. It is possible to specify and `unlimited` number of repairmen. In this case, the `STRATEGY` construct is omitted. The simpler strategy `ros` (random order service), in which priorities are not specified, can be used when all the components scheduled to the repair team have the same priority.

2.4. Metric specification

The same type of metrics which are incorporated in METFAC [5] can be evaluated from a model specified using the language described here. These metrics include, among others, the steady-state availability, the point availability, the mean time between failures, the mean time to failure, the reliability, and the maintainability, as well as cost and performance-related metrics, which result from the assignment of cost or performance indices to the states of the model. The metric to be evaluated is specified using the `METRIC` construct. This construct includes the `OPTION` construct to select the metric and, if required by the chosen option, the `INITIAL STATE` and `INDEX` constructs. The `INITIAL STATE` construct can be omitted for the options requiring an operational state with the default of the state in which all components are unfailed and has the structure illustrated in the next section. The `INDEX` construct specifies a function to be used for the evaluation of state indices.

3. EXPLOITING SYMMETRIES

A CTMC is automatically constructed from a model specification after selecting a particular type of metric. Advantage is taken of the symmetries made explicit by the model specification to reduce the size of the generated CTMC. This is done using a hierarchical state description in which cluster macrostates and component states of instances with the same name are factorized out. This is possible because, by construction, components and clusters with the same instantiation name are undistinguishable. We give next for illustration purposes the description of the state of the fault-tolerant distributed database in which one front-end is failed and one processor of one site is in hard failed mode.

```
CLUSTERS: Channel
    MACROSTATE
        INSTANCES: 1
        MODE: dormant
        COMPONENTS: Fe
            failed: 1
        COMPONENTS: Lan
            dormant: 1
    MACROSTATE
        INSTANCES: 1
        MODE: operational
        COMPONENTS: Fe
            operational: 1
        COMPONENTS: Lan
            operational: 1
CLUSTERS: Site
     MACROSTATE
         INSTANCES: 1
         MODE: operational
         COMPONENTS: Proc
             operational: 1
             Hf: 1
         COMPONENTS: Db
             operational: 2
     MACROSTATE
         INSTANCES: 3
         MODE: operational
         COMPONENTS: Proc
             operational: 2
         COMPONENTS: Db
             operational: 2
```

The reduction of size obtained can be significant. For instance, without using the cluster construct of the language, front-ends, LAN's, and the processors and databases of different sites have to be considered different, and the generated CTMC required for the computation of the availability has 408,969 states, while the lumped CTMC obtained from the specification using clusters has only 14,850 states (about 27 times less). The reduction is also significant in combination with pruning techniques [13]: 39 states for the reduced CTMC against 227 states when only states with up to two failed components are generated.

4. CONCLUSIONS

A hierarchical, object-oriented modeling language for fault-tolerant computer systems has been overviewed and its expressive power illustrated by an example of moderate complexity. We have used the language to specify concisely models for substantially more complex systems [11]. By providing facilities to express symmetries that complex systems often exhibit at levels higher than the component level, it is possible to generate automatically lumped state-level models of much smaller size than would be generated from a flat, component-level system description. This is important since it is the size of the state-level model what restricts the applicability of numerical solution methods. The kind of symmetries which the language captures could be extended by using concepts recently developed in the context of Petri net modeling [14], [15].

REFERENCES

[1] K. Trivedi, R. Geist, M. Smotherman, and J. B. Dugan ,"Hybrid modeling of fault-tolerant systems" *Comput. Elec. Eng. Int. J.*, vol. 11, no. 2/3, pp. 87–108, 1984.

[2] R. Geist, M. Smotherman and R. Talley, "Modeling Recovery Time Distributions in Ultrareliable Fault-Tolerant Systems," in *Proc. 20th Int. Symp. on Fault-Tolerant Computing (FTCS-20)*, Newcastle upon Tyne, June 1990, pp. 499–504.

[3] M. A. Marsan, G. Conte, and G. Balbo, "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems," *ACM Trans. on Computer Systems*, vol. 2, pp. 93–122, May 1984.

[4] O. C. Ibe, A. Sathaye, R. C. Howe, and K. S. Trivedi, "Stochastic Petri Net Modeling of VAXcluster System Availability," in *Proc. 3rd. Int. Workshop on Petri Nets and Performance Models (PNPM89)*, Kyoto, Japan, December 1989, pp. 112–121.

[5] J. A. Carrasco and J. Figueras, "METFAC: Design and Implementation of a Software Tool for Modeling and Evaluation of Complex Fault-Tolerant Computing Systems," in *Proc. 16th Int. Symp. on Fault-Tolerant Computing (FTCS-16)*, Vienna, July 1986, pp. 424–429.

[6] A. Goyal, W. C. Carter, E. de Souza e Silva, S. S. Lavenberg, and K. S. Trivedi, "The System Availability Estimator," in *Proc. 16th Int. Symp. on Fault-Tolerant Computing (FTCS-16)*, Vienna, July 1986, pp. 84–89.

[7] A. Goyal and S. S. Lavenberg, "Modeling and Analysis of Computer System Availability," IBM Research Rep. RC12458, Yorktown Heights, Nov. 1986.

[8] A. E. Conway and A. Goyal, "Monte Carlo Simulation of Computer System Availability/Reliability Models," in *Proc. 17th Int. Symp. on Fault-Tolerant Computing (FTCS-17)*, Pittsburgh, 1987, pp. 230–235.

[9] Juan A. Carrasco, "Failure distance-based simulation of repairable fault- tolerant systems," in *Proc. 5th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, Torino, February 1991, pp. 337–351.

[10] Juan A. Carrasco, "Efficient transient simulation of failure/repair markovian models," Technical Report, UPC, June 1990, submitted for publication.

[11] Juan A. Carrasco, "Dependability Modeling in SMART", Report, ESPRIT project 1609: SMART (System Measurement and Architecture Techniques), January 1990.

[12] R. E. Barlow and F. Proschan, *Statistical Theory of Reliability and Life Testing: Probability Models*, Silver Spring, 1981.

[13] R. R. Muntz, E. de Souza e Silva, and A. Goyal, "Bounding Availability of Repairable Computer Systems," *IEEE Trans. on Computers*, vol. 38, no. 12, December 1989, pp. 1714–1723.

[14] C. Dutheillet and S. Haddad, "Aggregation of States in Colored Stochastic Petri Nets: Application to a Multiprocessor Architecture," in *Proc. 3rd Int. Workshop on Petri Nets and Performance Models (PNPM89)*, Kyoto, December 1989, pp. 40–49.

[15] Juan A. Carrasco, "Automated Construction of Compound Markov Chains from Generalized Stochastic High-Level Petri Nets," in *Proc. 3rd Int. Workshop on Petri Nets and Performance Models (PNPM89)*, Kyoto, December 1989, pp. 93–103.