Tesi di Dottorato di Ricerca in
Informatica ed Applicazioni

XVIII ciclo

ON THE MODELING AND SOLUTION OF COMPLEX SYSTEMS: FROM TWO DOMAIN-SPECIFIC CASE-STUDIES TOWARDS THE DEFINITION OF A MORE GENERAL FRAMEWORK

Paolo Lollini

Titolo della tesi:
On the modeling and solution of complex systems: from two domain-specific case-studies towards the definition of a more general framework.

Il candidato:
Paolo Lollini

Il tutore:
prof. Andrea Bondavalli

Il co-tutore:
dr. Felicita Di Giandomenico

Il coordinatore:
prof. Rocco De Nicola

December, 2005

Università degli Studi di Firenze
Dipartimento di Sistemi e Informatica
Viale Morgagni 65, 50134 Firenze
Italy

INTRODUCTION

Complex software and hardware systems are widely used in different applications and they have become pervasive in many fields of human activity. Each system demands some specific properties, as a certain level of availability, reliability, performance or quality of service (QoS), whose evaluation has become a key issue in information technology and computer science. The quantitative evaluation of these system's properties is performed following two basic approaches: *measurement-based* and *model-based*. In the first approach, the required measures are estimated from measured data using statistical inference techniques, and the data are measured from a real system or from its prototype. It is usually a very expensive approach, since it requires to build a real system, take the measurements and analyze the data statistically. On the contrary, the model-based approach is inexpensive and easier to perform, since the system evaluation does not require to build and measure the system. The model-based approach can be used for system assessment in all phases of the system life cycle. During design phase, models give an early validation of the concepts and architectural choices, allow comparing different solutions to highlight problems within the design and to select the most suitable one. During the operational life of the systems, models allow to detect bottlenecks and to suggest solutions to be adopted for future releases. Moreover, the sensitivity analysis can be carried out after modelling, and it allows to identify system bottlenecks, highlight problems in the design, and identify the critical parameters, that are those to which the system is highly sensitive.

The models we are interested in have to cope with an important feature: the *random phenomena*. A random phenomenon is characterized by the fact that its future behavior is not predictable in a deterministic fashion. A very simple random phenomenon is, for example, the toss of a coin. We do not known deterministically which side we will obtain at each toss, but we can treat these phenomena using some statistical regularities. Therefore, the random phenomena are usually capable of mathematical descriptions and we can treat them introducing the probability theory and the concepts of random variables (discrete and continuous), probability mass functions, probability density functions, cumulative distribution functions (discrete and continuous), random processes (Markovian or not), and so on ([1]).

A model-based evaluation can be carried out through *discrete-event simulation* or *analytical models* (or a combination of them). A discrete-event simulator is a program whose execution simulates the dynamic behavior of the system and

evaluates the required measures. In general it can be used to solve all the models, whereas analytical solvers can be used on only those models having some particular properties (e.g. the underlying random process is Markovian). One of the main advantages of simulation over analytical modeling is the flexibility and generality of the solvable models; on the other hand, the numerical solvers are capable of providing exact solutions (up to machine precision), whereas simulation provides statistically accurate solutions within some user-specifiable confidence interval. An analytical model consists of a set of equations that describes the system behavior, and the solution of these equations provides the measures of interest. We can distinguish between *state* and *non-state* space analytical models. The non-state space models can be solved without generating the underlying state space, and they are based on the assumptions of statistically independent failures and independent repair units for components. Thanks to these assumptions, the solution can be computed in a very efficient way also for complex systems with hundred of components. Examples of non-state space models are Reliability Block Diagram (RBD) and Fault Trees (FT) [2]. Unfortunately, the introduced independency assumptions usually do not hold for most real systems, for which the use of state space models is mandatory.

The *complexity* of the state space models is a very critical problem that needs to be addressed very carefully. Being able to describe critical complex systems by accounting at the same time for all the relevant aspects is not trivial at all. The models built for evaluating the measures of interest are always a tradeoff between correctness of representation of the real systems behavior and capability to solve the model equations to obtain the measures. On one side, we would like to perfectly represent the system's behavior accounting for all the details that can have an influence on the measures of interest. On the other side, we aim to efficiently solve the obtained models despite their high level of complexity. Complexity may induce several problems, like very large state spaces for state-based analytical solutions (and consequently large memory requirements) or unacceptably long solution times for analytic solutions and simulations in case of stiff models, i.e. models having events occurring at very different time scales.

Several works have been presented in the literature trying to cope with the complexity problem. A first approach consists in building models with an high level of abstraction (or, equivalently, with a low level of detail). Using appropriate modeling formalisms (e.g. Markov Chains, Petri Nets, Stochastic Petri Nets and their extensions), some system's details are not explicitly modeled but they are described at an higher level as an aggregated information, thus leading to more tractable models. The major limitation is that the abstraction process usually introduces some approximations in the representation of the system behavior that can cause a loss of accuracy of the measures of interest or the model representa-

tiveness. Other works try to tackle the complexity problem building models in a modular way through a composition of its submodels (e.g. [3, 4, 5, 6, 7, 8, 9]). Some compositional rules are defined to interconnect the submodels, and the dependencies between submodels are exploited to manage the model complexity creating smaller, equivalent representations. Finally, other works follow the decomposition/aggregation approach (e.g. [10, 11, 12, 13, 14, 15, 16, 17]), whose basic idea is to decompose a large model in a set of more tractable submodels that are solved separately, possibly passing some intermediate results. The measures obtained from the evaluation of each submodel are then aggregated to obtain the measures for the original model.

It appears quite evident that lot of work has been done to model and solve complex systems, but a lot of work still remains which has to be done in the future. One of the major weaknesses in this research area is the lack of a general methodology applicable to a wide class of systems, since all the existing techniques are domain-specific and then can help the construction and the solution of a limited (although, possibly, well populated) class of systems. Though a universal methodology able to model and efficiently solve all the existing systems does not exist, in this dissertation we give a contribution in the definition of a general modeling and solution framework, focusing the attention on the decomposition/aggregation techniques. We follow a very pragmatic approach. We first tackle two domain-specific case-studies: the first concerning a mobile telephone infrastructure, and the second focusing on a class of hierarchical control systems. The complexity of the models has been managed through the definition of two specific decomposition techniques, one for each case-study. Both works provide a very useful insight in their respective application-domain, showing very attractive potentialities that could be exploited in the analysis of other similar problems. At the same time, they enable us to understand the main issues involved in the decomposition approach. This knowledge has been used in the second part of the dissertation to depict a more general modeling and solution framework that, although to be further detailed and refined, seems to have very good potentialities. Then, such methodology has been applied to the mobile telephone infrastructure case-study in order to prove its feasibility.

In more detail, the main topics addressed in the dissertation are the following.

1. *Analysis of a complete General Packet Radio Service (GPRS) mobile telephone infrastructure*, composed by a number of adjacent cells partially overlapped [18]. We consider one cell as affected by an outage and, through a transient analysis, we evaluate the effectiveness of a specific class of resource management techniques for congestion treatment in terms of service

availability related indicators. This work is a major extension and refinement to the previous studies dealing with a GPRS infrastructure ([19, 20]), and the classical availability analysis is enhanced by taking into account the congestion following outages and its impact on user's perceived QoS, both in each cell and in the overall GPRS network. In order to efficiently solve the large and complex model capturing the network's behavior, we introduce a decomposition/aggregation approach in which the solution of the entire model is constructed on the basis of the solutions of the individual sub-models.

2. *Efficient dependability evaluation of hierarchical control and resource management systems* [21, 22, 23]. We exploited the characteristics of this specific, but important, class of systems and derived a modeling methodology that is not only directed to build models in a compositional way, but it also includes some capabilities to reduce their solution complexity. The modeling methodology and the resolution technique are then applied to a case study consisting of a resource management system developed in the context of the recently concluded European project CAUTION++ [24]. The results obtained are useful to understand the impact of several system component factors on the dependability of the overall system instance.

3. *Definition of a general framework to model and solve complex systems through decomposition*. We propose a very natural decomposition/aggregation approach that operates at the system-level definition, rather than at the model-level one. Through a functional decomposition, the system is first decomposed in a set of sub-systems, called "entities". In each instant, an entity can i) work in isolation, or ii) interact with other entities through some "dependency relations", that are connections that state how the behavior of an entity affects the behavior of the others. The system's lifetime is then decomposed in a sequence of phases such that two consecutive phases have at least one different dependency relation (temporal decomposition). The modelling procedure is then applied to this decomposed system. First, following a modular approach, we build a single model representing the behavior of the whole system. Then, the application of a decomposition approach at model-level produces a set of separate submodels (one for each entity) that can be solved in isolation, passing some intermediate results between them if and when required (physical decomposition). The proposed modeling and solution framework is general, in the sense that it is not application-domain specific, and it induces a mitigation of the model complexity (both temporal and spatial) thanks to the adopted decomposition approach. A critical point is the accuracy of the final results that has to be addressed very

carefully, but it actually depends on the real system under analysis and on the way in which the models are separated.

4. *Feasibility case-study*. The case-study of the mobile telephone infrastructure presented in the first part of the dissertation is here reused as application domain for the proposed general modeling methodology. The goal is twofold: on one side we want to demonstrate the feasibility of the methodology, and on the other side we aim to evaluate its effectiveness and accuracy in solving a real system. The obtained results are then compared with those produced by the solution of the whole non-decomposed model (that we consider the exact solution) and with those produced by the application of the methodology specifically tailored for the considered class of systems. The first analysis emphasizes the efficacy of the decomposed modeling approach that substantially reduces the total computational time. The second one leads us to note that the application of the general modeling methodology enhances the modular construction of the models (each one performing a well-separated critical function), although as expected the processing time slightly gets worse with respect to the domain-specific methodology (but it is the price to be payed when a general methodology is applied to solve a very specific system), while the accuracy of the final results are similar.

The structure of the dissertation is the following. The research's framework is presented in Chapter 1. Chapter 2 deals with the analysis of the GPRS mobile telephone infrastructure, while the methodology for the dependability evaluation of hierarchical control and resource management systems is described in Chapter 3. The general modeling and solution framework is then presented in Chapter 4, and the feasibility case-study is shown in Chapter 5. Finally, conclusions are in Chapter 6.

This dissertation is also available in pdf format at the web site address `http://rcl.dsi.unifi.it/theses/TesiLollini.pdf`.
A detailed technical documentation concerning the models introduced in the dissertation is downloadable at the following web site address: `http://rcl.dsi.unifi.it/theses/TesiLollini_techDoc.pdf`.

ACKNOWLEDGEMENTS

I want to deeply thank my tutor Prof. Andrea Bondavalli, from the department of Computer Science of the University of Florence, and my co-tutor Dr. Felicita Di Giandomenico, from the institute ISTI of the Italian National Research Council. Thanks for your invaluable support and for the time we enjoy working together.

I also wish to thank all of the members of the PERFORM research group of the University of Illinois at Urbana-Champaign, both for their technical assistance on this dissertation and for their assistance to me during my visiting period at Urbana.

Many thanks to whoever gave me a moral support and encouragement, and in particular the colleagues and friends of the "PhD room" at DSI. Last but certainly not least, I would like to thank Chiara for her help, patience and love, and my family.

Contents

Chapter 1

Framework of the Research

The complexity of computerized systems has become a very critical issue in our daily lives. Many people need to rely upon the services provided by these system, as their malfunctions can cause very serious consequences both in terms of loss of human's life and/or in terms of conspicuous economical damages. Therefore, it is mandatory for such critical systems to have some specific properties that allow us to "sufficiently trust" on the correct delivery of their services. The set of these system properties is studied in the conceptual framework of *dependability*.

The purpose of this Chapter is to give an overview of the framework of the research in which the dissertation is contributing to. The focus is on the *model-based* approach for the evaluation of the dependability properties, that has proven to be useful and versatile in all the phases of the system life cycle. One of the main problems that has to be taken into account when we want to model a system is the management of *complexity*. To master complexity a modeling methodology is needed so that only the relevant aspects can be detailed still allowing numerical results to be effectively computable. A solution in this sense is the introduction of simplifying hypotheses, but it is a critical activity because the introduced approximation could limit or invalidate our faithfulness on results. A complementary (usually non alternative) approach is the application of a modeling technique that also includes some capabilities to reduce the overall complexity.

The rest of this Chapter is organized as follows. In Section 1.1 we introduce the dependability attributes and the *threats* to dependability (faults, errors and failures), while the *means* to achieve dependability are outlined in Section 1.2. The role of the model-based approach in the evaluation of the dependability attributes is discussed in Section 1.3, while some of the most used *modeling formalisms* based on Petri Nets and its extensions are outlined in Section 1.4. In Sections 1.5 and 1.6 we present two classes of available techniques that try to cope with the complexity problem, the first following a modular *construction/composition* ap-

proach, and the second a *decomposition/aggregation* approach. Finally, in Section 1.7 we give an overview of the available tools that can help the user to build and solve models, classifying them with respect to the modeling and solution features they provide.

1.1 Basic concepts

Dependability is a property of a computer system such that reliance can justifiably be placed on the services the system provides [25], and it is a term that unifies a number of different system characteristics:

- *availability*, the measure of the delivery of correct service with respect to the alternation between correct and incorrect service;
- *reliability*, the measure of continuous delivery of correct service;
- *confidentiality*, the absence of unauthorized disclosure of information integrity;
- *maintainability*, the ability to undergo repairs and modifications;
- *safety*, the absence of catastrophic consequences on the user(s) and on the environment;
- *integrity*, the absence of improper system state alterations.

Several other dependability or dependability-related attributes have been defined, among which: *security*, as the combination of availability, confidentiality and integrity; *performability*, which takes into account performance in degraded system states.

Although the systems are built to satisfy all the requirements prescribed for the service they provide, for complex systems it is practically impossible to guarantee that they will work properly during its whole life-cycle, and it is due to the occurrence of *faults*, *errors* and *failures*.

A *fault* is an adjudged or hypothesized cause of an error. With respect to its duration, it can be permanent, intermittent or transient: a permanent fault is continuous and stable, an intermittent fault is a fault that is only occasionally present due to instable hardware or varying hardware or software state, while transient faults are resulting from temporary environmental conditions. The faults can also be classified with respect to their origin. Physical faults arise from physical phenomena internal to the system (such as shorts or opens), or external changes (such

as environmental or electromagnetic). Human faults may be either design faults, which are committed during system design, modification, or establishment of operating procedures, or the may be interaction faults, which are violations of operating or maintenance procedures. With respect to their status, faults can also be classified in active, when they produce an error, otherwise they are dormant. Other possible classifications are in [26].

An *error* is the manifestation of a fault within a program or data structure. It is a part of the system state that may cause a subsequent failure: a failure occurs when an error reaches the service interface and alters the service.

A *failure* occurs when the delivered service deviates from the specified service; failures are caused by errors. A system may not, and generally does not, always fail in the same way. The ways a system can fail are its failure modes, and they can be characterized by their domain (value and timing failures), by the perception by the system users (consistent and byzantine failures), by the capability to detect them (signalled or un-signaled failures) and by the consequences on the system environment (from benign to catastrophic failures). An *outage* is a special case of failure that is defined as a loss or degradation of service to a customer for a period of time (called outage duration). In general, outages can be caused by hardware or software failures, human errors, and environmental variables (e.g. lightning, power failures, and fire). A failure resulting in the loss of functionality of the entire system is called system outage.

Faults, errors and failures are the threats to dependability as they can induce a system to deliver an incorrect service (or to deliver no service), and their effect is to deteriorate the dependability attributes (or some of them).

The causality relationship between faults, errors and failures is shown in Figure 1.1. An active fault is either a) an internal fault that was previously dormant and that has been activated by the computation process or environmental conditions, or b) an external fault. A fault is activated when an input is applied to a component that causes a dormant fault to become active. The computation process can induce an error that can propagate within a given component (i.e., internal propagation): an error is successively transformed into other errors. Error propagation from one component (A) to another component (B) that receives service from A (i.e., external propagation) occurs when, through internal propagation, an error reaches the service interface of component A. At this time, service delivered by A to B becomes incorrect, and the ensuing failure of A appears as an external fault to B and propagates the error into B.

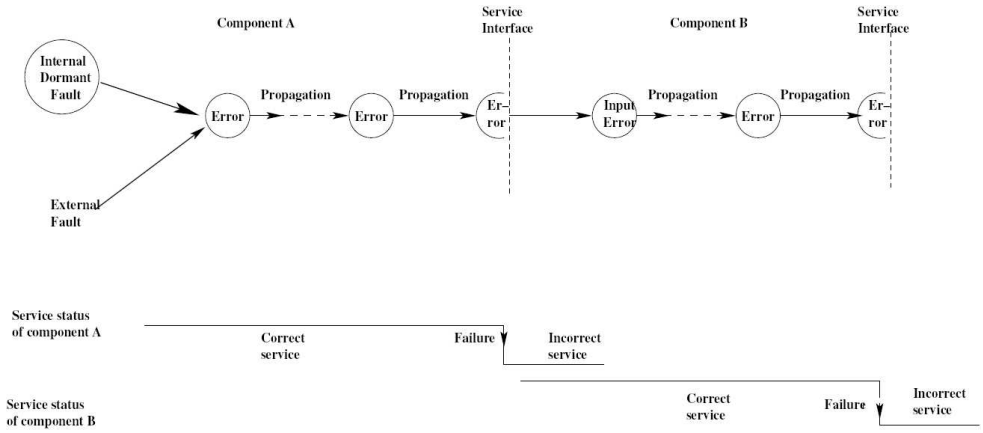

Figure 1.1: Error propagation scheme

1.2 The means for dependability

In this Section we outline the means to make a system dependable. They can be classified in four classes, but usually the development of a dependable computing system requires the application of a set of methods, rather than a unique one.

- *Fault Prevention*. The techniques belonging to this class aim to prevent the occurrence or the introduction of faults in the system. Examples are design review, component screening, testing, quality of control methods, formal methods and software engineering methods in general.

- *Fault Tolerance*. The fault tolerance techniques enable a system to provide a correct service in spite of faults. Fault tolerance is carried out by error processing and fault treatment: the first aims at removing errors from the computational state, possibly before the occurrence of a failure, while the second aims at preventing faults from being activated again.

  - *Error processing*. There are two techniques to carry out error processing: error recovery and error compensation. They both need error detection capability, that is the capability to identify the erroneous state as soon as it is activated, or before it propagates and produces a failure. Error recovery means that a detected erroneous state is substituted by a non-erroneous state, while error compensation means that the detected erroneous state contains enough redundancy to deliver an error-free service from the erroneous internal state. When error compensation is applied systematically, even in the absence of fault (for example majority vote), it is called fault masking.
  - *Fault treatment*. It consists of the following steps:

1. Fault diagnosis, determining the causes of errors in both location and nature;
2. Fault isolation, preventing the faults from being activated again by removing the components identified as being faulty from further executions;
3. System reconfiguration, which either switches in spare components or reassign tasks among non-failed components, whenever the system is no longer capable of delivering the same service. This reconfiguration should enable the delivery of an acceptable service, even if the system capability degrade (graceful degradation);
4. System re-initialization, which checks, updates and records the new configuration and update system tables and records.

A fault tolerant system can be achieved, for example, by recurring to redundant techniques, useful against independent faults, and design diversity, useful against design faults. The use of redundancy can provide the information needed to negate the effects of faults. There exist several dimensions of redundancy: time redundancy which is provided by software (extra executions of the same calculation which may be accomplished by different methods), components redundancy which is provided by hardware or software (use of extra memory, bus lines, functional modules to supply extra information), and information redundancy (mapping data in new representation containing redundant information to allow fault detection and fault masking).

- *Fault Removal*. These techniques aim to reduce the presence (number, seriousness) of faults, and they are obtained by means of a set of techniques used after that the system has been built. They are verification (checking whether the system adheres to properties, termed the verification conditions), diagnosis (diagnosis the fault which prevented the verification conditions from being fulfilled), and correction.

- *Fault Forecasting*. The purpose of the fault forecasting techniques is to estimate the present number, the future incidence and consequences of faults. Indeed, no existing fault tolerant technique is capable to avoid a failure scenario, then the dependability evaluation represents a suitable mean to verify the adequacy of a system design with respect to the requirements given in its specification. In this sense, fault forecasting is a way to achieve system assessment. Fault forecasting has two aspects: i) qualitative evaluation, that aims to identify, classify, and rank the failure modes, or the event combinations that would lead to system failures, and ii) quantitative evaluation, that

aims to evaluate in terms of probabilities the extent to which some of the attributes are satisfied; those attributes are then viewed as measures. In the following Section we focus the attention on this last aspect.

1.3 Quantitative evaluation of dependability attributes, and the role of the model-based approach

The quantitative evaluation of dependability attributes can be used for fault-forecasting [25], that is to probabilistically estimate the occurrence of faults and their impact on the ability of the system to provide a proper service. System assessment can be performed using several approaches like testing (experimental evaluation) and model-based evaluation, often combined together. Testing is a dynamic verification that consists in exercising a system with some actual inputs, observing the outputs and then deciding whether or not they satisfy the verification conditions. Testing necessitates faults or errors to be part of the input test patterns, that is usually referred to as fault injection [27]. It is an attractive option for assessing an existing system or prototype, but costly and not always applicable, e.g., when the interest is in very rare events.

As defined in [28], a model is an abstraction of a system "that highlights the important features of the system organization and provides ways of quantifying its properties neglecting all those details that are relevant for the actual implementation, but that are marginal for the objective of the study". Model-based evaluation is usually cheaper than experimental evaluation and it can be used in all the phases of the system life cycle. During the design phase, models allow to compare different alternative architectural solutions, to select the most suitable one and to highlight problems within the design ("early" validation). Once design decisions are made, models allow to predict the overall behavior of the system. Finally, for an already existing system, models allow an "a posteriori" dependability analysis [29], to understand and learn about specific aspects, to detect possible design weak points or bottlenecks, to perform a late validation of the dependability requirements and to suggest sound solutions for future releases or modifications of the systems.

There exist several types of models, and the choice of a proper model depends on many factors, like the complexity of the system, the specific aspects to be studied, the attributes to be evaluated, the accuracy required, and the resources available for the study. One of the most commonly used modeling techniques is discrete-event simulation, especially for highly complex systems for which analytical solution is generally precluded. Its major weakness is that it produces solutions that are only estimations of the exact measures. On the contrary, an-

alytical modeling provides exact solutions (up to machine precision) and thanks to recent developments in model generation, solution techniques and automated tools, it is a very used alternative to simulation for dependability evaluation.

Modeling shows also several problems that must be taken under special care. The first problem is complexity, as the overall description of critical complex systems can be a very critical issue. Another problem is the stiffness [30], which arises when model parameters assume values whose orders of magnitude differ significantly from each other. To master complexity and explosion of information to be accounted, a modeling methodology is needed so that only the relevant aspects can be detailed thus allowing numerical results to be effectively computable. In addition, simplifying hypotheses are very often necessary to keep the model manageable; of course, the choice of such hypotheses is critical, to avoid resulting in a system model too far from the real behavior that evaluation results become useless in practice. Despite the recent results in the analytical modeling field, the information explosion problem remains the major difficulty for analyzing practical applications. Another problem is the determination of the values to assign to the parameters required by the models. Actually these values can be difficult to obtain (usually by way of experimental tests), and they can not be provided during the preliminary design phases of the system. Since even slight variations of critical parameter values may result in relevant changes of system dependability attributes, a thorough calibration of such parameters is necessary to increase the level of confidence that can be put on the dependability evaluation itself.

In the following Section we present some of the most used modeling formalisms for the evaluation of dependability/performability attributes.

1.4 Modeling formalisms

Markov Chains (MCs), Petri Nets (PNs), Stochastic Petri Nets (SPNs), Generalized Stochastic Petri Nets (GSPNs), Deterministic and Stochastic Petri Nets (DSPNs) and Stochastic Activity Networks (SANs) are only some examples of modeling formalisms that facilitate abstraction. They are listed following an increasing abstraction order, from the less abstract level (MCs) to the higher one (SANs). Each formalism has its own peculiarity. For example, some formalisms provide very efficient solution methods (e.g., DSPN offers a very efficient analytical solution technique under some assumptions [31]), while some others offer a powerful way to compactly represent very complex behaviors (e.g., SAN). The advantage of using these formalisms is that a modeler can describe a system with a low level of details, thus developing more tractable models. The major limitation is that the abstraction process usually introduces some approximations in the

representation of the system behavior that can cause a loss of accuracy of the measures of interest or the model representativeness. Therefore, the applicability of this modeling approach is usually a trade-off between model tractability and accuracy of the obtained results. Their main characteristics will be briefly described in the following Subsections.

1.4.1 Markov chains

A Markov Chain (MC) [1] is a Markov process with a discrete (or countable) state space. A system can be modeled using a MC if its evolution in time is independent from the past, but only depends on the current state. The set of possible states of a Markov chain is called the state space, denoted by S. A state change of a Markov chain is called a state transition. More formally, a MC is a stochastic process $\{X(t), t \geq 0\}$ with a discrete state space such that for any $n > 0$ and any sequence of increasing time instants $t_1, t_2, \ldots, t_n, t_{n+1}$, the following equation holds:

$$Prob\{X(t_{n+1}) = j | X(t_n) = i_n, X(t_{n-1}) = i_{n-1}, \ldots, X(t_1) = i_1\} =$$

$$Prob\{X(t_{n+1}) = j | X(t_n) = i_n\} \qquad \forall j, i_n, i_{n-1}, \ldots, i_1 \in S \qquad (1.1)$$

It is the memoryless (or Markov) property: the future behavior of the process is independent from its past. If the exact characterization of the present state of the process is independent from the current time, then the Markov chain is said to be time-homogeneous, otherwise it is said to be a non-homogeneous Markov chain. The parameter t that indexes the Markov chain can be either discrete or continuous. In the first case we have a discrete-time Markov chain $\{Xn | n \geq 0\}$, where state transitions only occur at discrete points in time, often called steps, whereas in the latter case we have a continuous-time Markov chain $\{X(t) | t \geq 0\}$ and state transitions may occur at any point in time. Because of the memoryless property, each transition from state i to state j of a homogeneous continuous-time Markov chain occurs in an exponentially distributed time, and the rate of the transition is exactly the inverse of the expected time to the transition, that is the rate of the corresponding exponential distribution. This implies that time needed to perform whichever activity of a system must be modelled with an exponential transition in the Markov chain model of that system. This is the most severe constraint that limits the applicability of MCs. If we model a transition having non-exponential duration with an exponential transition, we unavoidably induce an approximation in the model that can have significant impact on the final results. In order to cope with this problem, there exist several techniques that try to contain this error. For example, the phase expansion [32] uses a sequence of exponential stages to approximate a non-exponential random variable.

The solution of a MC model consists in solving ordinary differential equations (for transient solutions) or linear equations (for steady-state solution) using some available numerical solution technique.

1.4.2 Petri Nets

Petri Nets (PN) were originally introduced by C. A. Petri in 1962. Formally [1, 33], a place-transition Petri net (PN) is 5-tuple PN = $(P; T; A; M; \mu_0)$, where:

- $P = \{P_1, P_2, \ldots, P_n\}$ is a finite set of places (draw as circles).
- $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions (draws as bars).
- $A \subseteq (P \times T) \bigcup (T \times P)$ is a set of arcs connecting P and T. Arcs going from a place to a transition are called input arcs, and arcs directed from a transition to a place are called output arcs.
- $M : A \rightarrow \{1, 2, 3, \ldots\}$ is the multiplicity associated with the arcs in A.
- $\mu : P \rightarrow \{1, 2, 3, \ldots\}$ is the marking that denotes the number of tokens (drawn as black dots or a positive integer) for each place in P. The initial marking is denoted with μ_0.

The places that are linked to transition t by an input arc are called the input places of the transition. Similarly, the places linked to transition t by an output arc are called the output places of the transition. In the graphical representation of the Petri net model, places are drawn as circles and transitions are drawn as bars, with the input and output arcs linking them. Places may contain tokens, which are represented as black dots. The state of the Petri net model is a vector $(m(P1), m(P2), \ldots, m(Pn))$ called the marking of the net, and it is defined by the number of tokens $m(Pi)$ in each place i of the model. Transitions model activities which can occur (the transition fires) and change the state of the system (the marking of the Petri net). Transitions are only allowed to fire if they are enabled, and this happens when there are enough tokens available in the corresponding input places. When the transition fires, it removes from each of its input places a number of tokens equal to the cardinality of the corresponding input arc, and adds to each of its output places a number of tokens equal to the cardinality of the corresponding output arc. When two enabled transitions share an input place and the number of tokens therein is not sufficient for both of them to fire, the transitions are said to be in conflict, and a selection rule (usually a priority associated to the transitions) must be employed to break the competition in favor of one of them.

A system can be modelled by representing its states as markings of the Petri Nets. Tokens can be used to represent entities of the system, such as tasks to be

executed, messages to be sent. Transitions model activities or synchronization constraints of the system, and the firing rules define the preconditions to be satisfied for the activities to be executed or the synchronization to be completed, respectively. The absence of time in the class of place-transition Petri nets does not allow quantitative analysis of the modelled systems. This formalism was mainly introduced to model qualitative aspects of systems (concurrency, parallelism) and to verify its structural properties (like, for example, the absence of deadlocks, a given order in the actions performed etc.).

1.4.3 Stochastic Petri Nets

A very popular timed extension of the place-transition Petri nets is the class of Stochastic Petri Nets (SPNs) [34]. In a SPN model, each transition t has an associated random firing delay exponentially distributed. The enabling of a transition is the same as of the PN models. As soon as a transition t gets enabled, a random firing time is sampled from the exponential distribution associated to t, and a timer starts counting from that time down to zero. Transition t fires if and only if it remains continuously enabled until the timer reaches zero. When t fires, the tokens are removed from their input places and added to the output places in a single atomic and instantaneous operation (atomic firing rule). It is interesting to observe that in the time interval between the enabling and the firing of t, other transitions sharing some input places with t can get enabled and fire without disabling it, provided that there is a sufficient number of tokens in the common input places. On the contrary, in the case of a conflict, the transition whose timer reaches zero the first is the one that fires (race model). It is also important to notice that the use of exponential distribution relieves the user from the specification of the behavior of those transitions that do not fire after having been enabled. Indeed, thanks to the memoryless property of the exponential distribution (see Equation 1.1), whether the memory of the time they have already been enabled is kept or not, the remaining time to the firing is exponentially distributed with the same rate. The evolution of a SPN model can be represented by a continuous-time homogeneous Markov chain, whose state space elements are in a one-to-one correspondence with the elements of the reachability set (the set of all the states reachable with a sequence of transition firing, starting from the initial marking), and whose transitions rates among states are equal to the firing rates of the transitions that produce the corresponding marking change in the SPN. An SPN model can be solved in terms of the marking occupation probabilities by performing the analysis of the associated Markov chain.

1.4.4 Generalized Stochastic Petri Nets

The class of Generalized Stochastic Petri Nets (GSPNs) [33, 35], relaxes the assumption that all the transitions have an exponentially distributed delay, and allows for exponential transitions, and for instantaneous transitions as well, that is transitions that once enabled fire in zero time. Conflicts among timed transitions are solved according with the same race model as in the case of SPNs, whereas conflicts among instantaneous transitions are solved by a priority assignment, and by associating weights (or probabilities) to instantaneous transitions at the same priority level. The solution of a GSPN model resorts again to that of an associate Markov chain. However, for GSPN models, the reachability set elements are not in a one-to-one correspondence with the states of the associated Markov chain. Indeed, because of the instantaneous transitions, some of markings in the reachability graph have a zero sojourn time, that is the GSPN model spends a zero time therein. These markings are called the vanishing markings of the GSPN model, whereas the non-vanishing markings are often called tangible markings. Nevertheless, it is possible to operate a reduction of the reachability graph to eliminate the vanishing markings, and to obtain the reduced reachability graph. The reduced reachability graph is isomorphic to a Markov chain, and the reduction procedure does not affect the equivalence between the non vanishing marking occupation probabilities of the GSPN and the state occupation probabilities of the Markov chain. Therefore, the Markov chain associated to the reduced reachability graph can be solved to study the GSPN model evolution over time.

1.4.5 Deterministic and Stochastic Petri Nets

Deterministic and Stochastic Petri Nets (DSPNs) [36] have been introduced as an extension of GSPNs, to allow the modeling of events having deterministic occurrence times [37]. The set of transitions of a DSPN can be partitioned into three disjoint sets: the set of instantaneous transitions, represented by a thin bar, the set of transitions having exponentially distributed firing times, represented by empty rectangles and the set of transitions with deterministic firing times represented by filled rectangles. This enriched set of possible transitions offered by DSPNs allows the exact modeling of a wider set of system features, such as timeouts and the message propagation delays in synchronous systems. Repair delays represent another example of activities that are typically more accurately modelled by deterministic transitions rather than by exponential ones. Unfortunately, the analytical solution of a DSPN model is not possible in general. Indeed, the deterministic distribution does not enjoy the Markov memory-less property, and the time-dependent evolution of the model requires keeping track of much additional information [38], which greatly complicates the analysis. However, the analytical

tractability is guaranteed for the subset of DSPN models whose structure satisfies the following assumption: at most one deterministic transition is enabled in each of the possible markings of the DSPN. This hypothesis severely limits the expressiveness of DSPN models, nevertheless, quite recently some attempts have been made to relax it [36].

1.4.6 Stochastic Activity Networks

Stochastic Activity Networks (SANs) were first introduced in [39] and then formally defined in [40]. The formalism is a generalization of SPNs, and have some similarities to the GSPN formalism. The building blocks composing a SAN are places, activities, arcs, input gates and output gates.

Places in SANs have the same interpretation as in PNs: they hold tokens, the number of tokens in a place is called the marking of that place, and the marking of the SAN is the vector containing the marking of all the places. There are two types of activities, timed and instantaneous.

Timed activities are used to represent delays in the system that affect the measure of interest, while instantaneous activities are used to abstract delays deemed insignificant relative to the measures of interest. Uncertainty about the length of the delay represented by a timed activity is described by a continuous probability distribution function, called the "activity time distribution function", that can be a generally distributed random variables, and each distribution can depend on the marking of the network. Activities can have cases. Cases are used to represent uncertainty about the action taken upon completion of an activity.

Gates connect activities and places. Input gates are connected to one or more places and one single activity. They have a predicate, a boolean function of the markings of the connected places, and an output function. When the predicate is true, the gate holds. Output gates are connected to one or more places, and the output side of an activity. If the activity has more than one case, output gates are connected to a single case. Output gates have only an output function. Gate functions (both for input and output gates) provide flexibility in defining how the markings of connected places change when the delay represented by an activity expires.

Arcs in SANs are default gates, defined to duplicate the behavior of arcs in Petri nets. Thus, arcs are directed. Each arc connects a single place and a single activity. The arc is an input arc if it is drawn from a place to an activity. An output arc is drawn from an activity to a place. An input arc holds if there is at least one token in the connected place. The function of an input arc removes a token from the connected place, while the function of an output arc adds a token to the connected place. An activity is thus enabled only when i) all of its input gates hold, and ii) all of its input arcs hold.

More formally, a *Stochastic Activity Network (SAN)* is defined as a 11-tuple $SAN = (P, IA, TA, IG, OG, IR, OR, C, F, \pi, \rho)$ where:

- P is a finite set of *places*,
- IA is a finite set of *instantaneous activities*,
- TA is a finite set of *timed activities*,
- IG is a finite set of *input gates*. Each input gate has a finite number of inputs. To each $G \in IG$, with m inputs, is associated a function $f_G : N^m \rightarrow N^m$, called the *function* of G, and a predicate $g_G : N^m \rightarrow \{true, false\}$, called the *enabling predicate* of G,
- OG is a finite set of *output gates*. Each output gate has a finite number of outputs. To each $G \in OG$, with m outputs, is associated a function $f_G : N^m \rightarrow N^m$, called the *function* of G,
- $IR \subseteq P \times \{1, ..., |P|\} \times IG \times (IA \cup TA)$ is the *input relation*. IR satisfies the following conditions:
  - For any $(P_1, i, G, a) \in IR$ such that G has m inputs, $i \leq m$,
  - For any $G \in IG$ with m inputs and $i \in N$, $i \leq m$, there exist $a \in (IA \cup TA)$ and $P_1 \in P$ such that $(P_1, i, G, a) \in IR$,
  - For any (P_1, i, G_1, a), $(P_1, j, G_2, a) \in IR$, $i = j$ and $G_1 = G_2$.

  In a graphical representation, $(P_k, k, G, a) \in IR$ means that place P_k is linked to k-th input of an input gate G whose output is connected to activity a. P_k is said to be an input place of a and G is referred to as an input gate of a.
- $OR \subseteq (IA \cup TA) \times OG \times \{1, ..., |P|\} \times P$ is the *output relation*. OR satisfies the following conditions:
  - For any $(a, i, G, P_1) \in OR$ such that G has m outputs, $i \leq m$,
  - For any $G \in OG$ with m outputs and $i \in N$, $i \leq m$, there exist $a \in (IA \cup TA)$ and $P_1 \in P$ such that $(a, G, i, P_1) \in OR$,
  - For any (a, G_1, i, P_1), $(a, G_2, j, P_1) \in OR$, $i = j$ and $G_1 = G_2$.

  In a graphical representation, $(a, G, k, P_k) \in OR$ means that activity a is linked to the input of an output gate G whose k-th output is connected to place P_k. G is said to be an output gate of a and P_k is referred to as an output place of a.

- $C : N^n \times IA \rightarrow [0,1]$ is the *case probability function*, where $n = |P|$.
- $F = \{F(.|\mu, a); \mu \in N^n, a \in TA\}$ is the set of *activity time distribution functions*, where $n = |P|$ and, for any $\mu \in N^n$, and $ainTA$, $F(.|\mu, a)$ is a probability distribution function,
- $\pi : N^n \times TA \rightarrow \{true, false\}$ is the *reactivation predicate*, where n is defined as before,
- $\rho : N^n \times TA \rightarrow R^+$ is the *enabling rate function*, where n is defined as before.

In order to better understand the models defined in this dissertation, all developed using SAN formalism, we give some further details on two particularly important aspects: the execution of a timed activity, and the measure specifications.

Completion rules

When an activity becomes enabled, it is activated, and the time between activation and the scheduled completion of an activity, called the activity time, is sampled from the activity time distribution. Upon completion of an activity, the following events take place: i) if the activity has cases, a case is (probabilistically) chosen; ii) the functions of all the connected input gates are executed; iii) tokens are removed from places connected by input arcs; iv) the functions of all the output gates connected to the chosen case are executed; v) tokens are added to places that are connected by output arcs to the chosen case. An activity is aborted when the SAN moves into a new stable marking in which at least one input gate no longer holds.

Measures definition

Upon completing the model of the system, a modeler has to specify the measures in terms of the model. In the SAN modeling framework, the measures are specified in terms of reward variables [41]. Let $R(m)$ be the rate at which the reward accumulated in state m, and let $C(a)$ be the reward earned upon completion of transition a. If $\{X_t, t > 0\}$ is the modeled stochastic process and M the set of all possible states, a reward variable collected at an instant of time conventionally denoted by V_t is informally defined as

$$V_t = \sum_m R(m)P(X_t = m) + \sum_a C(a)I_t^a \quad ,$$

where I_t^a is the indicator of the event that a was the activity that completed to bring the SAN into the marking observed at time t. The steady-state reward can be obtained as $V_{t\to\infty}$. In the case in which the reward variable is evaluated considering an interval of time $[t, t+l]$, then the accumulated reward is related both to the number of times each activity completes and to the time spent in a particular marking during the interval. More precisely,

$$Y_{[t,t+l]} = \sum_m R(m) J^m_{[t,t+l]} + \sum_a C(a) N^a_{[t,t+l]} \quad ,$$

where $J^m_{[t,t+l]}$ is a random variable representing the total time that the SAN is in the marking m during $[t, t+l]$ and $N^a_{[t,t+l]}$ is a random variable representing the number of completions of activity a during $[t, t+l]$.

1.5 Modular construction/composition approaches

Now the focus moves on the available modeling and solution techniques able to mitigate the complexity problem. In this Section we present some of the existing techniques that follow a modular construction/composition approach, while the techniques presented in Section 1.6 follow a decomposition/aggregation approach. Here the emphasis is on the modular model construction rather than on the model decomposition, and this is why we have grouped these techniques in two separate Sections. Indeed, the separation between the modular construction/composition approach and the decomposition/aggregation approach is not so marked.

The principle of the modular construction/composition approach is to build complex models in a modular way through a composition of its submodels. Most of the works belonging to this class define the rules that can be used to construct and interconnect the submodels, and they provide an easy way to describe the behavior of systems having an high degree of dependency among subcomponents. These dependencies can be exploited to manage the model complexity creating smaller, equivalent representations. In the following we outline some of the existing modeling approaches, although it is not an exhaustive list.

1.5.1 Blocks composition

The blocks composition approach has been presented in [3]. The authors developed a general approach that can be applied to any complex system of several hardware computers and software replicas. This approach is modular and it is based on GSPN, because of their ability to handle modularity and hierarchy. From the system composition and the interactions between components, a high level

behavioral model composed of blocks linked by arrows is derived. A block represents a GSPN describing either a component behavior or an interaction; the arrows indicate the direction of the links between the blocks. The GSPN of the blocks are derived in a second step. The GSPN of the architecture is obtained by composition of the GSPN of the components with those representing their interactions. In addition to modularity, the formalism brings flexibility and re-usability, thereby allowing easy sensitivity analysis with respect to the assumptions that could be made about the behavior of the components and the resulting interactions.

1.5.2 Replicate/Join formalism and graph composition

The replicate/join formalism [4, 5] is a composition technique for Stochastic Activity Networks (SAN) that combines models by sharing state, decreasing the overall number of states of the entire model. The Join operator takes as input a) a set of submodels and b) some shared places owning to different submodels of the set, and provides as output a new model that comprehends all the joined submodels' elements (places, arcs, activities) but with the shared places merged in a unique one. The Replicate operator combines multiple identical copies of a submodel, which are called replicates. It does not explicitly store the state of each submodel, but rather the number of copies of the model that are in a particular state. Storing the number of models in a state, rather than recording which models are in that state, decreases the overall number of states of the entire model.

[6] introduces the graph composition that extends the replicate/join formalism and it also combines models by sharing a portion of the state of each submodel, reducing the total state-space size. Contrarily to the replicate/join formalism that requires the use of a special operation, the graph composition detects all the symmetries exposed at the composition level and uses them to reduce the underlying state space. The graph composition uses the detected symmetries to replace each set of equivalent states with one aggregate state, reducing the total state-space size.

1.5.3 The "separation of concerns" approach

[7] provides a framework for modeling the performability of different Multipurpose Multiprocessor Systems (MMSs), that is application-independent support systems composed of Commercial Off-The-Shelf components (COTS). The originality of the approach is in the explicit separation between the architectural and the environmental concerns of a system. The overall dependability model, based on Stochastic Reward Nets, is composed of i) an architectural model describing the behavior of system hardware and software components, ii) a service-level model, and iii) a maintenance policy model. In this way, it can be clearly analyzed the

impact of the architectural choices on system performability while the end-user context is taken into account explicitly.

The separation of concerns property allows reuse of the architectural model when the same architecture is to be used in different environments. Obviously, changing the architectural model will lead to changing its links with the service-level and maintenance models. However, as these links are clearly identified and formalized, the changes can be performed more easily than when the various models are not separated. From the end-user perspective, this approach allows the reuse of the environmental models to compare different architectures for choosing the most suitable one for his/her own environment. Moreover, building the component model in a modular way favors reuse of some component submodels in the component model, which is of prime interest for both system manufacturers and the end-users.

1.5.4 Stepwise refinement approach

In [8] a stepwise approach for dependability modeling, based on Generalized Stochastic Petri Nets (GSPNs), is presented. It is a stepwise approach in the sense that the various dependencies defined in a model are taken into account at the right level of abstraction. The starting point is the construction of a functional-level model based on the system's specifications. The knowledge of the system's structure is captured in another model, the high-level dependability model, that may be successively refined according to three different aspects: component decomposition, state and event fine-tuning, and distribution adjustment (to take into account increasing event rates). Component refinement consists in replacing a component by two or more components. State/event fine-tuning consists in replacing, by a subnet, the place/transition corresponding to this state/event. The method of stages is finally used for the distribution adjustment: a transition is replaced by a subnet. Thanks to the refinement approach, the modelers can easily analyze and compare the dependability of one or several systems at the same level of modeling abstraction, also mastering the model complexity.

1.5.5 Reuse by inheritance

The concept of inheritance has been originally defined by [42] in the context of work-flow nets, and then it has been applied to large class of models. In the context of object-oriented programming languages, inheritance allows the definition of new classes based on already existing one(s), from which attributes and methods definitions are inherited. Very recently, [9] discusses the role of inheritance in the definition of GSPN models for dependability evaluations. Like a class P inherits from another class Q, similarly the modeler can derive a GSPN model p for objects

of type P starting from a GSPN model q already defined for objects of type Q, thus reusing input parameters and result definitions of q. This type of compositional approach mitigates the model complexity since it discriminates the part of the system that needs to be modelled in detail, (by means of sub-class models) from the rest of the system that can be modelled by using model component at an higher level of abstraction (by means of super-class models).

1.6 Decomposition/aggregation approaches

The most of these methods are characterized by a hierarchical decomposition approach to avoid the generation of large models. The overall model is decoupled in simpler and more tractable submodels, and the measures obtained from solution of the submodels are then aggregated to compute those concerning the overall model.

1.6.1 Component-based decomposition

[10] presents a decomposition approach for solving reliability models for systems with repairable components. The essential idea is to build one sub-model per component and to synthesize the system's reliability from sub-model solutions. The sub-model for a given component must contain system-state information to ensure that the repair process is active only for the system's up states. A natural construction procedure is to identify if the component in question is up or down, and to augment the sub-model with the system up/down information. This leads to a four-state submodel having two absorbing states. Each sub-model state is an aggregate of system-states. The authors show that the sub-models are Markovian but time-inhomogeneous since the transition functions vary with global time. This is an approximation because the parameters used in the sub-model are approximately derived from the monolithic model, since these functions are difficult to obtain.

1.6.2 Time scale decomposition

Although Generalized Stochastic Petri Nets (GSPNs) are highly effective in modeling concurrency, synchronization, and communication, they suffer from two major drawbacks. First, there is the explosion of the state space of its reachability graph when sufficient details are introduced in the model. Second, the presence of timed transitions having orders of magnitude difference in their firing rates lead to an underlying stiff Markov chain, which results in numerical instability in its solution technique.

In [11] is developed a technique that decomposes a GSPN model based on time scale. It is applicable to systems containing activities whose durations differ by several orders of magnitude. The given model can be decomposed into a hierarchical sequence of aggregated sub-nets each of which is characterized by a certain time scale. These smaller sub-nets are solved in isolation, and their solutions are combined to get the solution of the whole system. The aggregation at each level is done by assuming that the transitions included in the lower level are immediate transitions. At each level of the hierarchy, the current marking of an aggregated sub-net determines the number of tokens in the sub-net at the lower level, which are then analyzed to determine the rate of transitions in the aggregated sub-net.

1.6.3 Logical decomposition

A logical decomposition for stochastic rendezvous networks (SRNs) is presented in [12]. Stochastic rendezvous networks consist of tasks that take a random amount of time to complete and that may require the services of other tasks in order to complete. Each task is modeled separately, with the dependencies between tasks specified, and they communicate by messages in a request-wait-reply sequence which models a rendezvous. Compared to Petri Nets, the SRN framework is at a higher level of abstraction. Queueing and synchronization involving inter-task messages are implicit, so a given model can be stated much more compactly. The SRN thus applies the concepts of queueing networks, which have been of great power in modelling hardware servers, to software and hardware servers together.

1.6.4 Modular and hierarchical decomposition

In [13], the study of a railway interlocking system leads the authors to the definition of a modelling strategy based on a modular and hierarchical decomposition. The system is modeled at the various level of the hierarchy. Each layer has been structured for producing some results while hiding implementation details and internal characteristics: the output values from one layer are used as parameters of the next higher layer. In this way the entire modelling can be simply handled. Further, different layers can be modelled using different tools and methodologies: this leads to flexible and changeable sub-models so that one can vary the accuracy and detail with which specific aspects can be studied.

1.6.5 Phased decomposition

[14] addresses the analytical dependability modeling of Phased Mission Systems (PMS) by proposing a new methodology for their modeling and evaluation based

on a Markov Regenerative Stochastic Petri Nets (MRSPN) approach (a Petri Net is an MRSPN if its underlying marking process is a Markov regenerative process [36]). PMS are characterized by a sequence of phases in which the system configuration can change during operations. The existence of phases is a consequence of: i) diverse tasks to be performed, and ii) diverse environmental conditions, in different periods of system lifetime. The model of a PMS is seen as composed of two logically separate MRSPN: the System Net (SN), representing the system (its components, their interactions and their failure/repair behavior), and the Phase Net (PhN), representing the control part and describing the phase changes. In the SN, a single model is built for the whole mission, characterized by a set of phases without detailing the behavior of the system inside each phase. This allows easy modeling of a variety of mission scenarios by sequencing the phases in appropriate ways. The parameter values to be used in the SN model are obtained by solving the PhN models. These models detail the behavior of the system inside phases and are built and solved separately from each other. In particular, the state space of the Markov regenerative process never needs to be generated and handled as a whole, but rather the various subordinate intra-phase processes are separately generated and solved. Therefore, the computational complexity of the analytical solution is reduced to the one needed for the separate solution of the different phases.

1.6.6 Interaction-based decomposition

[15] proposes a decomposition approach for the solution of large stochastic Petri nets in which the overall model consists of a set of submodels whose interactions are described by an import graph. Each node of the graph corresponds to a parameterized stochastic Petri net submodel and an arc from submodel A to submodel B corresponds to a parameter value that B must receive from A. The authors show that the probability that a subnet is in a state satisfying a given condition, the average time a given condition remains satisfied, and the expected time until the subnet satisfies a given condition are three quantities that suffice for intercommunication among subnets for the net structure types that they define.

More recently, [16, 17] describe a new set of connection formalisms that reduce state-space size and solution time by identifying submodels that are not affected by the rest of a model, and solving them separately. The result from each solved submodel is then used in the solution of the rest of the model. The authors develop four abstractions that can be used to make connection models, and they involve passing a continuous-time random process, a discrete-time random process, a random variable, and an average value between the models. When these abstractions are applied, each submodel should have a smaller state space and

fewer time scales than the complete model.

1.7 Available modeling and solution tools

In this Section we provide a (non-exhaustive) list of tools that can help the user to automatically build and solve models (see [43] for a more complete list). They can be grouped in two main classes: the single-formalism/multi-solution tools, and the multi-formalism/multi-solution tools.

1.7.1 Single-formalism/multi-solution tools

The single-formalism/multi-solution tools are built around a single formalism and one or more solution techniques. They are very useful inside a specific domain, but their major limitation is that all parts of a model must be built in the single formalism supported by the tool. We only cite two sets of tools. The first set includes DSPNexpress [44], GreatSPN [45], SURF-2 [46], DEEM [47], TimeNET [48], and UltraSAN [49], that are tools based on Stochastic Petri Nets formalism and its extensions. They all provide analytic/numerical solution of a generated state-level representation and, in some cases, support simulation-based solution as well. Another set of tools uses other model specification approaches, sometimes tailored to a particular application domain. For example, DEPEND [50] and HIMAP [51] focus on evaluating the dependability of fault-tolerant computing systems, while TANGRAM-II [52] evaluates computer and communication systems using analytical/numerical methods.

1.7.2 Multi-formalism/multi-solution tools

The multi-formalism/multi-solution tools support multiple modeling formalisms, multiple model solution methods, and several ways to combine the models, also expressed in different formalisms. They can be distinguished with respect to the level of integration between formalisms and solution methods they provide.

In particular, some tools try to unify several different single-formalism modeling tools into a unique software environment. Examples are IMSE (Integrated Modeling Support Environment) [53], that is a support environment that contains tools for modeling, workload analysis, and system specification, IDEAS (Integrated Design Environment for Assessment of Computer Systems and Communication Networks) [54], that provides a broad range of modeling capabilities without the need to learn multiple interface languages and output formats, and FREUD [55], that focuses on providing a uniform interface to a variety of web-enabled tools.

A more recent model design framework is DRAWNET++ [56], that supports an Object-Oriented (OO) design process of system models and provides a graphical front-end to existing performance tools. Among its several OO features we cite the inheritance of model specifications, that allows to define model class hierarchies: new model classes are created from existing ones by absorbing their elements (nodes and/or edges), and overriding some of them or extending their properties. DRAWNET++ provides a GUI to any graph-based formalism, like Queuing Networks, Petri Nets, Fault Trees and Bayesian Networks. A custom XML language is used in the framework to represent the models and formalisms, and the interface towards the solvers consists of i) XSL filters to translate the XML representation of the models into the formats used by the external solvers, ii) scripts for running solvers, iii) filters to feed the results back into the XML models representation. Therefore, a different external solver can be used in the environment just writing proper filters.

In other tools, new formalisms, composition operators and solvers are actually implemented within a unique comprehensive tool. Though more difficult than building a software environment out of existing tools, this approach has the potential to much more closely integrate models expressed in different modeling formalisms. To the best of our knowledge, there are five main tools having these attributes: SHARPE, SMART, DEDS, POEMS and MÖBIUS.

SHARPE (Symbolic Hierarchical Automated Reliability and Performance Evaluator, [57]) is a tool for specifying and analyzing performance, reliability and performability models. It supports several modeling formalisms, both combinatorial (e.g. fault-trees and queuing networks) and state-space based (e.g. Markov and Stochastic Petri Nets), and several solvers performing steady-state, transient and interval of time analysis. The output measures of a model can be used as parameters of other models, thus facilitating the hierarchical combination of different model types.

SMART (Stochastic Model checking Analyzer for Reliability and Timing, [58]) is a multi-formalism modeling tool to study complex discrete-state systems. A single modeling study can be composed by different stochastic modeling formalisms, which can be solved using both numerical solution algorithms and discrete-event simulation techniques. It supports multiple interacting models in a variety of ways, from a simple hierarchical solution of submodels to a built-in fixed-point iteration mechanism.

DEDS (Discrete Event Dynamic System, [59]) is a toolbox for the construction of modular tools for functional and quantitative analysis of discrete event dynamic systems. The tools cooperate through an interface based on a general interchange format for model description: all the models built using different

formalisms are converted into a common "abstract Petri net notation" that, afterwards, can be solved using a set of functional and quantitative analysis approaches for Markovian models.

POEMS (Performance-Oriented End-to-end Modeling System, [60]) is a tool for modeling complex parallel and distributed systems. Different model types (with some restrictions) can be combined together creating a single multi-formalism platform. A system is specified by a multi-domain dependence graph in which the models are nodes and the dependencies are specified by unidirectional edges. The models interact through a common protocol that can be used to receive or send information.

MÖBIUS [61] tool has been developed following an integrated multi-formalism, multi-solution approach. The multi-formalism characteristic allows us not only to build models using different formalisms (such as SAN, PEPA, Buckets and Balls, and Fault Tree), but also to compose them obtaining a single multi-formalism model, that is a model composed by submodels described using different formalisms. It currently supports two classes of solution techniques: discrete event simulation (both transient and steady-state, applicable to any model), and state-based, analytical/numerical techniques. The simulator may be executed on a single workstation, or distributed on a network of workstations. In addition, it also includes a connection model formalism ([62]) that enables the passing of results between models, using a database to collect and share the produced results.

1.8 Summary

In this Chapter we gave an overview of the framework of the research we are contributing to. There is a general consensus that one of the most promising means of successfully dealing with large and complex models is to use a decomposition approach, in which the solution of the entire model is constructed on the basis of the solutions of its individual sub-models. In the rest of the dissertation we propose a set of techniques adopting this type of approach. We first develop two techniques specifically tailored for two particular classes of systems, a GPRS infrastructure (Chapter 2) and a hierarchical control system (Chapter 3), and then we propose a non domain-specific generalization (Chapter 4) and a case-study proving its feasibility (Chapter 5).

Chapter 2

QoS analysis for GPRS infrastructures

Service availability is an important aspect of telecommunication systems and it is tightly linked to the concept of Quality of Service (QoS), a general term defining the "level of satisfaction" perceived by an end user. This Chapter focuses on the specific application-domain of cellular networking environments, and we perform an analysis of the QoS perceived by the users camped in a General Packet Radio Service (GPRS) infrastructure. With respect to classical analysis of GPRS systems, we account for outage periods (due to the unavailability of a set of traffic channels) that induce congestion, and the application of reconfiguration actions (in terms of cell resizing) for congestion alleviation ([18]). The complexity of the system, composed of several partially overlapped cells that interact each other during the application of a Resource Management Technique (RMT), is here managed introducing a specific solution technique based on a decomposition/aggregation approach.

2.1 Introduction

Congestion events constitute a critical problem in the operational life of networked systems. A network is congested when the available resources are not sufficient to satisfy the experienced workload traffic, and this can occur for many reasons, such as in case of extraordinary events determining an increase of traffic, or in case of unavailability of some network resources because of malfunctions (outage). Careful management techniques are necessary, to alleviate the consequences of such phenomena. Building a resource management system to efficiently cope with congestion events in heterogeneous wireless networks has been the main contribution of the recently concluded European project IST-2001-38229 CAUTION++ [24].

Management techniques are usually equipped with internal parameters, whose values have to be properly assigned in accordance with the specific system characteristics. In order to support this "fine-tuning" activity, a model-based analysis has been promoted in CAUTION++ to analyze the behavior of the management techniques and to understand the impact of techniques and networks configuration parameters on properly identified Quality of Service indicators.

In this Chapter, the focus is on the General Packet Radio Service (GPRS) technology, which has been already analyzed in previous studies under more simplistic network configurations. An inspiring work is certainly [19], in which the authors analyze the dependability of a GPRS cell under outage conditions. Another work ([20]) evaluates the effects of outage periods on the service provision considering two GPRS cells partially overlapping (and then possibly interacting), and accounting for outage congestion treatment and outage recovery.

Now we perform a major extension and refinement to the previous studies dealing with a general GPRS infrastructure, where clusters of cells are considered, each cluster being realized through a number of partially overlapping cells. In case of an outage experienced by a cell in a cluster, a Resource Management Technique (RMT) is put in place to alleviate the congestion in the affected cell by distributing part of its traffic (users requests) on all the neighbor cells. In such a system context, we propose a methodology to evaluate the impact of congestion treatment on all the cells. The purpose of such analysis is to provide feedbacks for an optimal tuning of the parameters of the RMT (namely, the number of users to switch), so as to have the highest efficacy from its application towards resolving the congestion event. The definition of the general framework for the analysis of GPRS infrastructures has required a relevant effort, especially in the evaluation phase, due to the high level of complexity that can lead to very large state spaces for state-based analytical solutions or unacceptably long solution times for simulations. In order to efficiently solve the large and complex model capturing the network's behavior, we introduce a solution technique that follows a "divide and conquer" approach, in which the solution of the entire model is constructed on the basis of the solutions of the individual sub-models.

The rest of this Chapter is organized as follows. Section 2.2 gives a brief introduction to the GPRS technology, focusing on the Random Access Procedure mechanism through which the users compete to get a service. Section 2.3 presents the system context and the measures of interest. Section 2.4 introduces the solution technique selected to perform the QoS analysis, and provides the description of the required models. Then, in Section 2.5 the numerical results of the studies are presented and discussed. Conclusions are finally drawn in Section 2.6.

2.2 General Packet Radio Service

General Packet Radio Service (GPRS) is being specified as a part of the GSM phase 2+. The principal objectives of GPRS are high data rate, flexibility and efficiency utilization of scarce bandwidth across the air interface. The introduction of GPRS is a first step towards the full deployment of packet-data wireless networks. The use of the GSM circuit-switched transmission mode with data traffic, typically characterized by frequent alternation between activity and idle periods of the data source, results in an inefficient use of the scarce radio resources. In fact, in circuit switching allocation mechanisms, with high set-up time as in GSM, it is necessary to allocate a channel to a Mobile Station (MS) for all its transmission time without taking into account its real activity during this time. The GPRS introduces a packet oriented data service for GSM with a more efficient packet switching allocation mechanism. An important goal of the GPRS technology is to make it possible for GSM license holders to share physical resources on a dynamic, flexible basis between packet data services and other GSM services.

The GPRS allows several "logical channels" to share a physical channel (called Packet Data CHannel, PDCH) through time division multiplexing. PDCHs are associated with a single time slot of a Time Division Multiple Access (TDMA) frame (composed by eight time slots).

Among the logical channels, we focus on a specific channel dedicated to the uplink transmission of channel request: the Packet Random Access Channel (PRACH). When a mobile station needs to transmit, it has to send a channel request to the network through the PRACH. Since the network does not control the PRACH usage, the access method, based on a Random Access Procedure, may cause collisions among requests by different MSs. As it is observed during massive congestion events (such as New Year's Eve, important sporting events or natural disasters), the blocking on the PRACH may become a bottleneck of the system. The following Subsection summarizes the main characteristics of the Random Access Procedure (see [63] for a more detailed description).

2.2.1 The Random Access Procedure

The Packet Random Access CHannel (PRACH) is a GPRS logical channel used by Mobile Stations (MSs) to initiate packet transfers. On this channel MSs transmit access bursts with long guard times. The MSs get the access control parameters by listening to the Packet Broadcast Control CHannel (PBCCH). Such parameters are the number of maximum retransmissions M, the persistence level P and the parameters S and T. The MS is allowed to make a maximum of $M + 1$ attempts to send a Packet Channel Request message. At the beginning of the procedure a timer is set (to 5 sec). At the expiry of this timer, the procedure, if still active, is

aborted and a failure is indicated to the upper layer. The first attempt to send a Packet Channel Request can be initiated at the first possible TDMA frame containing PRACH. For each attempt, the mobile station extracts a random value R, and only if R is bigger than, or equal to, the persistence level P the station is allowed to send a Packet Channel Request. After a request is issued, the MS waits for a time, dependent on S and T. If it does not receive the Packet Downlink Assignment (or a Packet Queuing) a new attempt is tried, if it is still allowed to make one, otherwise a failure is notified to the upper layer. From parameters S and T, the MS also determines the next TDMA frame in which a new attempt is possible, should the previous one be unsuccessful and a new attempt still allowed. Under normal workload conditions, this retry mechanism is able to make the MS request to reach the Base Station Subsystem (BSS) with a very high probability. Once the MS request successfully reaches the BSS, traffic packet data channels, called slave PDCH, are allocated if available in the cell to transport users' data and transmission signaling. For what concerns data transfer, uplink and downlink channels allocation is completely independent and a MS can operate uplink and downlink data transfer simultaneously. Should the selected cell be not immediately able to allocate the PDCHs, the MS request may be put in a queue to wait for the first available resources. In case the request cannot be accommodated, a reject message is sent to the MS.

2.3 The system context and QoS indicators

We address a generic GPRS infrastructure, whose topology results in clusters of cells partially overlapping. To cope with congestion events, which may affect GPRS cells, e. g. due to a temporary lack of a number of traffic channels or to failures of their architectural components, we assume that appropriate RMTs are applied. Instead of focusing on a specific RMT, we consider the class of RMTs which operate congestion alleviation by reducing the traffic of the congested cells, which is redirected to the neighbor partially overlapping cells. That is, a cell resizing is performed, and those users in the area no more covered by the resized cell are assigned to a neighbor cell covering the area where the users are located (if such an overlapping cell exists; in general, some users can be lost because of the black-spot phenomenon). This implies that the user population attached to such neighbor cells increases, thus affecting the QoS of such cells. Once the congestion is overcome, a re-switching process is operated to restore the initial user population. The cells involved in the traffic reconfiguration applied through the RMT are the congested cell (called the *sending* cell) and a varying number of neighbor cells (called *receiving* cells). We call this set of cells a *congestion-effect* cluster. At a certain instant of time, a number of cells in the overall GPRS infras-

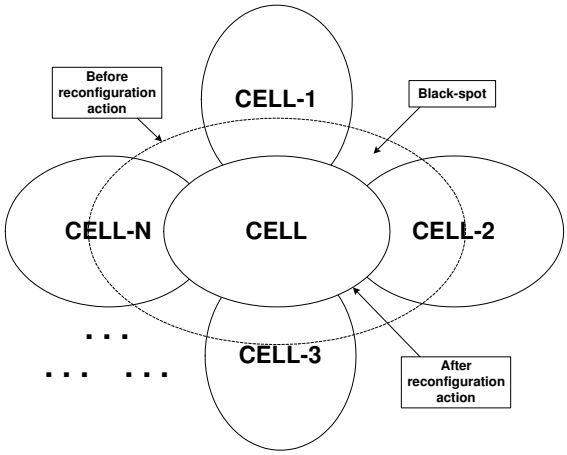

Figure 2.1: Congestion-effect cluster

tructure could be experiencing a congestion event. Since, as just said, the effects of applying a RMT are local to each *congestion-effect* cluster, the analysis of the congestion impact can be carried on independently for the different *congestion-effect* clusters. Concerning a single *congestion-effect* cluster, three scenarios could be theoretically observed: i) a *sending* cell overlaps with N *receiving* cells and no such *receiving* cells overlap with any other *sending* cell; ii) a *sending* cell overlaps with N *receiving* cells and at least one of such *receiving* cells overlaps with another *sending* cell; iii) two or more overlapping *sending* cells are surrounded by N *receiving* cells (not all overlapping with all the *sending* cells).

In many cases the congestion of a cell lasts a short time (e.g. in case the partial outage is caused by a software error that can be fixed in a few minutes restarting the software); then, the probability of having multiple congested cells in a *congestion-effect* cluster is low and it would be reasonable to neglect the cases ii) and iii) above, and restrict to consider scenario i) only. Therefore, in the following we will refer to the *congestion-effect* cluster scenario depicted in Figure 2.1. Anyway, accounting for the other situations would not require changing the principles at the basis of our methodology and the steps it is composed of, but necessitates some extensions to the developed models (especially for the case ii) where a cell may contemporary receive users from multiple *sending* cells, while case iii) would be simply treated considering the set of overlapping *sending* cells as a single *sending* cell).

As mentioned, we do not concentrate on a specific resource management technique, but we consider the class of techniques that ultimately result in a cell resizing or, equivalently, in a switching of users from one cell to another(s). The considered techniques are fully identified by the following characteristics:

1. the sending cell (CELL), that is the cell affected by outage;
2. the list of the receiving cells, that are the cells involved in the reconfiguration action (CELL-1, . . . , CELL-N);
3. for each receiving cell CELL-*i* (with *i*=1, . . . , N), the types of users to switch. A user may be: i) in the *idle* mode if he/she is not making any service request to the network system; ii) in the *active* mode if he/she is attempting to connect the network to get a service (it is trying to pass the random-access procedure), and finally iii) in the *in-service* mode if he/she is connected and awaiting to get the service completed;
4. for each couple of cells [CELL,CELL-*i*] and type of users, the maximum number of users to switch from CELL to CELL-*i* (see Figure 2.2). The actual number of switched users can be lower than this, and it happens when the outage duration is short and then the re-switching procedure starts before the switching procedure completed. The "observed users switching/re-switching distribution" represents the observed number of users switched/re-switched at varying of time. This distribution can not be computed a priori but only when the reconfiguration action is actually applied, since it depends on the availability of the users to be switched/re-switched, as we will better detail later. The number of users to re-switch corresponds to the number of users previously switched, as we suppose that all the users previously switched from CELL to CELL-*i* are then re-switched to the original cell.

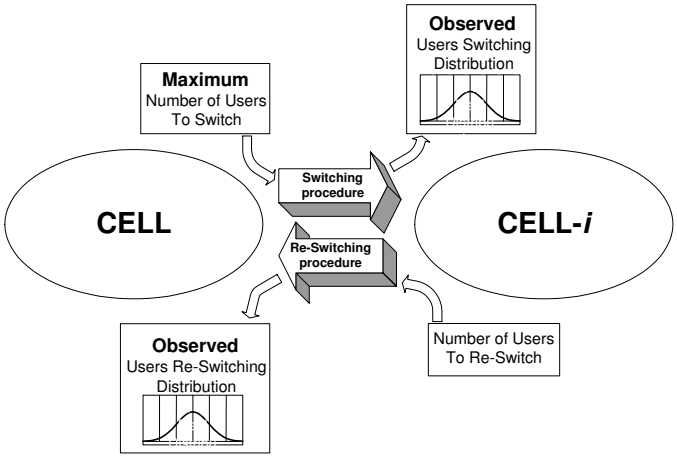

Figure 2.2: The interactions between two cells

The goal of our analysis is to investigate the effects of outage, congestion treatment and outage recovery on the service provision, with special attention on

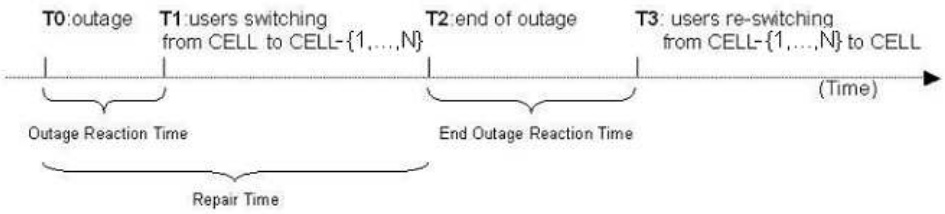

Figure 2.3: Scheduled Temporal Events

the user perception of the QoS. More precisely, we aim to analyze the behavior of the network during the following temporal events (see Figure 2.3):

- At time T0, an outage occurs in the central cell (CELL), thus determining congestion some time after;
- At time T1, the switching procedure starts, causing some users to be switched from the congested cell to its adjacent ones;
- At time T2, the outage ends;
- At time T3, a Resource Management System (RMS) reacts to the end of the outage and starts the re-switching procedure from CELL-1, . . . , CELL-N to CELL.

We are interested in the following service availability measures:

- the point-wise congestion function perceived by the users at varying of time (**PCf**), calculated as the ***percentage*** *of the unsatisfied users with respect to the total number of users in the cell*. An unsatisfied user is a user that is requiring a service but is not still served (active user);
- the total congestion indicator (**TCi**), inspired by [64], representing the ***average*** *congestion perceived by the users in a considered interval of time (E[PCf]).*

2.4 How to model and solve the system

The main problems in solving the model capturing the overall network's behavior are the time complexity (for the simulation) and the state space dimension (for the analytical solution), that rapidly increase if the number of receiving cells increases. Therefore, we investigated a decomposition approach, in which the solution of the entire model is constructed on the basis of the solutions of its individual sub-models. A simple, efficient solution would consist in splitting the

overall model of Figure 2.1 in a number of simpler sub-models to be solved separately, for example one for each cell. In this case, the main problem we have to cope with is the temporal dependency between the congested cell and each of the receiving cells during the switching/re-switching procedure. In fact, the observed users switching/re-switching distribution that describes the number of users switched/re-switched at varying of time is not known a priori, as it depends on the availability of users to be switched/re-switched (Figure 2.2). For example, suppose that a RMT performs an instantaneous cell resizing (the reconfiguration action) that corresponds to instantaneously switch a maximum of X active users from CELL to CELL-*i*. If, at switching time, only Y active users are available (with $Y < X$), the switching procedure will follow this "observed" distribution: Y active users will be instantaneously switched, while $X - Y$ users will be switched one by one as soon as they become available.

To properly cope with this temporal dependency, we decomposed the overall model of Figure 2.1 in a set of more simple sub-models, each one composed by the couple [CELL,CELL-*i*]. The temporal dependency disappears as each sub-model manages the switching/re-switching procedure between sending and receiving cells.

In our developed methodology, a top-down approach is adopted to move from the entire system description to the definition of more simple sub-models. Then, the model solution process follows a bottom-up approach: the solution of the entire model is constructed on the basis of the solutions of its individual sub-models.

It is a three step methodology. As it can be seen from Figure 2.4, we first decompose the overall model in N independent sub-models, each one composed by two cells: the first cell is always that affected by the outage (CELL), while the second is chosen from the other N receiving cells. Therefore, we solve N sub-models separately. From the solution of each single sub-model, we obtain two types of results for CELL-*i*:

- The QoS measures for CELL-*i* (a receiving cell), namely the percentage of unsatisfied users with respect to the total population;
- The "observed users re-switching distribution", that is the number of users re-switched from CELL-*i* to CELL as time elapses.

We note that in this first phase we do not obtain any information pertaining CELL, as each sub-model accounts for the re-switching procedure of only those users that have been previously switched from CELL to CELL-*i*, leaving out those users that have been previously switched from CELL to all other cells. In order to provide the QoS evaluations for CELL (the central cell), we perform another step in the solution technique. The "observed users re-switching distributions" from

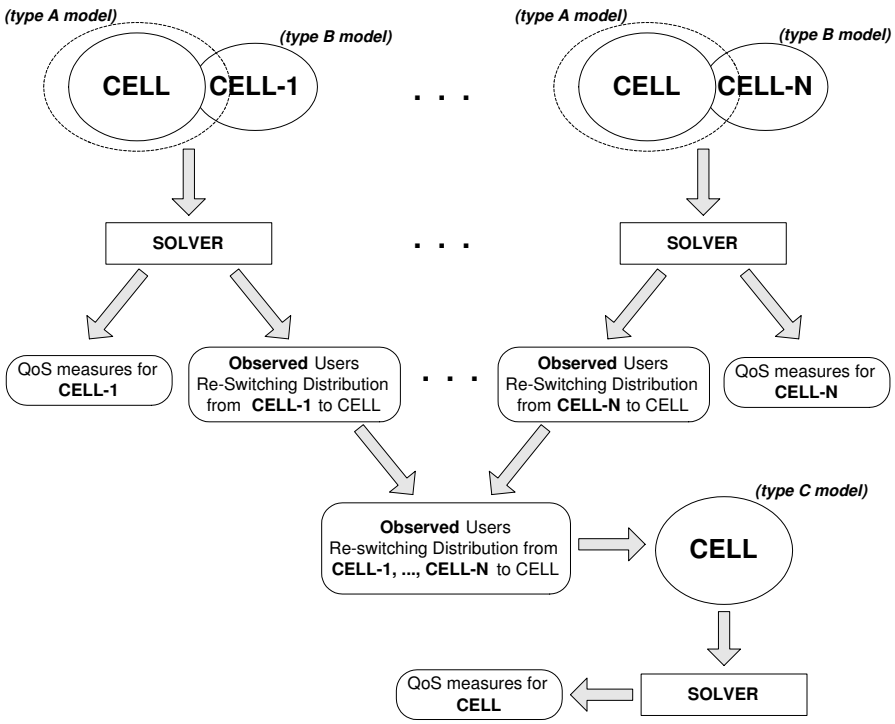

Figure 2.4: Modeling and solution technique

each CELL-*i* to CELL are collected and combined, obtaining the "observed users re-switching distribution" from CELL-1, . . ., CELL-N to CELL. Finally, this distribution is given as input to another model (that represents the behavior of the central cell considering the re-switching procedure from all the neighbor cells to the central one) whose solution provides the QoS measures for CELL. We note that this last model requires the "observed users re-switching distribution" as input, while the "observed users switching distribution" is not explicitly required. This happens because we suppose that a receiving cell could not refuse an incoming user, and then the switching procedure only depends on the behavior of CELL (the sending cell).

2.4.1 The models needed

In order to apply the methodology depicted in Figure 2.4 we need to construct three types of models only: type A, type B and type C. All the models are derived using stochastic activity networks (SAN) formalism.

These models can be obtained as a specification of the template model of Figure 2.5 representing an abstract view of a generic GPRS cell. The "internal GPRS cell model" was originally defined in [19], and it models the behavior of a GPRS

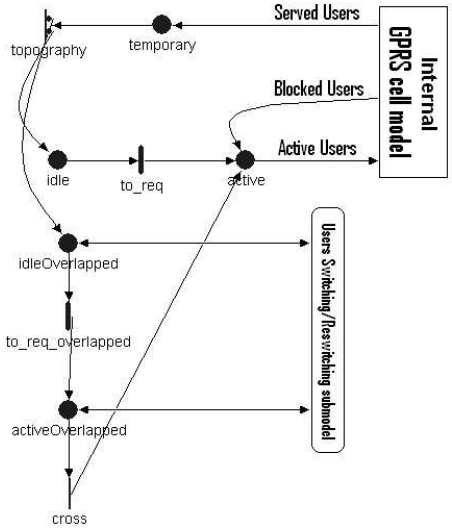

Figure 2.5: A generic GPRS cell

cell during the Random Access Procedure, when users compete to get a free channel.

The sub-model capturing the interactions between the central cell and the neighbor cells is the "users switching/reswitching sub-model". This sub-model has to be specified in order to:

- represent the behavior of the congested cell (CELL) during outage, cell resizing and outage recovery (type A model of Figure 2.4);
- represent the behavior of a receiving cell (CELL-*i*) during the resizing of the congested cell (type B model of Figure 2.4);
- represent the behavior of the congested cell (CELL) during outage, cell resizing and outage recovery using the provided "observed users re-switching distribution" (type C model of Figure 2.4).

The generic model of Figure 2.5 works as it follows. When a user has been served, a token exits from the "internal GPRS cell model". This generic user has to be mapped (using the `topography` activity) in the overlapping area of the cell (place `idleOverlapped`) or in the non overlapping one (place `idle`), in accordance with the topography of the network. The probability that a generic user is mapped in the overlapping area is dynamically calculated considering the original number of users in the overlapping area and the overlapping users that have been switched to the other cells. When an idle user requests a new service, he/she becomes active and enters in the "internal GPRS cell model" that simulates the

random access procedure of a GPRS cell. Finally, we note that the users switching and re-switching procedure affects only the users in the overlapping area, both in idle and in active mode.

The "internal GPRS cell model" and of the "users switching/reswitching submodel", depicted in Figure 2.5 as black-boxes, will be detailed in the rest of this Section.

Modeling assumptions

The random access procedure model has been defined under the following assumptions concerning the configuration of the GPRS:

1. Each cell contains a constant number of users (except that during the reconfiguration actions), uniformly distributed inside the cell. In other words we are realistically assuming that, during normal conditions, the users' movements are in equilibrium with a stable average population;

2. All users belong to the same priority class (they are indistinguishable from the point of view of generated traffic). This assumption is realistic in a scenario in which only basic GPRS services are provided, and no specific QoS level are negotiated; this is indeed the current situation of GPRS services;

3. Once a request has been made from a user, he cannot abort it but has to wait until the service is provided. This choice is not realistic, but conservative; it forces a higher load on the system, inducing a less favorable situation;

4. Each traffic channel is allocated to a single user at a time, who will retain it until the completion of his data transmission; concurrent usage of traffic channels and multi-slot assignments to a single user are not considered;

5. It is allowed to queue the request when all channels are occupied (through an Access Grant Reservation), accordingly to the ETSI standard specification of GPRS.

In the following we present the modelling assumptions concerning the user switching/reswtiching process and the outage. These restrictions do not impair the validity of our work, as they characterize a typical real GPRS outage scenario.

1. An outage is experienced by the central cell as a consequence of traffic channels malfunctions. If the damage does not affect all the channels, the cell doesn't stop working, but enters a degraded operational mode. The repair time is assumed to follow a deterministic distribution;

2. The users switching/reswitching procedure applies only to users in the overlapping area (see Figure 2.1) and doesn't affect the in-service users.

3. Once they become available, the users can be switched/re-switched instantaneously, that is their transmissions can be instantaneously camped in the neighbor cell;

4. The users lost during the switching procedure (e.g. because of non totally overlapping cells) are subtracted from CELL before the switching procedure starts. The users lost (both in idle and active mode) are instantaneously re-added as idle users to CELL at re-switching time;

5. When the outage occurs, all the cells composing the congestion-effect cluster are working in a steady-state condition.

2.4.2 The "internal GPRS cell model"

The "internal GPRS cell model" is shown in Figure 2.6 and it is here described as reported in [19, 65].

- Tokens in the place `idle` and `idleOverlapped` represent, respectively, those users outside the overlapping area and inside the overlapping area between CELL and CELL-i that have sent successfully their up-link data. After some time, accounted for by the timed (exponential) activities `to_req` and `to_req_overlapped` a user issues a new request and a token is moved to the corresponding active places (`active` and `activeOverlapped`).

- The block starting with the instantaneous activities `req` and ending with the input gate `control` represents the dynamics of the random access procedure. The activity `req` states the maximum number of attempts a user is allowed to make in sending an Access Burst. It has one case for each possibility; the associated probabilities have been derived on the basis of the parameters M, P, S, T and the timer. Tokens in places `ready1`, ..., `ready8` represent the number of users allowed to make a maximum of 1, ..., 8 attempts, respectively. The instantaneous activities `check_p1`,..., `check_p8` model the persistence level. If the user passes the persistence level, he can send an Access Burst and moves into the place $\texttt{try}_i$, otherwise he moves into the correspondent place $\texttt{fail}_i$. Should a user consume all his assigned attempts to make his request, or should the time-out regulating the maximum allowed time for making a request (set to 5 sec) expire, the user is moved into the place `block`. A blocked user will do a new attempt

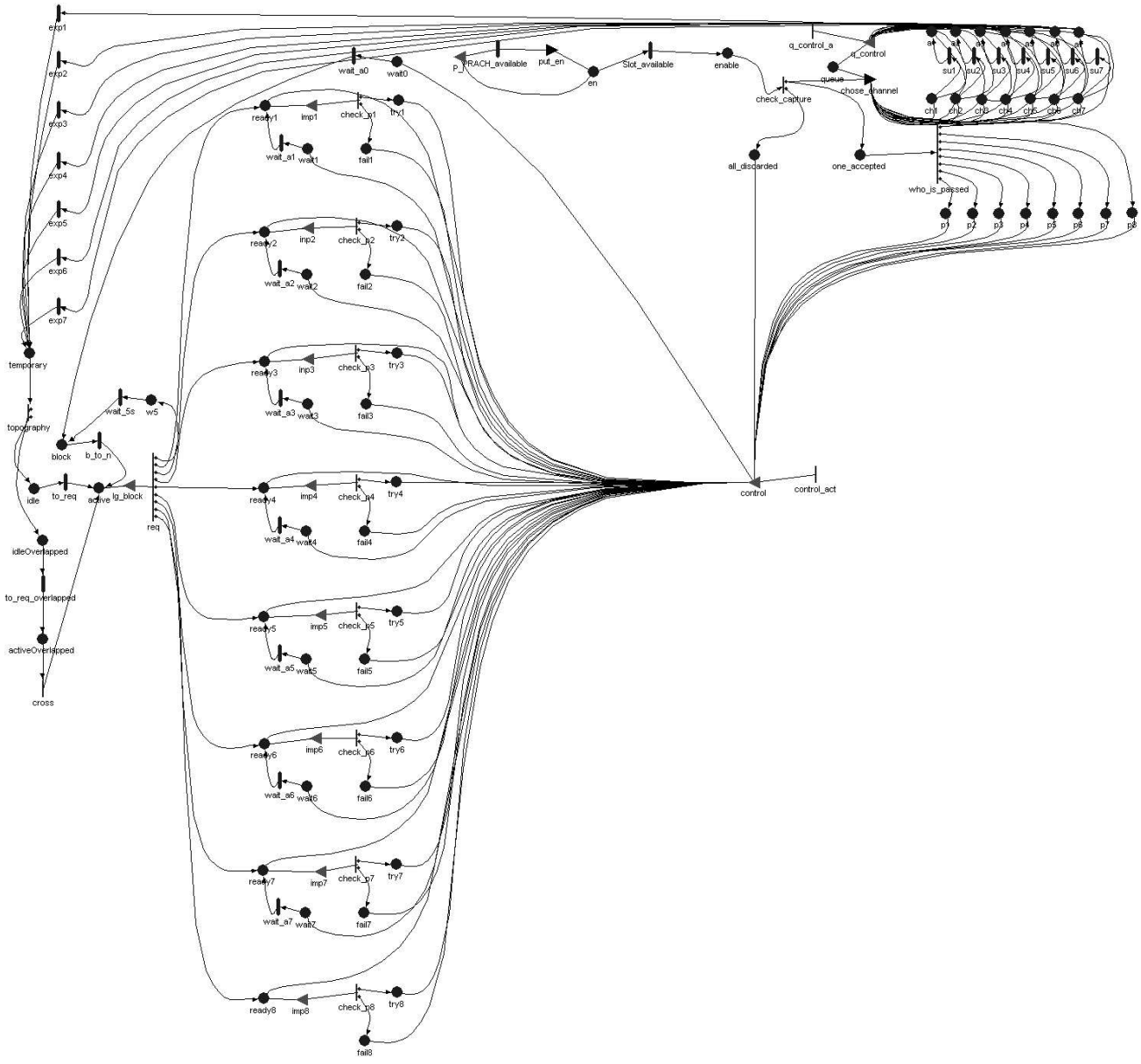

Figure 2.6: The “internal GPRS cell model”

after a time sampled from the timed activity b_to_n, having exponential rate and taking into account Automatic Retransmission Time (ART). The place w5 and the activity wait_5 take into account those users that haven't been assigned any attempt, because they will always fail the persistence level. According to the standard specification, they have to wait 5 seconds before moving in the place block.

- The instantaneous activity check_capture checks, stochastically, if there is a successful receipt of one Access Burst; if yes, a token is placed in one_accepted, unless the queue is full and there is no available traffic channel, in which case a token is put in all_discarded. The instantaneous activity who_is_passed fires when there is a token in one_accepted and it allows to choose which level the accepted Access Burst comes from, placing a token in one of the places p1, ..., p8 (each Access Burst at each level has the same probability to be the accepted one). The input gate control and the activity control_act properly update the places recording the residual tries made available to the other concurrent requests (places ready1, ..., ready8, try1, ..., try8, fail1, ..., fail8, wait_a0, ..., wait_a7 and p1, ..., p8).

- When there is a successful receipt of one Access Burst and there is a free channel (that is at least a free pair between ch1-a1,...., ch7-a7), the output gate choose_channel puts a token in one of the places ch1, ..., ch7 otherwise it puts a token in the place queue. The timed activities su1, ..., su8 simulate the set-up time of a radio link to send user data. The activities exp1, ..., exp7 follows a uniform distribution and represent the data sending time.

- A token (generic user) that reaches the temporary place is instantaneously moved through the topography activity in the place idle or idleOverlapped following a dynamically calculated distribution.

- The sub-net enclosing the timed activities PRACH_available and slot_available, and the places en and enable, models the multi frame on the PDCH.

- A token in the place queue represents a pending request waiting for uplink channel reservation. The instantaneous activity q_control_a fires when a channel is released and there are pending requests in the queue. When activity q_control_a fires, the input gate q_control moves a token from queue to a place among ch1, ..., ch7, corresponding to the available channel.

2.4.3 Type A model: the "users switching/reswitching sub-model" for CELL

Our purpose is to construct a GPRS cell model that represents the behavior of the congested cell (CELL) during outage, cell resizing and outage recovery (type A model). In Figure 2.7, we describe the "users switching/reswitching sub-model" that specifies the general GPRS cell behavior as required. The black line separates the components belonging to the "generic GPRS cell model" (on the left) from those belonging to the added model (on the right).

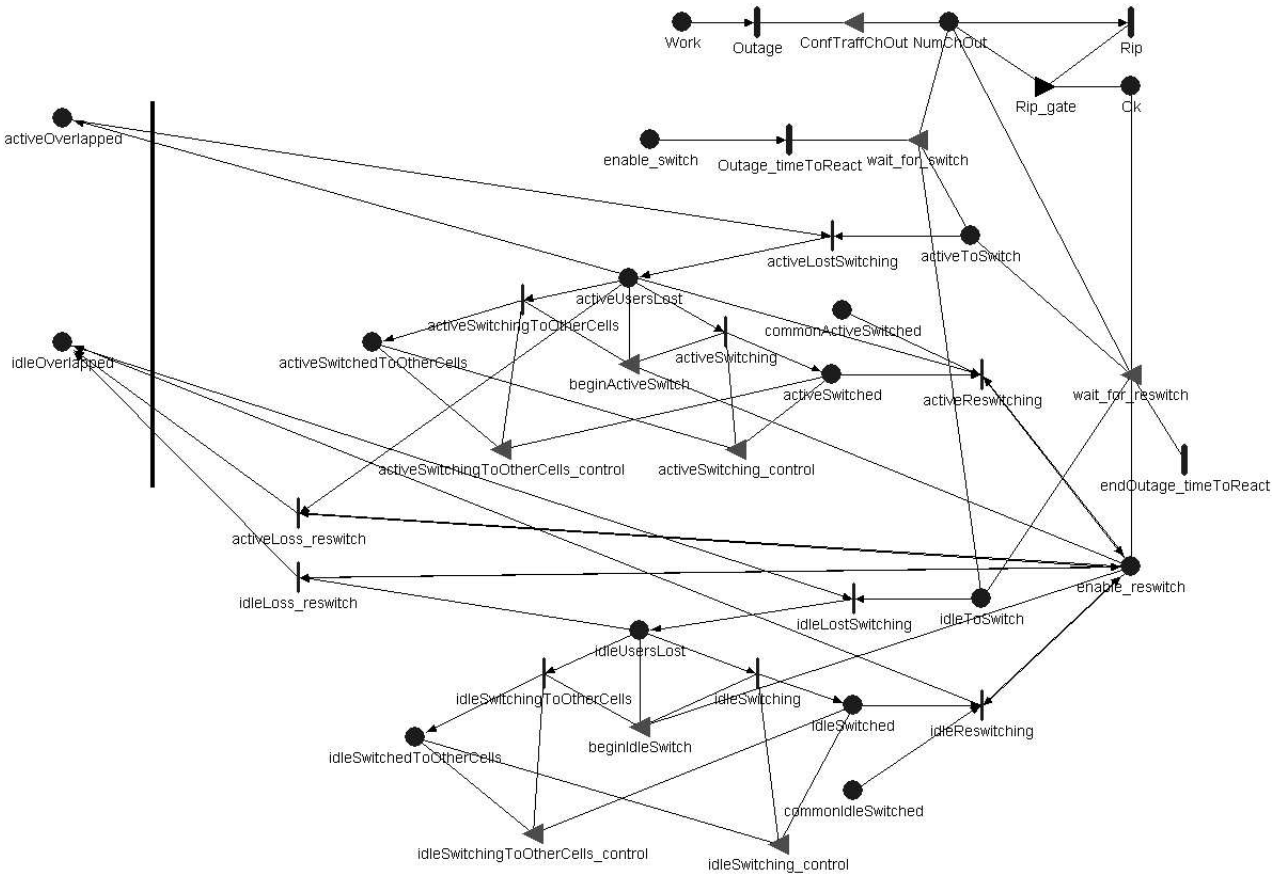

Figure 2.7: "users switching/reswitching sub-model" for CELL

Here, we sketch the model behavior following the temporal events of Figure 2.3.

- Before time T0, the system is in steady-state and one token is in place `Work` (the system works properly).

- At time T0 the `Outage` activity fires (deterministic distribution) and some tokens are put in place `NumChOut`, whose marking represents the number of unavailable traffic channels. From this moment, the system is working in a degraded manner. In order to actually cause the unavailability of the traffic channels in the central cell, the output gates `choose_channel` and `q_control` of the "internal GPRS cell model" of Figure 2.6 have been modified to assign a free channel only if the number of free pair between `ch1-a1`,...., `ch7-a7` is greater than the marking of place `numChOut`.

- At time T1 the deterministic activity `Outage_timeToReact` fires (in *outageReactionTime* sec.), representing the time necessary for the system to react to the outage (it's equal to T1-T0). At firing time, the input gate `wait_for_switch` puts the established number of tokens (users) in place `activeToSwitch` (maximum number of active users to switch) and `idleToSwitch` (maximum number of idle users to switch) and then the switching procedure starts. Following the assumptions, we first loose the users in the non-overlapping area (black-spot phenomena), and then the tokens (users) are put in place `activeUsersLost` and `idleUsersLost`, representing the number of active and idle users lost during the switching procedure. The active or idle users successfully switched from CELL to CELL-*i* (observed number of switched users) are represented by tokens in place `activeSwitched` and `idleSwitched`, respectively (these two places are shared with the CELL-*i* model through the Join operator). Finally, tokens in place `activeSwitchedToOtherCells` and `idleSwitchedToOtherCells` represent, respectively, the number of active and idle users really switched from CELL to all other receiving cells except for CELL-*i*.

- At time T2 the deterministic activity `Rip` fires (in *riptime* sec.), representing the time necessary for the system to be repaired (it's equal to T2-T0). At firing time the outage ends and CELL restarts to work properly (the marking of place `numChOut` is set to zero).

- At time T3 the deterministic activity `endOutage_timeToReact` fires (in *outageEndReactionTime* sec.), representing the time necessary for the system to react to the end of the outage (it's equal to T3-T2). At firing time, the mark of the place `enable_reswitch` (shared with the CELL-*i* model) is set to 1 and the re-switching procedure starts. The activities `activeLoss_reswitch` and `idleLoss_reswitch` are enabled and the users previously lost are instantaneously reinserted in the `idleOverlapped` place. Moreover, the `activeReswitching` and `idleReswitching` activities are enabled and then the users in places

commonActiveSwitched and commonIdleSwitched (shared with the CELL-*i* model and representing, respectively, the active and idle users to be re-switched from CELL-*i* to CELL) are sent, respectively, to place activeOverlapped and idleOverlapped. The re-switching procedure ends when the same number of users previously switched from CELL to CELL-*i* has been re-switched to the original cell. We finally note that the users switched from CELL to all the other receiving cell except for CELL-*i* are not re-switched.

2.4.4 Type B model: the "users switching/reswitching sub-model" for CELL-*i*

The purpose is to construct a GPRS cell model that represents the behavior of a receiving cell (CELL-*i*) during the resizing of the congested cell. In Figure 2.8, we show the sub-model that specifies the general GPRS cell behavior as required. The vertical black line separates the components belonging to the "generic GPRS cell model" (on the left) from those belonging to the added model (on the right).

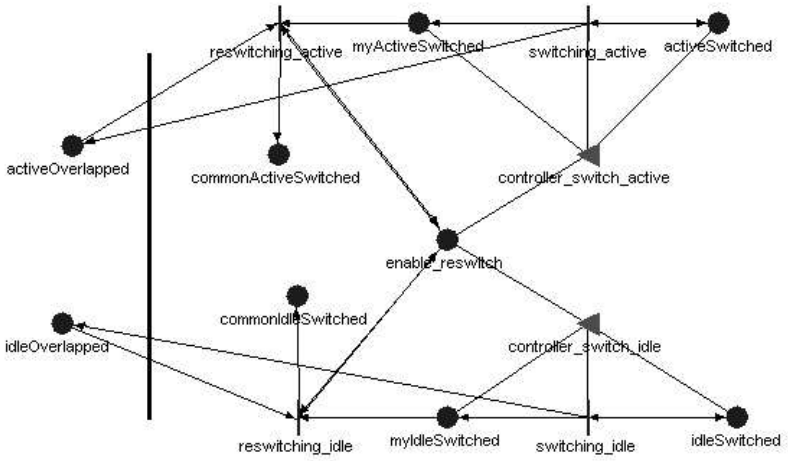

Figure 2.8: "users switching/reswitching sub-model" for CELL-*i*

Tokens in place activeSwitched (or myActiveSwitched) and idleSwitched (or myIdleSwitched) represent, respectively, the number of active and idle users really switched from CELL to CELL-*i*. The input gate controller_switch_active keeps the number of tokens in activeSwitched equal to the number of tokens in myActiveSwitched, until the re-switching procedure starts. The input gate controller_switch_idle performs the same action for the idle users. The

enable_reswitch place contains one token if the re-switching procedure is enabled, zero otherwise. Tokens in places commonActiveSwitched and commonIdleSwitched represent, respectively, the active and idle users re-switched from CELL-*i* to CELL.

Here, we briefly describe the model behavior following the temporal events of Figure 2.3.

- Before time T0, the system is in steady-state.
- At time T1, the switching procedure from CELL to CELL-*i* starts and then some tokens arrive in places activeSwitched and/or idleSwitched, that represent the number of active and/or idle users really switched from CELL to CELL-*i*. Places myActiveSwitched and myIdleSwitched follow the respective variations, thanks to the input gates controller_switch_active and controller_switch_idle.
- At time T3, the mark of the place enable_reswitch is set to 1 and the re-switching procedure starts. The users re-switched from CELL-*i* to CELL are available in place commonActiveSwitched and commonIdleSwitched. The re-switching procedure ends when places myActiveSwitched and myIdleSwitched are empty.

2.4.5 The overall model for the couple [CELL, CELL-*i*]

Previously, we showed how to construct the models for CELL and CELL-*i* specifying the behavior of the "internal GPRS cell model" that represents the random access procedure of a generic GPRS cell. In this Subsection we describe how to construct the overall model for the couple [CELL, CELL-*i*]. This model will be used in the resolution technique of Figure 2.4 to solve each couple of [type A model, type B model] in the first step of the methodology.

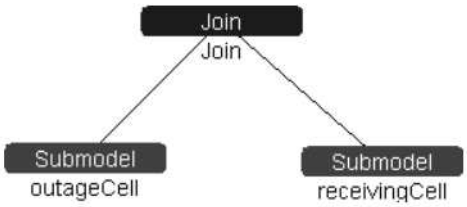

Figure 2.9: [CELL, CELL-*i*] model

The overall model of Figure 2.9 is composed by two cells: the outageCell (CELL), that is the cell affected by the outage, and a receivingCell (CELL-

i), that is a cell that receives some users from the congested cell. These two models interact each other through some shared places defined in the Join operation [4]. The shared places are the following: `activeSwitched`, `idleSwitched`, `commonActiveSwitched`, `commonIdleSwitched` and `enable_reswitch`. Each sharing consists of a non-empty set of state variables of children of the Join node.

2.4.6 Type C model: the "users switching/reswitching sub-model" for CELL using the provided "observed users re-switching distribution"

The purpose is to construct a GPRS cell model that represents the behavior of the congested cell (CELL) during outage, cell resizing and outage recovery (type C model of Figure 2.4). It uses the provided "observed users re-switching distribution" to reconstruct the re-switching procedures of all the neighbor cells.

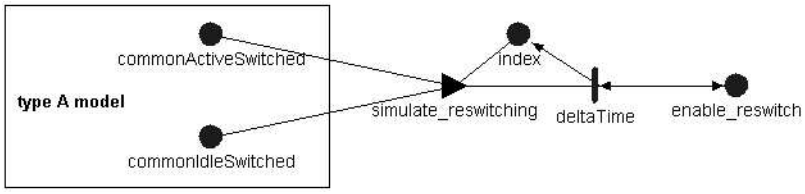

Figure 2.10: "users switching/reswitching sub-model" for CELL using the provided "observed users re-switching distributions"

As we can see from Figure 2.10, the model is derived from the type A model by adding a very simple SAN that is responsible to represent the effect of the re-switching procedures from all the receiving cells CELL-1, . . ., CELL-N to CELL. The output gate `simulate_reswitching` has the following output function:

```
double activeToReSwitch[]={...};
double idleToReSwitch[]={...};
commonActiveSwitched→Mark()=commonActiveSwitched→Mark()+
   activeToReSwitch[index→Mark()+1];
commonIdleSwitched→Mark()=commonIdleSwitched→Mark()+
   idleToReSwitch[index→Mark()+1];
```

The variables `activeToReSwitch` and `idleToReSwitch` are defined as two arrays, and their values represent, respectively, the number of active and idle users to re-switch from CELL-1, . . . , CELL-N to CELL at different instants of time. These arrays are the discrete versions of the "observed users re-switching

distributions". If δ is the delay associated to the deterministic distribution `deltaTime`, activeToReSwitch[k] (or idleToReSwitch[k]) is the number of active (or idle) users to be re-switched at time T3 + $\delta(k+1)$, where T3 is the time in which the re-switching procedure starts (or, equivalently, the time in which the place `enable_reswitch` is set to 1). Therefore, δ is the sampling period of the "observed users switching distributions". The shorter is δ, the higher is the accuracy of the two distributions. As a detail we finally note that we multiplies δ for $(k+1)$ because, when `deltaTime` fires, the output function of the gate `simulate_reswitching` is executed before that one token is added to the place `index`.

2.4.7 About effectiveness

The major characteristic of this technique is its capability to manage the complexity of the overall model, as we provide the solutions solving N+1 sub-models only and combining some basic QoS measures. In case of state-based analytical solution, the state-space explosion problem is drastically reduced thanks to the lower number of states generated for each individual sub-model. In case of simulation, the major advantages are related to:

- the mitigation of the stiffness-problem, if the submodels to be simulated during Step 1 and 3 have less time scales than the monolithic model. This property could be extremely useful in dealing with an heterogeneous network composed by cells of different technologies, e.g. GPRS and UMTS (Universal Mobile Telecommunications System);
- the decrement of the overall solution time, since the N sub-models constituted by the couple [CELL,CELL-*i*] in Step 1 can be solved concurrently. This favors the scalability of the method, which can easily deal with high numbers of receiving cells;
- the alleviation of the memory requirements for the simulator, as the sizes of the sub-models to be solved are reduced thanks to the models decomposition.

Although both analytical and simulation solution methods can be applied, in this study we adopt the simulation approach to numerically solve the sub-models obtained applying our methodology, using the simulator offered by the Möbius tool. The main advantage in using the simulation is that it allows to represent real system conditions better than analytical approaches do (e.g., to use distribution functions more realistic than the exponential one).

Symbol	Description	Values
GPRS_Channel	Number of available GPRS channels	3
GPRS_Channel_Out	Number of GPRS channels affected by outage	2
QL	Length of the queue	2
Treq	User inter-request time, following an exponential distribution	60 sec. (average)
Rsize	Request size, following a uniform distribution	[1000-1600] byte
totalNumUsersCELL	Total number of users camped in CELL	180
numUsersCELL_OvCELL1	Number of users camped in the overlapping area between CELL & CELL1	60
numUsersCELL_OvCELL2	Number of users camped in the overlapping area between CELL & CELL2	50
numUsersCELL_OvCELL3	Number of users camped in the overlapping area between CELL & CELL3	40
numActiveUsersToSwitchToCELL1	Number of active users to switch from CELL to CELL1	0, 30, 60
numActiveUsersToSwitchToCELL2	Number of active users to switch from CELL to CELL2	0, 25, 50
numActiveUsersToSwitchToCELL3	Number of active users to switch from CELL to CELL3	0, 20, 40
totalNumUsersCELL1	Total number of users camped in CELL1	140
totalNumUsersCELL2	Total number of users camped in CELL2	170
totalNumUsersCELL3	Total number of users camped in CELL3	200
outageReactionTime	Time between outage and the beginning of the switching procedure	variable

Table 2.1: Relevant parameters and their values

2.5 Model evaluation

We perform a transient analysis in the interval of time from the occurrence of an outage (time T0) to the new system steady-state after the outage repair, using the simulator provided by Möbius tool [61].

2.5.1 Settings for the numerical evaluation and the Analyzed Scenarios

We analyze a GPRS network composed of one central cell (CELL) and three partially overlapping cells (CELL1, CELL2 and CELL3). In Table 2.1 we detail the values we assigned to the main parameters of each cell. All the four cells have the same number of traffic channels (three) but different user populations; therefore, each cell has a different workload level at steady-state.

We analyzed two scenarios, which have been set up in order to tune the following two parameters of a resource management technique: *activeUsersToSwitch*, that is the total number of active users to switch from CELL to CELL1, CELL2 and CELL3 (*activeUsersToSwitch*=$\sum_{i=1}^{3}$ numActiveUsersToSwitchToCELLi), and *outageReactionTime*, that is the time necessary to the Resource Management System to react to the outage.

- SCENARIO 1: The fine-tuning is performed in terms of the number of active users to switch from CELL to each other cell. In particular, we consider three cases: i) the case where no cell resizing is performed (no users switching), ii) the case where the cell resizing involves 50% of the users in

the overlapping area (active users to switch = 75), and iii) the case where the cell resizing involves 100% of the users in the overlapping area (active users to switch = 150). Moreover, we set the *outageReactionTime* parameter to 30 seconds and assumed that 10% of the switched active users are lost during the reconfiguration action.

- SCENARIO 2: The number of active users to switch from CELL to the other cells is set to 75 users (30 to CELL1, 25 to CELL2 and 20 to CELL3). The focus in this scenario is on evaluating the impact of the time necessary to the Resource Management System to apply a traffic reconfiguration after the occurrence of an outage. So, the parameter under tuning is *outageReactionTime*, for which three values have been considered: 15, 45 and 75 seconds. This performance indicator is useful to set a maximum value on the time the RMS is allowed to spend to elaborate a reaction to the observed overload.

We suppose that the switching and re-switching procedures are instantaneous. Moreover, we suppose that the partial outage affecting the central cell consists of a software error that reduces the number of available traffic channels from 3 to 1, and we set the outage duration to 120 seconds (average time needed to restart the software). The *outageEndReactionTime* parameter (the time that occurs between the end of the outage and the users re-switching) is set to 15 seconds (typical real value). In all the simulations we choose a relative confidence interval of 0.1 and a confidence level of 0.95, that is in the 95% of the times, the mean variable will be within 10% of the mean estimate. Moreover, we choose a value for δ such that considering lower values would have not lead to significant improvement in the accuracy of the results (δ=4 sec.).

2.5.2 Numerical evaluation

In this Section we show the results obtained from the simulations, both concerning the Pointwise Congestion function (PCf, on the Y-axes) and the Total Congestion indicator (TCi, in the labels of the figures). In all the figures plotting the simulation results, the time interval on the x-axis starts at time 200 sec. (the outage occurrence time) and ends at time 556 sec. (the time the new steady-state is reached in all the cells). The labels T0, T1, T2 and T3 on the x-axis have the same meanings as in Figure 2.3.

Evaluation in scenario 1: tuning of parameter 'activeUsersToSwitch'

Figures 2.11, 2.12 and 2.13 show, respectively, the congestion perceived by the users (the Point-wise Congestion function) in CELL1, CELL2 and CELL3 at vary-

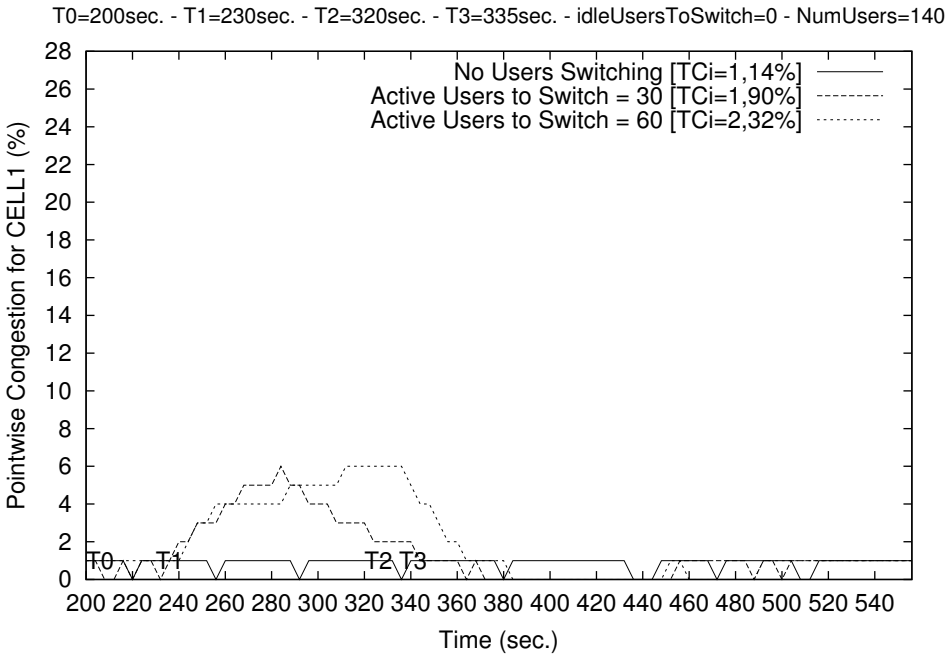

Figure 2.11: Congestion perceived in CELL1

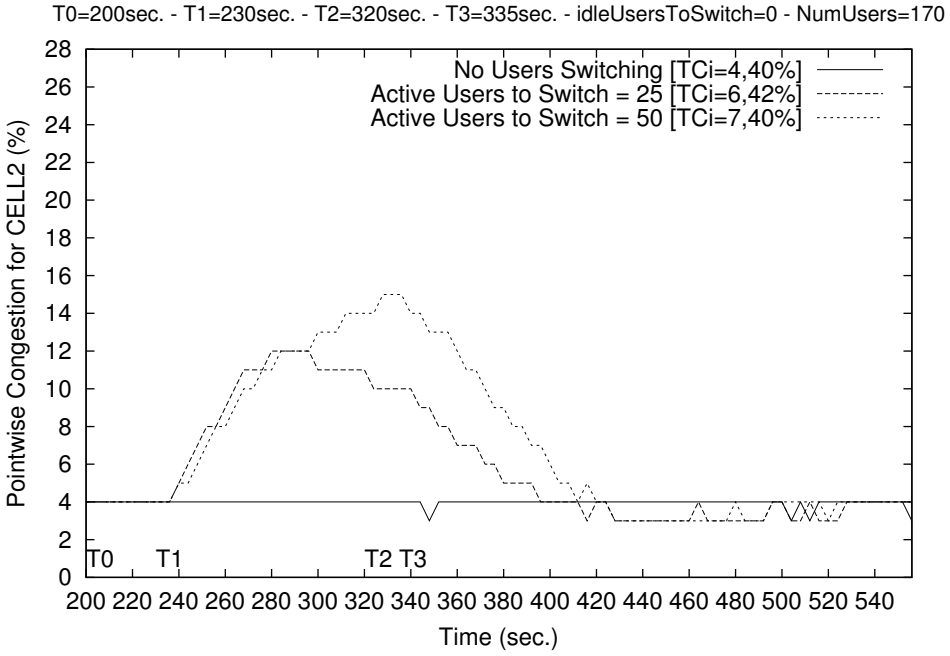

Figure 2.12: Congestion perceived in CELL2

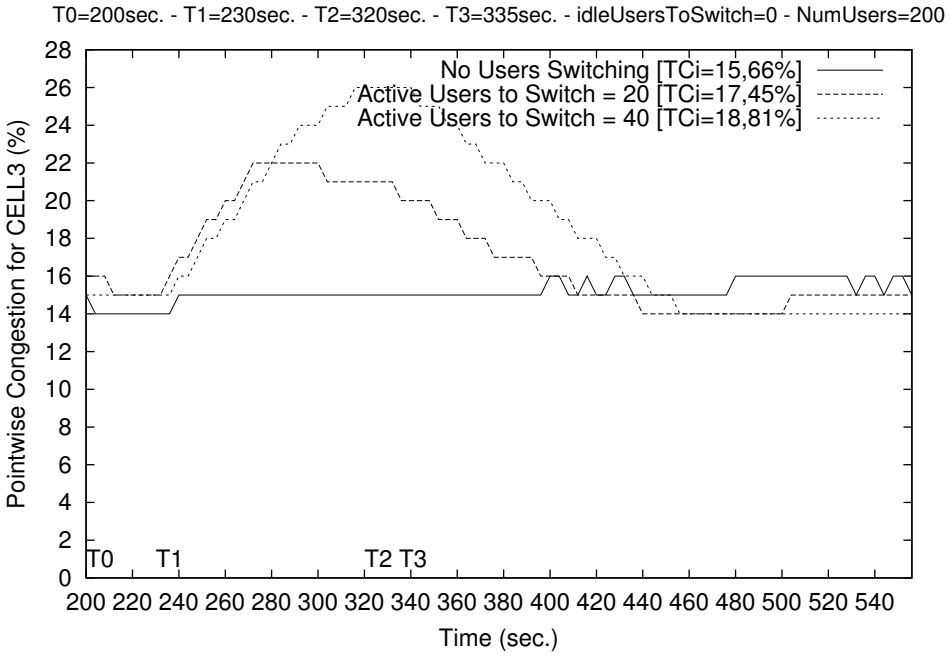

Figure 2.13: Congestion perceived in CELL3

ing of the number of the active users to switch (0%, 50%, 100% of the number of users in the overlapping area). Obviously, the TCi value increases when we increase the value of the *activeUsersToSwitch* parameter. We note that the congestion level at steady state (time T0) is about 1% for CELL1, 4% for CELL2 and 14% for CELL3, mainly because of a different number of users camped in. After time T1 (the switching time), the congestion initially increases, but decreases immediately after. This happens when the receiving cell is not congested and, then, can absorb the added traffic. Moreover, the traffic overload induced in CELL3 has the most negative impact, as the congestion level at steady-state is the highest.

Figure 2.14 shows the congestion perceived by the users in the cell affected by the outage at varying the number of the active users to switch from this cell to all the adjacent cells. From the figure we note that if we increase the total number of active users to switch from 75 to 150, the TCi value remains the same. This happens, in general, when the system tries to switch "too many" users and then the negative effects due, for example, to the augmented number of lost users is equivalent to the positive effects due to the augmented number of switched users. At time T1 the switching procedure starts and the perceived congestion is beneficially affected by the actuation of the technique in a very short amount of time. At time T2 the outage ends, CELL starts working properly and the congestion rapidly

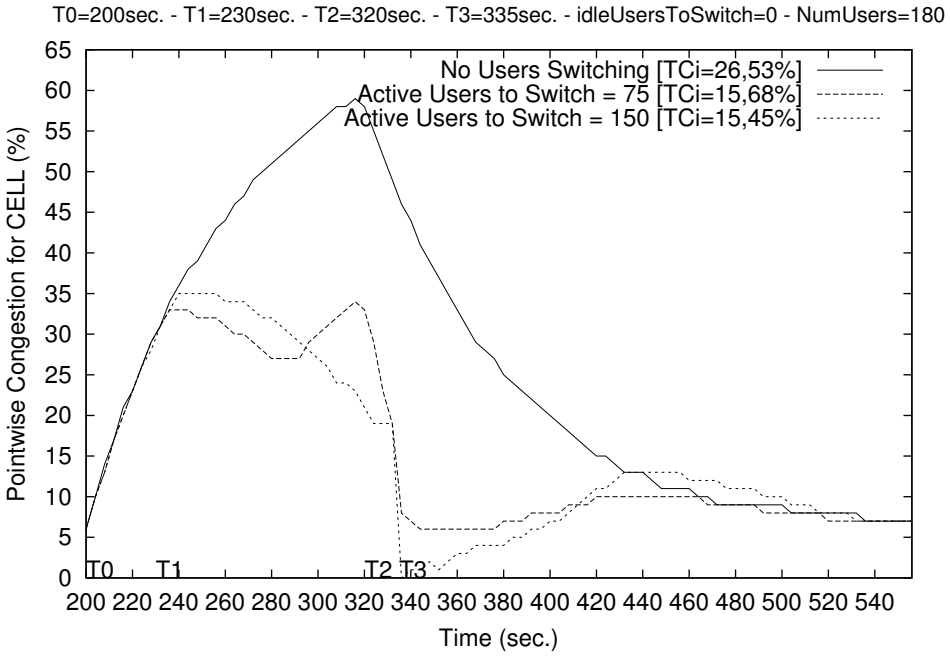

Figure 2.14: Congestion perceived in CELL

decreases, while increases from time T3 (because of the users re-switching), till reaching again the steady-state level.

Figure 2.15 shows the behavior of the overall GPRS network composed of CELL, CELL1, CELL2 and CELL3 at varying values of the *activeUsersToSwitch* parameter. We analyze the percentage of the unsatisfied users in the network with respect to the total number of users camped in the four cells (in this example 180+140+170+200=690 users). We note that the 100% cell resizing curve (*activeUsersToSwitch*=150) is worse than the 50% one (*activeUsersToSwitch*=75) as the positive effects induced by the decongestion in CELL don't compensate the negative effects on CELL1, CELL2 and CELL3 (the receiving cells).

Lastly, Figure 2.16 shows the number of active users really switched from CELL to the other cells. We note that the switching and re-switching procedures are not instantaneous. This means that there are not enough active users immediately available to be switched at time T1 (the switching time) and re-switched at time T3 (the re-switching time).

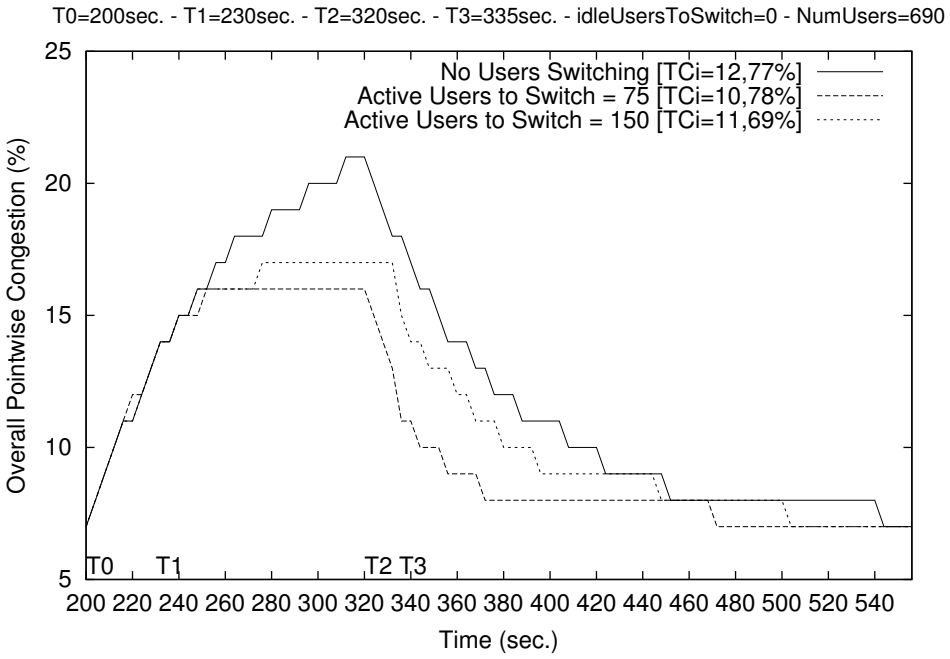

Figure 2.15: Overall congestion perceived

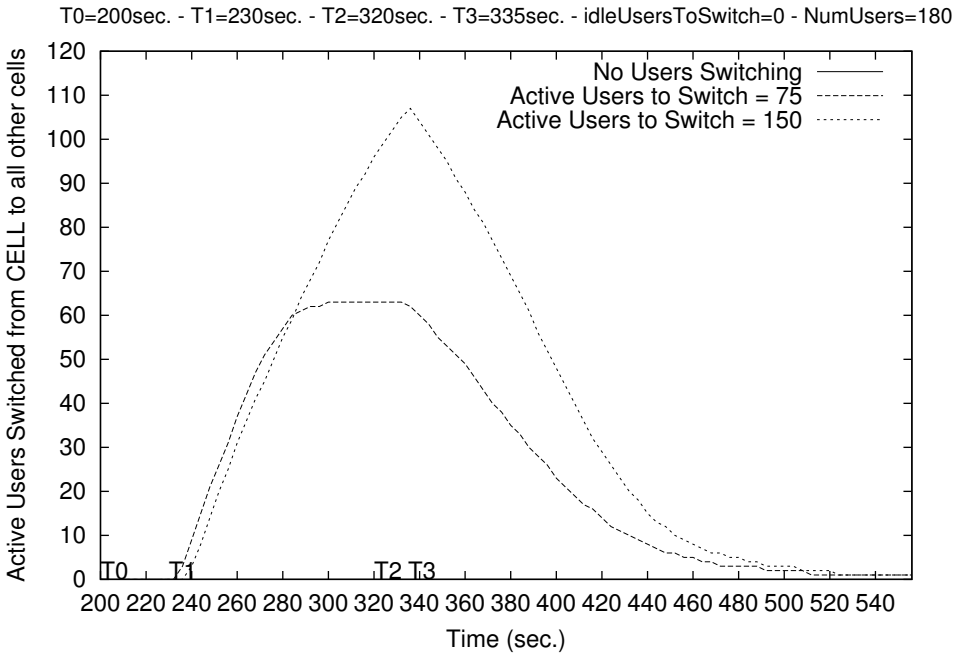

Figure 2.16: Active users switched from CELL to all other cells

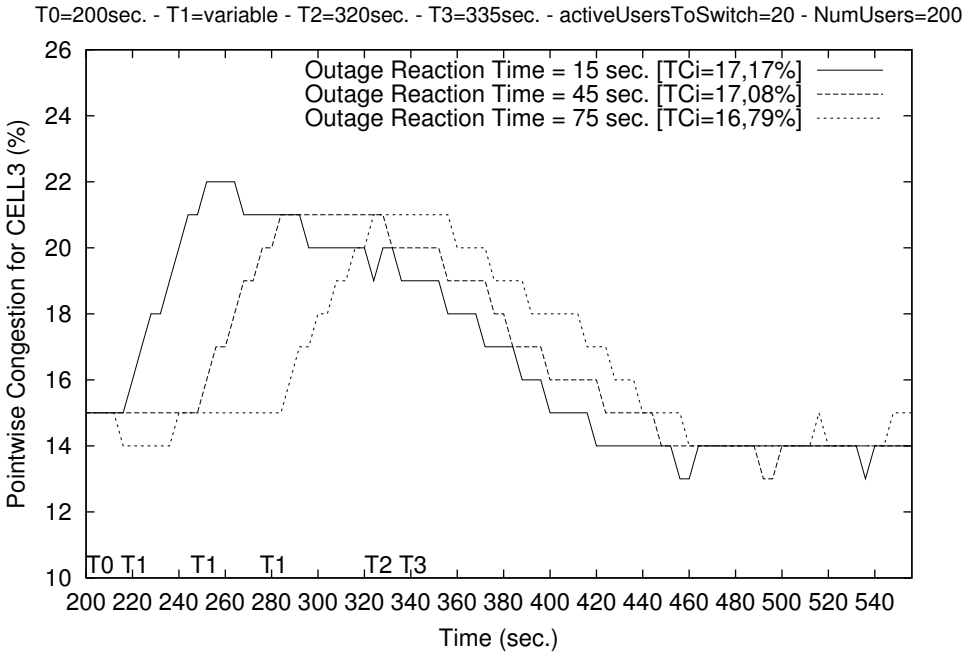

Figure 2.17: Congestion perceived in CELL3

Evaluation in scenario 2: tuning of parameter 'outageReactionTime'

Figure 2.17 shows the congestion perceived by CELL3 (one of the receiving cells) at varying the time needed by the system to react to the outage (*outageReaction-Time* parameter). As expected, the congestion increases if the outage reaction time decreases, both concerning PCf and TCi, as the switched users reach the cell earlier. The other receiving cells behave similarly (they only vary in the workload at steady-state level, about 1% for CELL1, 4% for CELL2) and then they are not presented.

Figure 2.18 shows the congestion perceived by the users camped in the central cell at varying of the *outageReactionTime* parameter. As expected, the TCi decreases when reducing the outage reaction time, as the reconfiguration action is applied earlier.

Finally, Figure 2.19 shows the percentage of unsatisfied users in the overall network at varying the *outageReactionTime* parameter. This is the percentage of unsatisfied users in the network with respect to the total number of users camped in it (690 users for the considered setting). We note that if the reaction time parameter increases, the congestion perceived increases as well. The obtained results

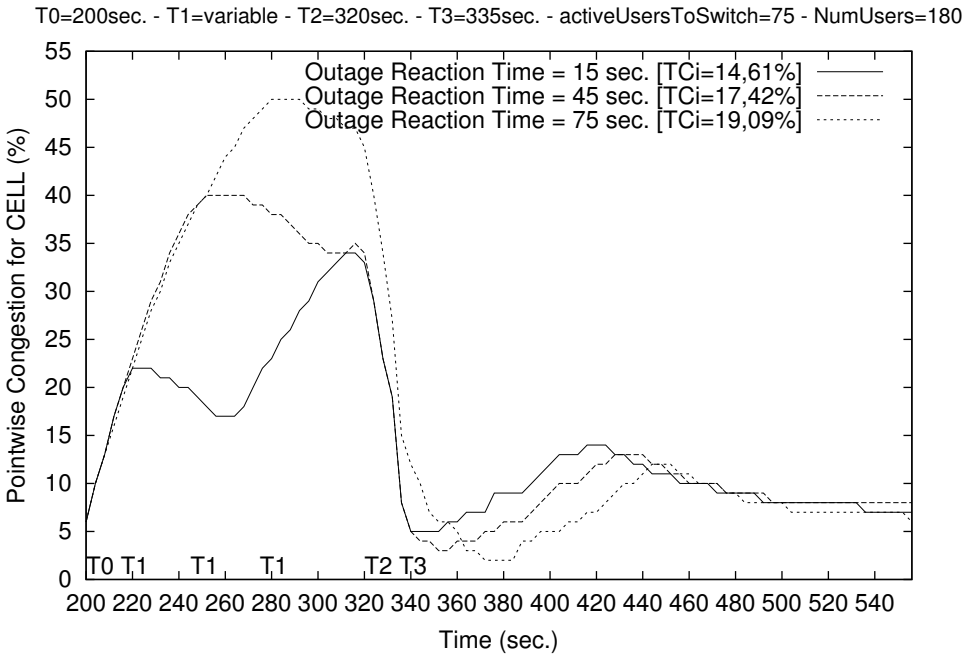

Figure 2.18: Congestion perceived in CELL

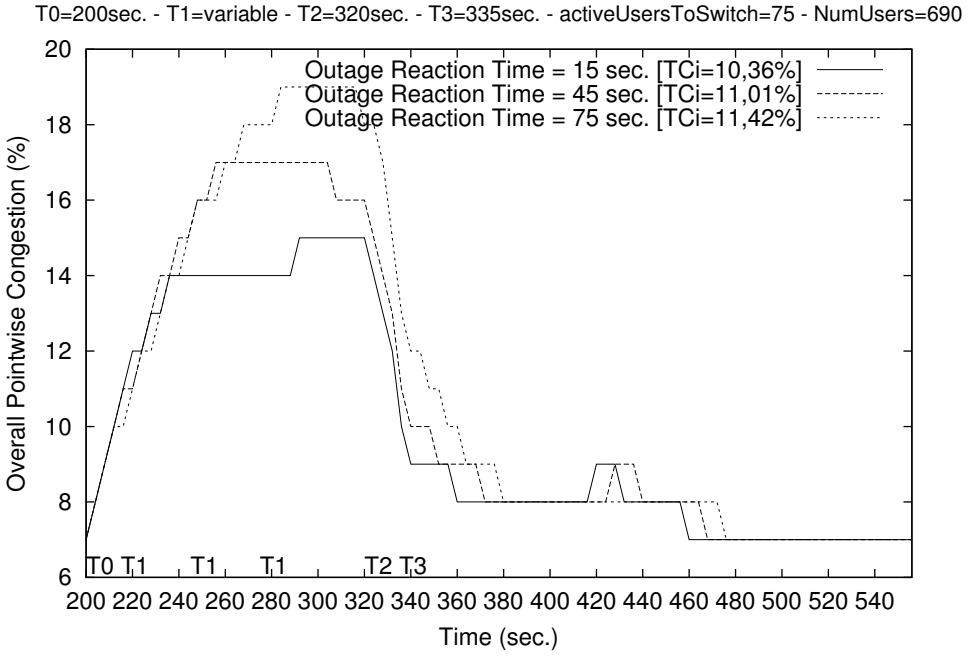

Figure 2.19: Overall congestion perceived

allow performing an interesting investigation on the amount of time that the system should be permitted to spend for its decision-making processes. For example, if a maximum tolerable level of degradation is known a priori, by looking at the results in Figure 2.19 it can be inferred the maximum value for the *outageReactionTime* parameter.

2.6 Summary

In this Chapter, the congestion analysis of GPRS infrastructures consisting of a number of cells partially overlapping has been performed in terms of QoS indicators expressing a measure of the service availability perceived by users. When a congestion is experienced by one of these cells, a family of congestion management techniques is put in place, to operate a redistribution of a number of users in the congested cell to the neighbor ones, in accordance with the overlapping areas. Since the service availability perceived by users is heavily impacted by the congestion experienced by the cells, determining appropriate values for the users to switch, so as to obtain an effective balance between congestion alleviation in the congested cell and congestion inducement in the receiving cells, is a critical aspect in such contexts.

In order to carry on such fine-tuning activity, a modeling methodology, appropriate to deal with the system complexity, has been defined. In particular, we introduced a solution technique following a decomposition approach, in which the solution of the entire model is constructed on the basis of the solutions of the individual sub-models.

Models solution through a simulation approach has been performed in order to provide numerical estimates. The obtained results, although dependent on the considered parameters setting, show behavior trends very useful to make an appropriate choice of the number of users to switch, which is a critical parameter for the congestion management technique. Moreover, an investigation on the amount of time that the system should be permitted to spend for its decision-making processes is carried on.

Chapter 3

Efficient Dependability Evaluation of Hierarchical Control Systems

Current and future computerized systems and infrastructures are going to be based on the layering of different systems, designed at different times, with different technologies and components and difficult to integrate. Control systems and resource management systems are increasingly employed in such large and heterogeneous environment as a parallel infrastructure to allow an efficient, dependable and scalable usage of the system components. System complexity comes out to be a paramount challenge to solve from a number of different viewpoints, including dependability modeling and evaluation. Key directions to deal with system complexity are abstraction, decomposition and hierarchical structuring of the system functionalities. In this Chapter we address the issue of an efficient dependability evaluation by a model-based approach of hierarchical control and resource management systems ([23]). The proposed methodology is then applied to the resource management system developed in the context of the European project CAUTION++.

3.1 Introduction

In this Chapter we focus on the class of control and resource management systems. To cope with their increasing complexity, such systems are typically developed in a hierarchical fashion: the functionalities of the whole system are partitioned among a number of subsystems working at different levels of a hierarchy. At each level, a subsystem has knowledge and control of the portion of system under its control (lower levels), while it acts just as an actuator with respect to the higher level subsystems. In this organization, the flow of the information goes vertically from one level to the other, but not horizontally inside the same level. More pre-

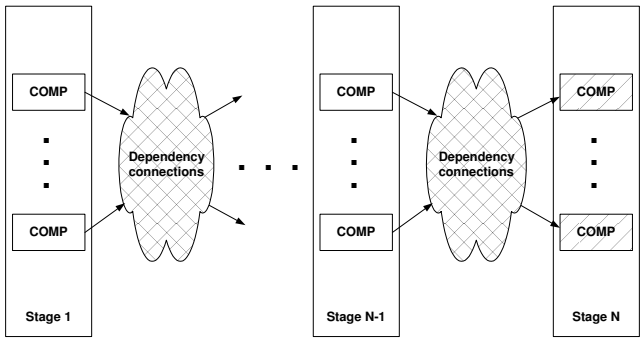

Figure 3.1: Class of systems with "multi-stage" representation

cisely, the flow of decision taking goes from the bottom to the top, while the flow for decision actuation goes from the top to the bottom. Here we are interested in modeling and evaluating the system behavior with reference to a unidirectional flow (be it for decision taking or for decision actuation). To improve dependability, fault tolerance measures may be taken at each level, typically introducing interface checks to cope with erroneous inputs and/or outputs and internal checks to cope with faults during the internal computation. We exploited the characteristics of this specific, but well representative, class of systems and derived a modeling methodology that is not only directed to build models in a compositional way, but it also includes some capabilities to reduce their solution complexity. To show how it works, in the second part of the Chapter we applied the methodology to a case study, which consists of a resource management system developed inside the CAUTION++ project [24].

The rest of this Chapter is organized as follows. Section 3.2 provides some preliminaries on the considered class of systems. Section 3.3 outlines the modeling approach. Section 3.4 presents the multi-stage system instance considered in the analysis. In Section 3.5 the models set-up for the selected CAUTION++ instance are discussed, while the results of the numerical evaluation are provided in Section 3.6. Finally, conclusions are in Section 3.7.

3.2 System context

The class of systems we focus on consists of a set of hardware or software components (the COMP boxes), which are grouped in "stages" (Stage 1, ..., Stage k, ..., Stage N), as shown in Figure 3.1.

Components at a certain stage may interact with others at an higher level through some "Dependency connections". Each connection identifies a flow of

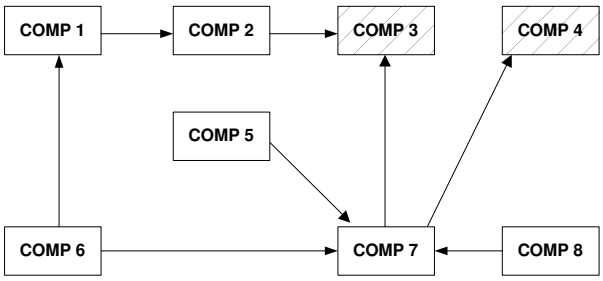

Figure 3.2: Example of system

information between two system components: a component A is connected to a component B ($A \longrightarrow B$) if B is dependent from A, that is the behavior of B depends on the information received from A. The components without any incoming connections have an independent behavior with respect to the others, while the components without any outgoing connections (called *root* components, the dashed boxes in the figure) do not affect the behavior of any other component.

From the general system depicted in Figure 3.1 and following the dependency connections from a root component back to the leaves of the graph, a number of individual subsystems structured in a hierarchical fashion may be derived, equal to the number of root components.

As already discussed earlier, a component at stage k may interact only with those at stages $k-1$ and $k+1$ and these dependencies are unidirectional, from the lower stage to the higher one. A dependency between one component at stage k and more than one component at stage $k+1$ is not explicitly considered as it is equivalent to consider some (logical) replications of the component at stage k, each one interacting with only one component at stage $k+1$. In doing this we make the assumption that, if a component is used in computing two or more outcomes, its behavior is independently modeled in each context. This means that the behavior of each replica does not depend on the behavior of the others.

The components in a stage can be partitioned in more sub-sets (*groups*), each one composed of components having a connection to the same component in the next stage.

For a better understanding, let us consider the example of Figure 3.2.

It is a system with eight different components, two of which are root nodes. The corresponding representation, by grouping components in stages, is shown in Figure 3.3. The original system has been decomposed in two sub-systems of four and three stages, respectively, obtained following the reversal path from each root node to the leaves. We note that COMP 6 is replicated twice in the first sub-system, as it is originally connected to two different components (COMP 1 and COMP 7, see Figure 3.2). We identify the groups composed of more that one component

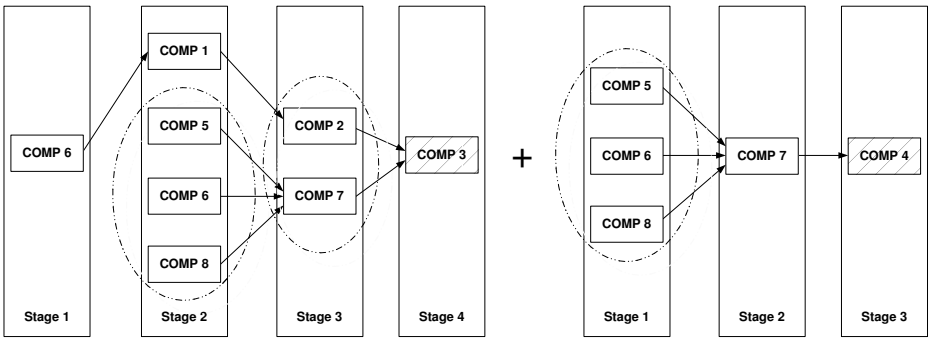

Figure 3.3: Example of "multi-stage" representation

with a dotted circle.

In the following Subsection we detail the system's behavior, specifying how two generic components may interact each other.

3.2.1 Interactions between components and Measures of interest

The interactions among components and the failure assumptions on each component are highlighted in Figure 3.4. This scheme is very general and must be specialized for the particular component under analysis. To explain the generic component's behavior, let's suppose it receives an input following a Poisson distribution with a rate λ^{IN}. These inputs are assumed to be correct or incorrect with a probability α and $1 - \alpha$, respectively. In correspondence of inputs, which arrive with a rate λ^{IN}, the component produces an output with a rate $p * \lambda^{IN}$, where p is the probability a received input leads the component to produce an output. Moreover, the component is assumed to possibly behave incorrectly by self-generating spurious outputs with a rate λ^{S}. Thus, the "potential"[1] output rate of the component is expressed as $\lambda^{IN \rightarrow OUT} = \lambda^{IN} + \lambda^{S}$.

For the sake of clarity, we give some definitions that we will use in the rest of the Chapter. A *correct emission* is the emission of a correct output, that occurs whenever a correct output is produced. It is possible i) in response to a correct input if the system is free from errors, or ii) in response to a correct input, if system errors are detected and tolerated. A *correct silence* is the non-emission of an incorrect output and it may happen as consequence of an incorrect input (if the incorrectness of the input is detected, for example using interface checks) or of an erroneous status of the system. An *incorrect emission* is the emission

[1]Here, a "potential" output encompasses both emitted and non-emitted output ($p = 1$), while for "output" we refer only to those emitted.

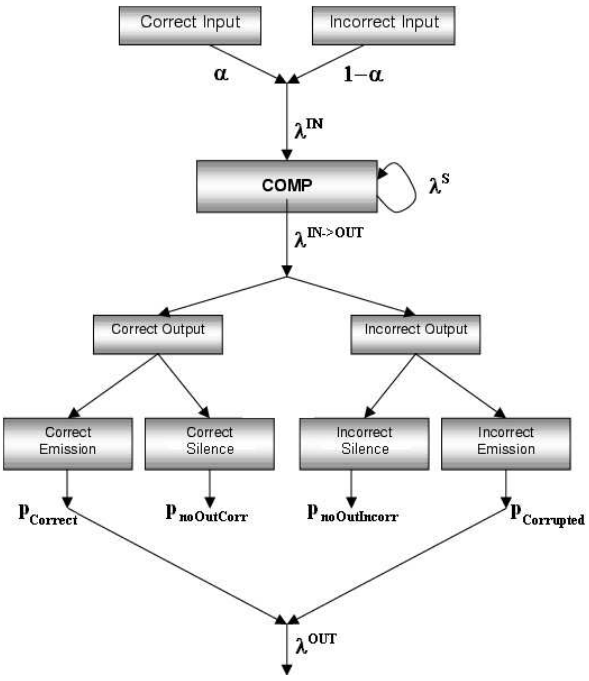

Figure 3.4: How a generic component interacts with others

Input	**Corresponding feasible output**
Spurious output (internally generated)	Incorrect Emission
Correct input	Correct Emission, Incorrect Emission, Incorrect Silence
Incorrect input	Correct Silence, Incorrect Emission

Table 3.1: Input-output combinations

of an incorrect output and it happens either in reply to an incorrect input, or as consequence of a spurious output or of a wrong processing of a correct input. Finally, an *incorrect silence* is the non-emission of a correct output and it may happen as consequence of wrong processing of a correct input. These input-output combinations are summarized in Table 3.1.

Therefore, each component can be characterized by two input parameters (α and λ^{IN}) and by the following five output parameters:

- $p_{Correct}$, that is the probability of generating a correct output (correct emission);
- $p_{Corrupted}$, that is the probability of generating an incorrect output (incorrect emission);

- $p_{noOutCorr}$, that is the probability that the output is correctly non-emitted (correct silence);
- $p_{noOutIncorr}$, that is the probability that the output is incorrectly non-emitted (incorrect silence);
- λ^{OUT}, that is the rate of the propagation of an output from the component to another. In particular, $\lambda^{OUT} = (p_{Correct} + p_{Corrupted}) * \lambda^{IN \rightarrow OUT}$.

From the point of view of propagation, it is clear that not all the outputs generated at a stage are always propagated up to the root. In fact when a component receives an output (correct or incorrect), it can operate in two different ways, depending on the correctness of the output received and on its internal state: it can generate another output and propagate it to the next stage (*emission behavior*), or it can not emit any output, thus interrupting the propagation flow (*silence behavior*).

Given the behavior structure and failure semantics depicted in Figure 3.4, typical measures of interest from the dependability point of view in this context include:

1. The probability of correct and incorrect emission;
2. The probability of correct and incorrect silence;
3. The overall probability that the system does not undertake wrong actions;
4. The mean time to incorrect emission.

In Section 3.5 we will specify the measures to evaluate with reference to a particular resource management system.

3.3 Description of the modeling methodology

The modeling methodology, originally introduced in [21], is fully described in this Section. First, we deal with the model design process, that is, how to model a complex system starting from its functional specification and applying a stepwise refinement to decompose it in small sub–models. Then, the second part of the methodology is presented, which concerns the modular model solution, carried out in a bottom-up fashion. The philosophy of our modeling approach is shown in Figure 3.5.

In order to construct an efficient, scalable and easily maintainable architectural model, we introduce a stepwise modeling refinement approach, both for the model design process and for the model solution. Another advantage of this approach is

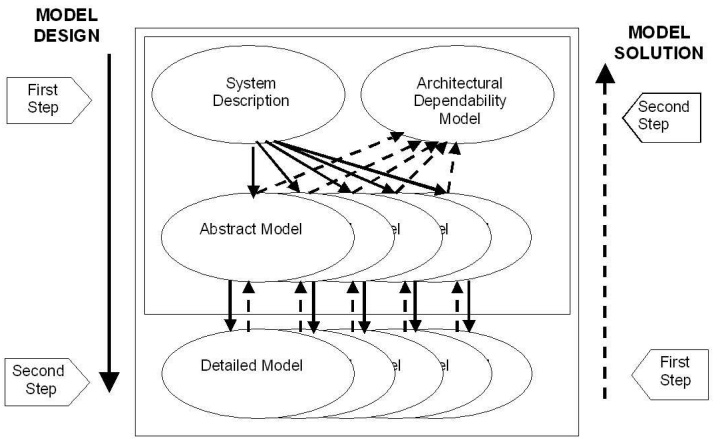

Figure 3.5: Modeling approach

to allow models refinement as soon as system implementation details are known or/and need to be added or investigated.

3.3.1 The model design process

The model design process adopts a top-down approach, moving from the entire system description to the definition of the detailed sub-models, while the model solution process follows a bottom-up approach. As inspired by [8], the system is firstly analyzed from a functional point of view (functional analysis), in order to identify its critical system functions with respect to the validation objectives. Each of these functions corresponds to a critical service provided by a component.

The overall system is then decomposed in subcomponents, each one performing a critical subfunction, and each subfunction is implemented using a model that describes its behavior. Therefore, starting from the high-level abstract model, we perform a decomposition in more elementary (but more detailed) sub–models, until the required level of detail is obtained.

The definition of the functional (*abstract*) model represents the first step of our modeling approach. The rules and the interfaces for merging them in the architectural dependability model are also identified in this phase. The second step consists in detailing each service in terms of its software and hardware components in a detailed (*structural*) model accounting for their behavior (with respect to the occurrence of faults). The fundamental property of a functional model is to take into account all the relationships among services: a service can depend directly from the state of another service or, indirectly, on the output generated from another service. The detailed model defines the structural dependencies (when existing) among the internal sub–components: the state of a sub–component can

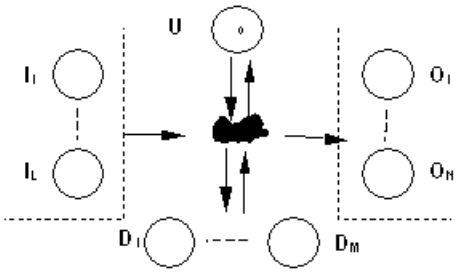

Figure 3.6: Functional-level model related to a single service

depend from the state (failed or healthy) of another sub–component.

Figure 3.6 shows the functional-level model related to a single service. The internal state S is here composed of the place U, representing the nominal state, and of the places $D_1 \ldots D_M$, representing different possible erroneous (degraded) states. The places $I_1 \ldots I_L$ and $O_1 \ldots O_N$ represent, respectively, the input (correct or exceptional, due to propagation of failures from interacting modules) and the output of the model (correct behavior or failure - distinguishing several failure modes). The state changes (from the nominal, correct state to the erroneous states and viceversa) and the flow between the input and output places are regulated by a structural model of the service implementation, indicated in Figure 3.6 as a black cloud.

3.3.2 The model solution process

The model solution follows a bottom-up approach from the detailed model up to the abstract model. The implementation is strictly related to the environment characteristics of the system under analysis. Actually, starting from the general class of systems of Figure 3.1, we can derive several simplified systems that can be solved very efficiently.

Environment characteristics

Suppose, for the sake of simplicity, that the generic system of Figure 3.1 has one root node only. If it is not the case, we can decompose the system in more sub-systems having one root each, as explained in Section 3.2. We denote with $\lambda_i^{OUT,\,COMP^k}$ the intensity of the output process of the i-th component belonging to stage k (COMP$_i^k$). We make the following assumptions:

1. The distribution of the input process of each component is Poisson with rate λ^{IN}. This is accepted in the literature when the number of arrivals in a given time interval are independent of past arrivals.

2. The distribution of the output process of each component is Poisson distributed with a rate λ^{OUT}. This assumption corresponds, for example, to the case in which the inputs are processed sequentially without queuing and losses, and the processing time of the input is deterministic. Equivalently, we could obtain the same output distribution considering that the service time is Poisson distributed and that the component operates as a steady-state M/M/1 queuing network [1].

Suppose to have a group of N_k components at stage k (COMP$^k_1$, ..., COMP$^k_{N_k}$). We remind that a group is a set of components belonging to a stage, and connected to the same component in the next stage. Using the assumption that the output process of COMP^k_i is Poisson distributed with rate $\lambda_i^{OUT,\ COMP^k}$, the superposition of N_k Poisson processes with intensities $\lambda_1^{OUT,\ COMP^k}, \ldots, \lambda_{N_k}^{OUT,\ COMP^k}$ is equivalent to a Poisson process with intensity equal to $\lambda_1^{OUT,\ COMP^k} + \ldots + \lambda_{N_k}^{OUT,\ COMP^k}$.

Solving the detailed model of components COMP$^k_1$, ..., COMP$^k_{N_k}$ leads to the evaluation of the probabilities of correct/incorrect output (both propagated and not propagated to the next stage) and the intensity of the output process of a group of N_k components. Let's defining as $P^{k_i}_{Correct}$, and $P^{k_i}_{Corrupted}$ the probability of correct emission, and the probability of incorrect emission of COMP^k_i, respectively. Notice that these probabilities depend upon the intensity of the input process ($\lambda_i^{IN,\ COMP^k}$) and of spurious alarms ($\lambda_i^{S,\ COMP^k}$) (both supposed being Poisson). The following relations holds:

$$\Lambda^{OUT,\ COMP^k} = \sum_{i=1}^{N_k} \lambda_i^{OUT,\ COMP^k} \quad , \tag{3.1}$$

$$\alpha_{COMP^{k+1}} = \frac{1}{\Lambda^{OUT,\ COMP^k}} * \sum_{i=1}^{N_k} \lambda_i^{OUT,\ COMP^k} \frac{P^{k_i}_{Correct}}{(P^{k_i}_{Correct} + P^{k_i}_{Corrupted})} \quad , \tag{3.2}$$

where $\Lambda^{OUT,\ COMP^k}$ is the intensity of the process achieved by aggregating the output processes of the components COMP$^k_1$, ..., COMP$^k_{N_k}$, while $\alpha_{COMP^{k+1}}$ is the probability that the next component at stage $k+1$ receives a correct input. Analogous considerations hold for COMP^{k+1}, and so on. This general approach can be specified for the following cases:

(I) If all groups at stage k are identical, the total number of detailed models to be solved in order to evaluate the system's behavior is equal to $\sum_{k=0}^{K} N_k$, where K is the number of stages in the system and N_k is the number of components belonging to each identical group at stage k.

(II) If all groups at stage k can not be considered identical at each stage, the number of models to be solved depends on the number of different "branches" in which the overall model can be simplified.

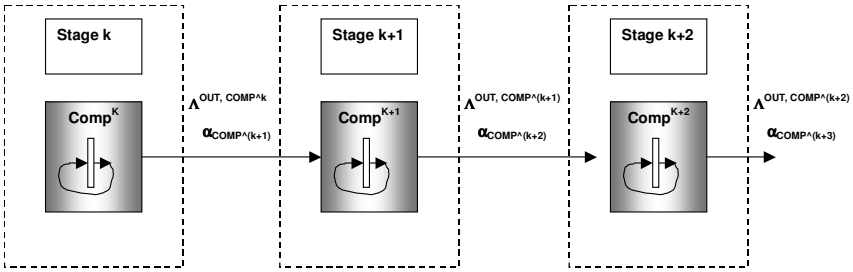

Figure 3.7: Part of the simplified system model

(III) If for each stage k of the system, all the components are identical, it is possible to solve only K detailed models, one for each stage. Therefore, if all the components at level k are identical, then $\lambda_i^{OUT,\ COMP^k} = \lambda^{OUT,\ COMP^k}$, $P^{k_i}_{Correct} = P^k_{Correct}$, $P^{k_i}_{Corrupted} = P^k_{Corrupted}$, and the previous equations reduce to

$$\Lambda^{OUT,\ COMP^k} = N_k^{TOT} * \lambda^{OUT,\ COMP^k} \quad , \tag{3.3}$$

$$\alpha_{COMP^{k+1}} = \frac{P^k_{Correct}}{(P^k_{Correct} + P^k_{Corrupted})} \quad , \tag{3.4}$$

where N_k^{TOT} is the total number of components at stage k.

In this case, the general model of Figure 3.1 is reduced to the equivalent simplified system model of Figure 3.7 that can be solved more easily, as the "tree" structure collapses in a unique "branch" from the point of view of system evaluation.

We note that case (II) is the more general one; next is case (I) and the least general is case (III).

If it can not be assumed that the output process of COMP_i^k follows a Poisson distribution, the general approach is still valid provided that the detailed model is slightly modified allowing to estimate the real distribution of such a process. The same distribution will be used as input at the $k+1$ stage. However, in general, it will be no longer possible to solve the models analytically.

If the measures of interest are probabilities, the moments of the distribution of correct/incorrect output (both propagated and not propagated to the next stage) which yield such probabilities are not considered at all. In this case it is not necessary to use, at the abstract level, models having the same distribution estimated at the detailed ones. If, on the contrary, we are interested in evaluating the moments, the output processes distributions achieved by the detailed models have to be used for the solution of the abstract models.

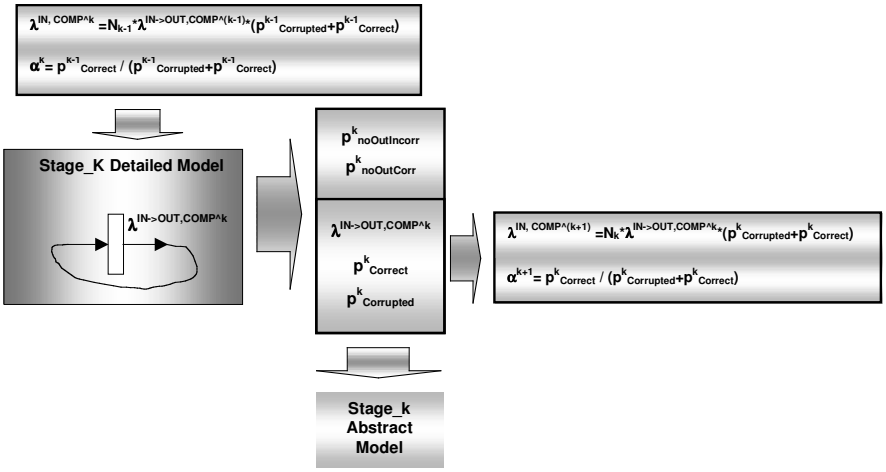

Figure 3.8: Relationships between models solutions

The model solution scheme

According to Figure 3.5 (showing the philosophy of our modeling approach) the model solution follows a bottom-up approach: the solution of a detailed model is exploited to set up the parameters of the corresponding abstract model and of the detailed model of the next (contiguous) components (the output of the detailed COMP^k model acts as input for the detailed COMP^{k+1} model). To keep the presentation simple, the model solution scheme is described in the case where, for each stage k, all the components at stage k are identical; therefore only K detailed models (one for each stage) have to be solved. Figure 3.8 shows the relationships among a detailed model of COMP^k and the model COMP^{k+1}.

With reference to the measures of interest listed in Section 3.2.1, the outcomes of the detailed model COMP^k are:

1. $p^k_{noOutCorr}$: is the probability that no output is produced by component COMP^k, as a consequence of an incorrect input;

2. $p^k_{noOutIncorr}$: is the probability that an expected output is incorrectly not propagated by component COMP^k, as consequence of an internal fault;

3. $p^k_{Correct}$: is the correct emission probability;

4. $p^k_{Corrupted}$: is the emission failure probability. This value encompasses both an expected wrong emission (as consequence of wrong internal processing) and the unexpected emission (as consequence of an internal self-generated false alarm);

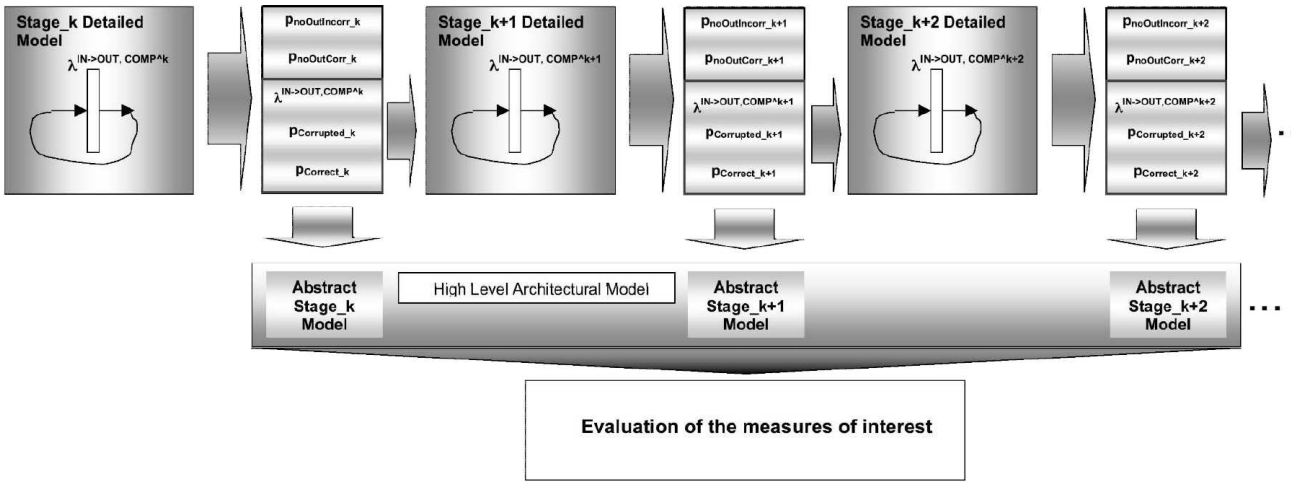

Figure 3.9: Overall solution scheme

5. $\lambda^{IN \rightarrow OUT,\ COMP^k} * (p^k_{Corrupted} + p^k_{Correct})$: is the rate of the output (both correct and corrupted) propagated by component COMP^k to component COMP^{k+1}.

All these parameters are used in the abstract model of component COMP^k (see Figure 3.8) while $\lambda^{IN \rightarrow OUT,\ COMP^k}$, $p^k_{Correct}$ and $p^k_{Corrupted}$ are used to derive the parameter $\lambda^{IN,\ COMP^{k+1}}$ to be used in the detailed model of COMP^{k+1}. In the system framework COMP^k and COMP^{k+1} represent two components directly connected that exchange information in one direction (from COMP^k to COMP^{k+1}).

Summarizing, the overall solution scheme is shown in Figure 3.9. The detailed models are solved separately: firstly the model of COMP^k is solved, then the values provided by equations (3.3) and (3.4) are passed as input to the detailed model of COMP^{k+1} and so on. Finally, the probabilities of correct/incorrect output (both propagated and not propagated to the next stage) are passed to the corresponding abstract models, they are joined together and then the overall abstract model is solved.

The advantages of the proposed approach are in two directions: first, to cope with the problem of state space explosion when modeling a system composed of a large number of independent components and, second, to allow efficient model solution for those systems having most of their components identical and interacting with each others only by means of information exchange. Actually, in case the components are not all equal, a larger number of detailed models have to be solved but still separately. Thus, the overall model, encompassing all the useful information with respect to the measures of interest, is achieved by joining the abstract models.

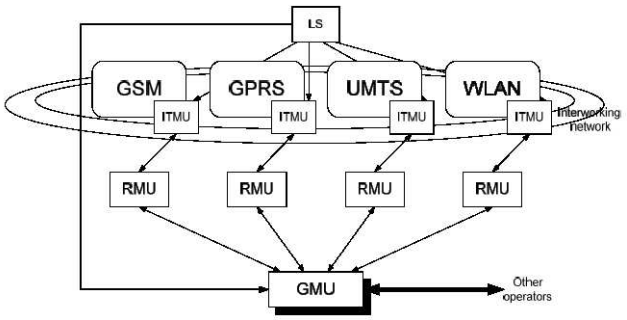

Figure 3.10: Network architecture for provision of capacity management mechanisms

3.4 An instance of a "multi-stage" system: the CAUTION++ platform

The IST-2001-38229 CAUTION++ project [24] has already been introduced in Section 2.1, and it is here detailed focusing the attention on its architecture. Its final goal was to develop a novel, low cost, flexible, highly efficient and scalable system able to be utilized by mobile telephone operators to increase the performance of all network segments. Capacity utilization in cellular networks is an extremely important issue from the operators' point of view. Successful usage of all the system resources especially in congestion situations can imply increased revenues for the cellular network operators via reduced call blocking and dropping rates. Also, in emergency situations the cellular networks are expected to work properly and be able to respond to the momentarily increased offered traffic. To pursue such goals, proper system components are developed to handle generated alarms through a set of RRM (Radio Resource Management) techniques, to be applied where needed. The CAUTION++ system, superimposed over the existing wireless networks, should allow putting in place correctly the identified RRM techniques, hopefully despite the occurrence of faults. The rationale is to enforce design solutions able to prevent a CAUTION++ component from carrying out a reconfiguration action wrongly or when it is not necessary (as consequence of some fault). Because of the involved functionalities which pose relevant dependability issues, the CAUTION++ project has promoted model-based evaluation, aiming at assessing dependability attributes of the architecture under development.

Figure 3.10 shows the main components of the CAUTION++ architecture. Each network segment has its own ITMU (Interface Traffic Monitoring Unit) and RMU (Resource Management unit) which allow to monitor and manage the attached network, respectively. Within each operator network, a GMU (Global Management unit) can perform a global optimization. A Location Server (LS)

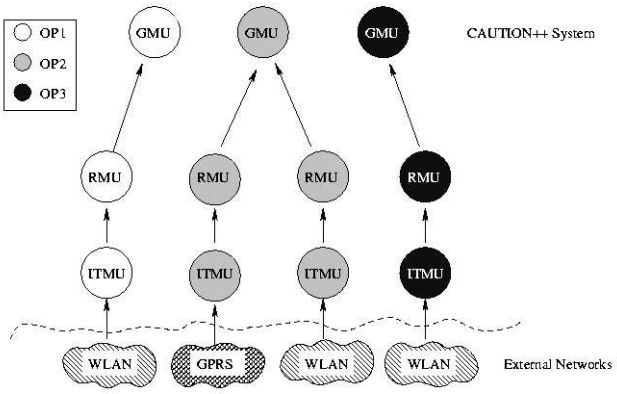

Figure 3.11: Trial configuration

can be used to track users' mobility and location: such information can be exploited by GMU for a global optimization.

To practically show the usage of the proposed modeling methodology, we consider a specific architecture's instance involving GSM/GPRS and WLAN network technologies deployed by two distinct operators, which is actually one of the trial systems set up by the consortium as a demonstrator of the project's results.

From the point of view of system composition, Figure 3.11 depicts the components included in such trial. Three operators are involved, Op1, Op2 and Op3, with Op1 and Op3 managing a WLAN network only, and Op2 managing both a GPRS and a WLAN network. From the point of view of CAUTION++ components employed in this instance, each network segment has its own ITMU (Interface Traffic Monitoring Unit) and RMU (Resource Management Unit) which allow to monitor and manage the attached network, respectively. Within each operator network, a GMU (Global Management Unit) is necessary to perform a global optimization. In fact, different GMUs cooperate to optimize among different operators. Therefore, this CAUTION++ instance includes 4 ITMU, 4 RMU and 3 GMU, connected as shown in Figure 3.11.

It is clearly an instance of a multi-stage system. Starting from the GMU components (the root nodes of the graph, see Section 3.2), we decompose the system in three subsystems, one for each GMU. Each subsystem can be seen as a "3-stage" system, that is a "multi-stage" system composed of 3 stages, in which all the components belonging to a stage are identical. In this context the interactions between system components consist of messages flowing from a level to its upper level. Moreover, each subsystem can be represented as shown in Figure 3.7, as the

"tree" structure collapses in a unique "branch" from the point of view of system evaluation. Therefore we have to solve only 3 detailed models for subsystem.

3.4.1 Components behavior and Modeling assumptions

In order to set up the detailed models, a characterization of the system components from the dependability point of view is necessary, briefly outlined in the following.

- Each CAUTION++ element (ITMU, RMU, GMU) can be either correctly working or wrongly working.
- Each CAUTION++ element (ITMU, RMU, GMU) is composed by three main elements: the Application Software (AS), the Operating System (OS), and the Hardware (HW). Each element has its own dependability figures and reference values, that have been chosen as explained later. In turn, the AS, OS, and HW can be either correctly working or wrongly working.
- At the end of its computation, each CAUTION++ component can emit an output or not. More precisely, the possible output can be either correct/incorrect emission or correct/incorrect silence.
- Fault tolerance mechanisms are in place in each system component, in order to improve the dependability of the components themselves and limit the error propagation between interacting elements. They are interface checks (to detect errors at input/output level), diagnosis and repair mechanisms. Their ability to work properly depends on their respective coverage.

3.5 The models derived for the selected CAUTION++ trial

In this Section, the models derived for the analysis of the selected CAUTION++ instance of Figure 3.11 are detailed, all developed using the SAN [41] formalism. First, the measures of interest are described, since they influence the definition of the system models.

3.5.1 Measures of interest

As previously mentioned, the goal of the CAUTION++ system is to increase the performance of all the controlled cellular networks. Then we expect it should

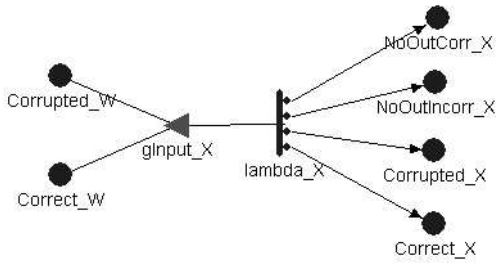

Figure 3.12: Generic abstract sub-models

never have a negative impact on the networks behavior, at the most becoming inactive in the worst case. Therefore, the main dependability requirement of CAUTION++ is that it should avoid taking wrong decisions, thus acting worse than doing nothing.

Particularly, an incorrect silence behavior (that is the system does not provide any output when, if correct, it would have emitted one) can be tolerated, since it leads to no benefit from CAUTION++. On the contrary, an incorrect emission of an output can lead the system to act worse than doing nothing, and therefore actions would be required to prevent such failure mode.

We have identified the following indicators as significant measures to evaluate the dependability of the CAUTION++ architecture. They are:

- The probability of incorrect emission at level of the GMU employed by a certain operator;
- Mean Time to Failure of the GMU employed by a certain operator;
- Reliability of the whole system(with contributions from all the present GMUs).

They appear to be suitable measures to evaluate the ability of CAUTION++ in fulfilling the general dependability requirement of not undertaking wrong reconfiguration actions.

3.5.2 The abstract models

In accordance with the proposed methodology described in Section 3.3, the starting point is the definition of a functional model for each involved component. Each functional ("abstract") model has to take into account all the relationships among critical services that, in this trial, are the emissions of outputs from ITMU to RMU and from RMU to GMU. The generic "abstract" model is represented in Figure 3.12.

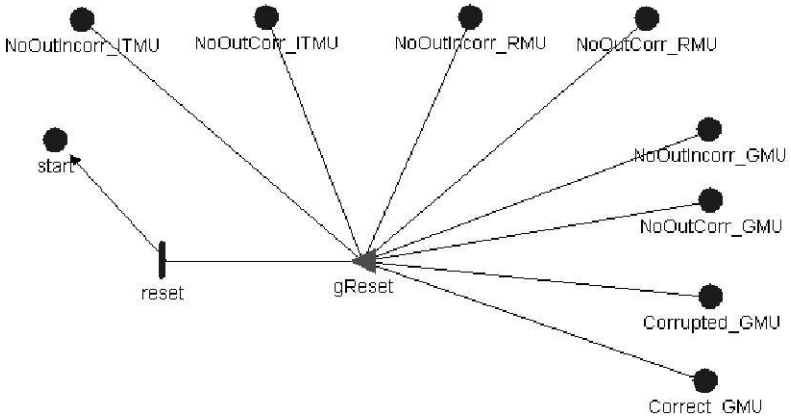

Figure 3.13: HighLevelJoin submodels

It is valid for ITMU, RMU and GMU. The extension X will be used whenever a generic parameter is indicated and need to be properly substituted by ITMU, RMU, GMU to designate the parameters of the corresponding component.

The input gate `gInput_X` allows handling the input of the component (both the correct input - place `Correct_W` - and corrupted input - place `Corrupted_W`). Transition `lambda_X` fires with a rate $\lambda^{IN \rightarrow OUT, X} = \lambda^{IN, X} + \lambda^{S, X}$ where $\lambda^{IN, X}$ and $\lambda^{S, X}$ are the rate of messages in input to component X and the rate of spurious messages generated by X, respectively.

Then, an output is produced. This output can be either correctly emitted (a token is moved in place `Correct_X` with probability *pCorrect_X)* or incorrectly emitted (a token is moved in place `Corrupted_X` with probability *pCorrupted_X*) or correctly non-emitted (a token is moved in place `NoOutCorr_X` with a probability *pnoOutCorr_X*) or incorrectly non-emitted (a token is moved in place `NoOutIncorr_X` with probability *pnoOutIncorr_X*). An output is propagated at the upper level of the CAUTION++ hierarchy (or as final output in case of GMU) with a rate $\lambda^{OUT, X} = (pCorrect\_X + pCorrupted\_X) * \lambda^{IN \rightarrow OUT, X}$.

The model of Figure 3.13 allows determining the measures of interest: a token will surely fall in one of the places (excepted place `start` which indicates an alarming condition from a network segment, e.g. an input for an ITMU). Whenever the token falls in one of these places it is collected by the input gate `gReset` then the transition `reset` gets enabled and puts a token once again in place start.

3.5.3 The detailed models

To obtain the parameters of each abstract model, the corresponding detailed models have to be set-up and solved. Therefore, a detailed model is built for each

involved component. Since ITMU, RMU and GMU employ the same subcomponents (HW, OS and AS, plus fault tolerance mechanisms, as already discussed), the detailed model is almost the same for all of them. The only difference is in the values of their parameters (as explained later in the Section on numerical evaluation). A generic detailed model is obtained by composing the generic detailed models for the component's subcomponents (i.e., HW, OS and AS) together with the dynamics of the error and fault detection mechanisms employed.

As typical when modeling a (complex) system behavior, a set of assumptions has been identified with the aim of enhancing simplicity and clarity (essential to keep the whole modeling activity under control), still capturing the relevant phenomena which impact the measures under analysis (essential to the practical usefulness of the evaluation effort). The assumptions we made are listed in the following. First the general assumptions concerning the CAUTION++ components are introduced, then those concerning the error detection capabilities at interface level (both IN and OUT), and, finally, those concerning the error detection capabilities internal to each component.

General assumptions

- Each CAUTION++ element X(ITMU, RMU, GMU) is in state `Up_X` if all its subcomponents Y (AS, OS and HW) are in state `Y_Up_X` (the prefix `Y` will be used throughout this Chapter whenever a generic parameter is indicated and need to be properly substituted by AS, OS, HW to designate the parameters of the corresponding component). It is in state `DownDet_X` if at least one sub-component is in state `Y_DownDet_X` and no one in state `Y_DownNoDet_X`. Otherwise it is in state `DownNoDet_X`.

- The input of the detailed model may be either correct with probability α or incorrect with probability 1-α.

- Each CAUTION++ element (ITMU, RMU, GMU) can generate by itself spurious outputs. We assume that these outputs are independent from outputs generated because of real inputs. Spurious outputs follow an exponential distribution with rate $\lambda^{s,x}$.

- An incorrect input does not affect the state of the AS, OS, and HW.

Assumptions on interface checks capabilities

- The coverage of the input interface checks is given by the probability *input-Coverage_X*.

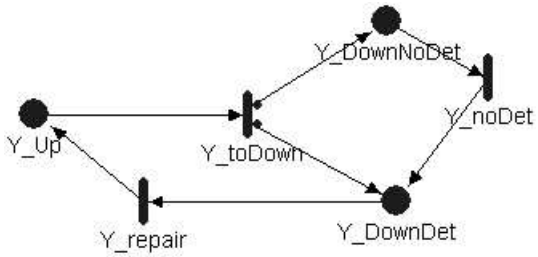

Figure 3.14: Detailed model for AS, OS, and HW

- When output interface checks are considered, the detection of an erroneous output leads to a non-emission of an output (silence behavior), which may be correct (with probability *outputCoverage_X*) or incorrect, depending on the inputs originating it and/or on the correctness of the component's status.

Assumptions on error and failure detection capabilities

- An undetected erroneous state either disappears or propagates and reveals itself.
- An undetected erroneous state of the AS may disappear (with probability *OS_errorDet_X*) when the OS is repaired, e.g. in the case of OS re-booting.
- An undetected erroneous state either at the AS or OS level may disappear (with probability *HW_errorDet_X*) when the HW is repaired (because of necessary system reboot, no hot-pluggable redundancy is envisioned).

We now explain how the overall detailed model is built step by step. Remind that each generic subsystem of CAUTION++ (i.e. ITMU, RMU, GMU) is supposed to be composed by three main components: AS, OS and the HW. The details of each component are shown in Figure 3.14. The extension $\_X$ is here omitted for brevity.

A token in place `Y_Up` indicates that Y is working correctly. The firing of activity `Y_toDown` models its failure: this failure can be detected (a token moves in the place `Y_DownDet`) or not (a token moves in the place `Y_DownNoDet`) with probabilities *Y_Coverage* and 1-*Y_Coverage*, respectively (*Y_Coverage* represents the coverage of the error detection mechanisms implemented in the element Y). An undetected failure can be revealed after a while; the firing of activity `Y_noDet` indicates such failure detection. A detected failure is then recovered by means of the activity `Y_repair`.

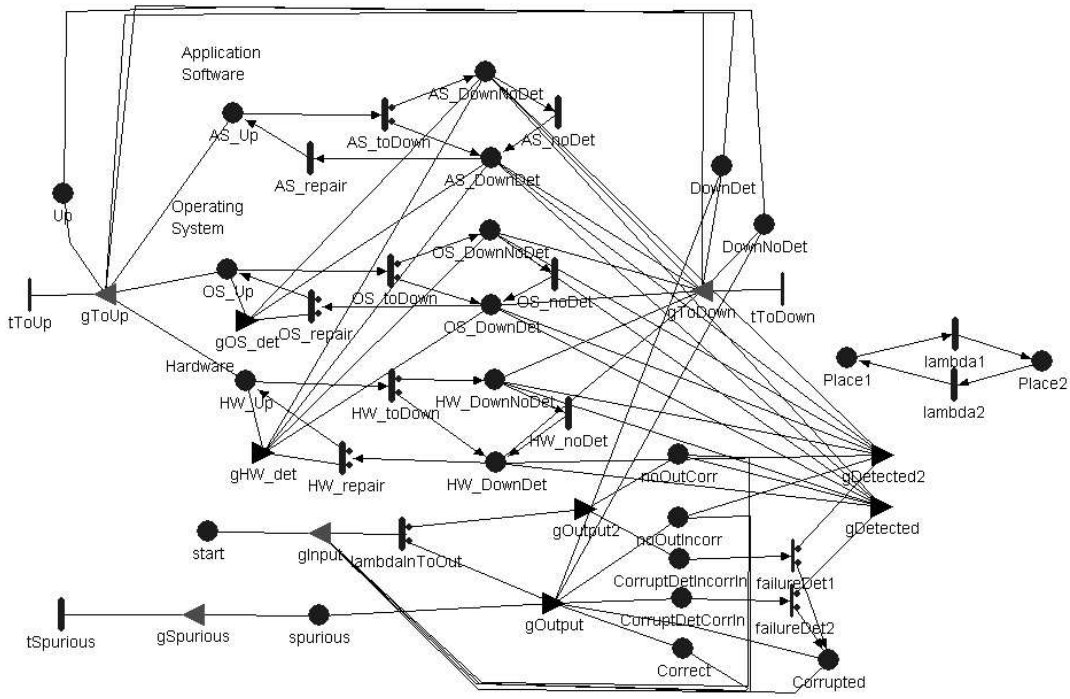

Figure 3.15: Generic detailed model

Figure 3.15 shows the detailed model for a generic CAUTION++ component (ITMU, RMU, GMU), which encompasses the detailed model of its AS, OS, and HW (the extension $\_X$ is still omitted for brevity). The occurrence of an error at AS, OS, and HW level is accounted for by putting a token in the corresponding places `Y_DownDet` or `Y_DownNoDet` in case of a detected or undetected error, respectively. An undetected error (at AS, OS, and HW level) may be eventually detected in one of the following ways:

- The undetected error sooner or later propagates and reveals itself by means of activity `Y_tToDown`.
- Whenever the detailed model is producing a (potential) incorrect emission, a token is in place `CorruptDetCorrtIn` or `CorruptDetIncorrIn` depending on whether it was generated by a correct or incorrect input, respectively. The instantaneous activities `failureDet1` and `failureDet2` implement the corresponding output interface checks. They reveal the (potential) incorrect emission with probability *outputCoverage* by means of the upper case of such activities. In this case, a diagnosis of the system is triggered and implemented by the output gates `gDetected` and `gDetected2`: possible undetected erroneous states become detected by

moving a possible token from the Y_DownNoDet places of the AS, OS, and HW to the corresponding places Y_DownDet. These output gates also move the token removed by the firing of activities failureDet1 and failureDet2 from places CorruptedDetCorrIn or CorruptedDetIncorrIn in places noOutCorr and noOutIncorr, respectively. If the lower case of activities failureDet1 and failureDet2 is chosen, the token is moved in place Corrupted.

- A repair of the OS may allow detecting an undetected error at the AS level with probability *OS_errorDet* by means of the lower case of the exponential activity OS_repair. In this case, if there is a token in place AS_DownNoDet it is moved in AS_DownDet by means of the output gate gOS_det.

- A repair of the HW may allow detecting an undetected error either at the AS or OS level with probability *HW_errorDet* by means of the lower case of the exponential activity HW_repair. In this case, if there is a token either in places AS_DownNoDet or OS_DownNoDet it is moved in AS_DownDet or OS_DownDet, respectively, by means of the output gate gHW_det.

The whole model output process is as follows: only one token circulates in the subnet composed by places start, spurious, noOutCorr, noOutIncorr, CorruptedDetCorrIn, CorruptedDetIncorrIn, Corrupted, and Correct. (notice that only one token circulates also in each AS, OS, and HW detailed models). The exponential activity lambdaInToOut models the output process and fires with rate $\lambda^{IN \rightarrow OUT, X} = \lambda^{IN, X} + \lambda^{S, X}$. For the ITMU it represents its responsiveness in capturing alarming conditions from the incoming reports on the status of the controlled network. For the RMU and GMU it represents the answer to an alarm from the ITMU and RMU, respectively. Moreover, activity lambdaInToOut has two cases (options). If a spurious output has been produced (a token is in the place spurious) the second case is chosen. Otherwise the first case corresponds to the elaboration of a correct input (which is supposed to have a probability α), while the second to an incorrect input, with probability 1-α.

The elaboration of a correct input and of the spurious outputs is governed by means of the output gate gOutput according to the following rules. If there is a false alarm (spurious->Mark() == 1) the component produces an outcome surely corrupted (the token is moved in place Corrupted). Otherwise three cases are possible:
i) If at least one error has been internally detected at the AS, OS or HW level (a token is in DownDet), no output is produced (the token is moved in place noOutIncorr),

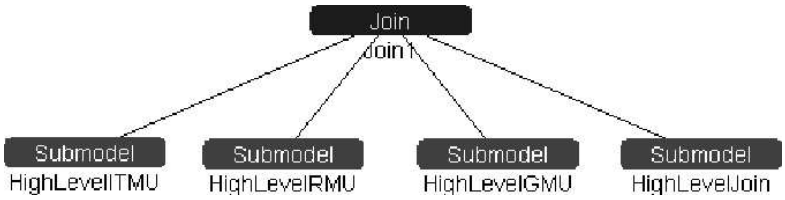

Figure 3.16: Composed model at GMU decision level

ii) If all components are in state up (a token is in the place `Up`) a correct output is produced (the token is moved in place `Correct`),
iii) Otherwise a (potential) corrupted output is produced (the token is moved in place `CorrupDetCorrIn`). Notice that the place `spurious` can hold at most one token, which is removed by the gate `gOutput` whenever such a spurious output has been processed.

The elaboration of an incorrect input is governed by means of the output gate `gOutput2` according to the following rules. If the incorrect input has been detected by the input interface checks (`Place1->Mark() == 1`) or there exists at least one error detected at the AS, OS or HW level (a token is in `DownDet`), the component produces no output (the token is moved in place `NoOutCor`). Otherwise the system (potentially) yields an incorrect emission (the token is moved in `CorruptDetIncorrIn`). Notice that the marking of place `Place1` in the subnet composed by places `Place1` and `Place2` and the timed activities `lambda1` and `lambda2` model the probability that the input interface checks are able to detect an incorrect input. At most one token circulates in such a subnet (no tokens in case the probability to detect an incorrect input by the input interface checks is zero).

3.5.4 The Overall Model

The overall model for the CAUTION++ instance under analysis has been constructed under the following assumptions:

- Messages coming from different ITMUs and RMUs are indistinguishable.
- The RMUs and the GMUs process the incoming input requests (from the ITMUs and RMUs respectively) individually and sequentially.

Figure 3.16 shows the SAN composed model for analyzing the CAUTION++ behavior at a single GMU decision level (e.g., to evaluate the probability of correctness of a reconfiguration decision issued by a GMU). Thanks to the above assumptions, the evaluation of the whole CAUTION++ instance is easily obtained by mathematically combining the evaluations at single GMU level, in accordance

with the specific measure under analysis. It is achieved by combining the abstract models of ITMU, RMU, and GMU (*HighLevelITMU*, *HighLevelRMU*, and *HighLevelRMU*, respectively), all having the same structure as shown in Figure 3.12. These models differ only for the values of their parameters, which are derived from their corresponding detailed models. All the models are joined together and along with the model *HighLevelJoin* (shown in detail in Figure 3.13) according to the following scheme: the input gate `gInput` of the HighLevelITMU (see Figure 3.12) is connected with the place `start` of the *HighLevelJoin* model. The *HighLevelITMU* model is also connected with the input gate `gInput` of the *HighLevelRMU* through places `ITMUCorrupted` and `ITMUCorrect`. Similarly, The *HighLevelRMU* model is also connected with the input gate `gInput` of the *HighLevelGMU* through places `ITMUCorrupted` and `ITMUCorrect`.

3.6 Evaluation results

The preceding models have been numerically solved using the analytical solver provided by the Möbius tool [61]. Since all the timed activities are exponentially distributed and the state space dimension of the models was not huge, it was possible to pursue an analytical solution achieving more accurate results than through simulation. Given the nature of the measures of interest, we resorted to a steady-state analysis for all models.

3.6.1 Settings for the numerical evaluation

The developed models have a number of internal parameters, to which values have to be assigned. For many of them, reference values from manufactures or previous studies in the literature are available. For others, mainly those concerning the components to be developed in the CAUTION++ framework, this is not true and the choice of appropriate values is more critical. Therefore, for such critical parameters, a range of values is experimented in the analysis, to determine the impact of such variations on the analyzed dependability figures (sensitivity analysis).

The meaning of the parameters in Table 3.2 is as follows:

- α_{ITMU}, α_{RMU} and α_{GMU} are the probabilities that the input provided to ITMU, RMU and GMU, respectively, is correct;
- MTBA_ITMU, MTBA_RMU and MTBA_GMU are the mean time between two inputs to ITMU, RMU and GMU, respectively (in the case of ITMU, it is the mean time between two external inputs for which ITMU generates an alarm to RMU);

Parameter	Value	Range
α_{ITMU}	-	0.90 - 0.999
α_{RMU}	-	from ITMU
α_{GMU}	-	from RMU
MTBA_ITMU	-	2 - 48 (hours)
MTBA_RMU	-	from ITMU
MTBA_GMU	-	from RMU
MTBFA_X	-	198 - 2000 (hours)
inputCoverage_X	-	0.00 - 1.00
outputCoverage_X	-	0.00 - 1.00
AS_Coverage_X	-	0.70 - 0.999
AS_MTTF_X	400 (hours)	-
AS_MTTRdet_X	0.2 (hours)	-
AS_noDetLatency_X	2 (hours)	-
OS_Coverage_X	0.80	-
OS_MTTF_X	400 (hours)	-
OS_MTTRdet_X	2 (hours)	-
OS_noDetLatency_X	2 (hours)	-
OS_errorDet_X	0.50	-
HW_Coverage_X	0.90	-
HW_MTTF_X	100000 (hours)	-
HW_MTTRdet_X	2 (hours)	-
HW_noDetLatency_X	2 (hours)	-
HW_errorDet_X	0.50	-

Table 3.2: Detailed model parameters and their values

- MTBFA_X is the mean time between two spurious outputs emitted by a generic component X;
- inputCoverage_X is the coverage of the error detection checks of a generic component X at input interface;
- outputCoverage_X is the coverage of the error detection checks of a generic component X at output interface;
- AS_Coverage_X, OS_Coverage_X and HW_Coverage_X are, respectively, the coverage of the checks at AS, OS and HW level, for a generic component X;
- AS_MTTF_X, OS_MTTF_X and HW_MTTF_X are, respectively, the mean time to failure of the AS, OS and HW of a generic component X;
- AS_MTTRdet_X, OS_MTTRdet_X and HW_MTTRdet_X are, respectively, the mean time to repair of the AS, OS and HW of a generic component X;
- AS_noDetLatency_X, OS_noDetLatency_X and HW_noDetLatency_X are, respectively, the latency of the undetected failure in the AS, OS and HW of a generic component X;
- OS_errorDet_X is the probability that the repair of the OS removes an undetected error at the AS level;

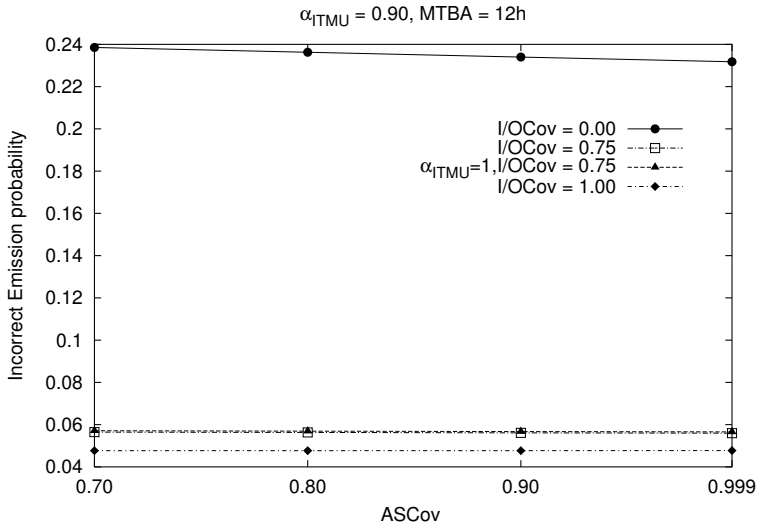

Figure 3.17: Incorrect emission probability related to Operator 1 (or Operator 3)

- HW_errorDet_X is the probability that the repair of the HW removes an undetected error at the OS level.

3.6.2 Numerical evaluation

In this Section, we present and discuss the results obtained.

To keep the notation in the figures as light as possible, we indicate with I/OCov the coverage of the input and output interface (which is the same for ITMU, RMU and GMU), and with ASCov the coverage of the application software (again, it is the same for ITMU, RMU and GMU).

Figure 3.17 shows the probability of incorrect emission of the GMU managed by Operator1 (it is actually the same for Operator3 also), at varying values of the coverage of the I/O Interface Checks and the coverage of the Application Software. The probability of incorrect emission decreases as the probability of coverage of the I/O Interface Checks increases; instead, it is very lightly influenced by As Coverage. Looking at the two overlapping curves, it can be observed that the impact of the correctness of the input to ITMU is not relevant. Therefore concerning the emission failure probability, significant benefits are achieved using the Interface Checks, since more incorrect messages are detected and no output is produced in these cases.

Figure 3.18 shows the reliability of the trial system at varying the observation time, that is the overall probability that the system does not undertake

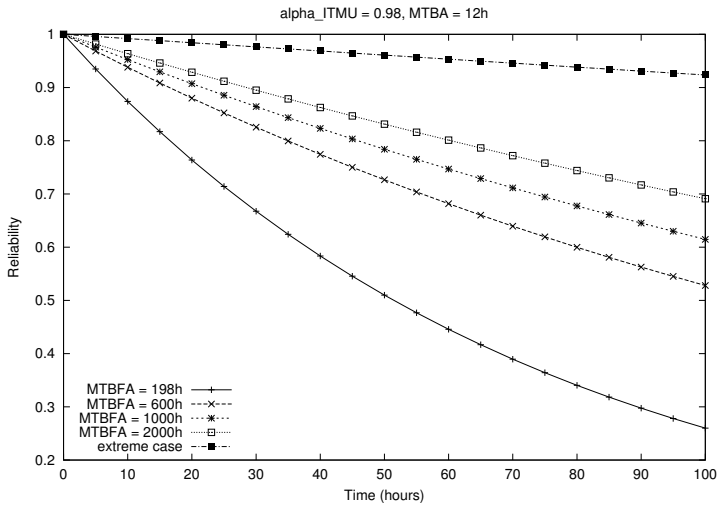

Figure 3.18: Reliability of the trial system

wrong actions. We suppose that the overall system fails if at least one root component fails or, equivalently, one GMU undertake a wrong action. The reliability of the trial system at time t is then equal to $e^{\frac{-t}{MTTF}}$, where $MTTF = (MTTF_1 + MTTF_2 + MTTF_3)$ and $MTTF_i$ is the mean time to failure related to operator i, with $i = 1, 2, 3$. Therefore, we have solved the three operators sub-nets separately, and then we have obtained the reliability for the whole system by exploiting the previous formula.

The plots have been obtained by fixing the mean time between alarms to 12 hours and the probability of correct input to ITMU to 0.98. The varying parameter is the MTBFA. The reliability of the system quickly decreases at lower values of MTBFA. In the figure, also an "extreme case" curve is plotted, obtained considering totally correct the external input to the ITMU, and assuming a very high coverage (0.99) for all the employed error detection mechanisms. The idea was to understand how would be the reliability of the CAUTION++ instance, in case a highly robust implementation of the CAUTION++ components is performed and in absence of faults external to the system. It can be appreciated that in such a case the reliability curve has a very good trend.

Despite the insertion of CAUTION++ induces a small reliability penalty (as exemplified by Figure 3.18), it is nevertheless very beneficial, since CAUTION++ allows to increase the resource utilization of the underlying networks through a cooperation among them. This is the final goal of the project that justifies the existence of system and the consequently introduction of new errors.

Figure 3.19 and Figure 3.20 are plotted at varying values of the mean time be-

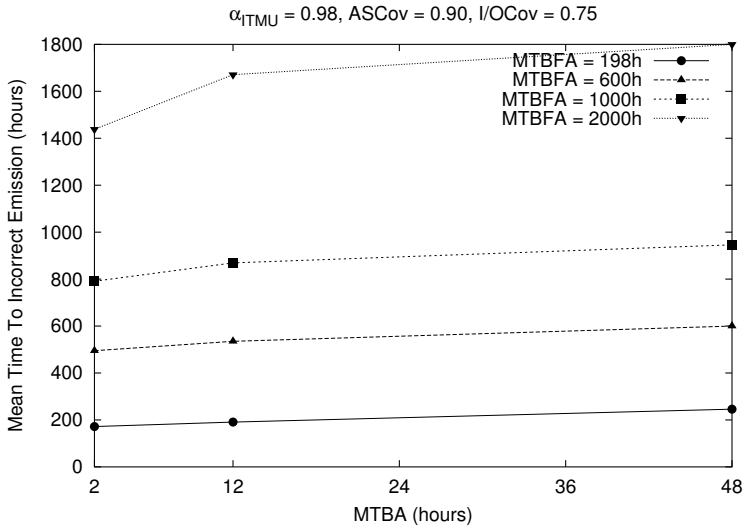

Figure 3.19: Mean time to incorrect emission for Operator 1 (or Operator 3)

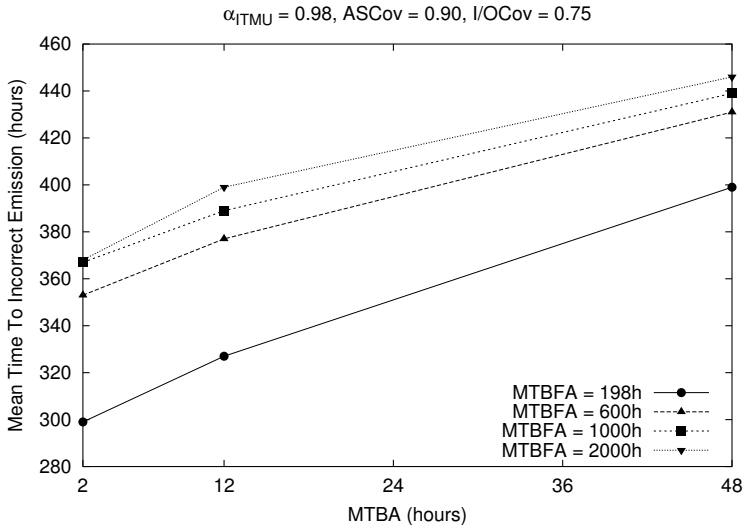

Figure 3.20: Mean time to incorrect emission for Operator 2

tween alarms and the mean time between spurious outputs, and setting to 0.98 the probability that the input to ITMU is correct. Not surprisingly, all the curves follow an increasing trend. Note that the time to an incorrect emission is significantly different for Operator 1 (or Operator 3) and Operator 2.

3.7 Summary

This Chapter has focused on a methodology for quantitative dependability evaluation of systems structured in a hierarchical fashion and on its application to a case study.

In more details, in the first part an efficient modeling methodology has been presented, consisting in defining "abstract" and "detailed" models of the system components, so as to reduce complexity and gain efficiency both at model design and at model solution levels.

In the second part, an instance of the CAUTION++ architecture has been selected, as a representative case study of the class of systems our methodology is directed to. In accordance with the basic dependability requirements stated in CAUTION++, the evaluated dependability indicators have been the probability of an incorrect output emission, the Mean Time to Failure of a GMU component and the reliability of the whole instance. We resorted to an analytical solution, using the automatic Möbius tool.

Thanks to the application of our modeling methodology and resolution technique, the biggest model solved had less than 1000 states, and the time needed to perform a single study did never exceed one minute on a Pentium M 1.3 GHz, 512Mb Ram PC. Actually, most of the time required to the resolution technique is due to the manual passing of the parameters' values between the detailed models and from these to the abstract one. Such waste of time could be significantly reduced using some appropriate automatic tools.

Of course, there is still work to do in evaluating the effectiveness of the methodology in more complex scenarios. Anyway, the indications that we are able to provide at the moment, as derived from this study, seem to be very encouraging.

Chapter 4

The general Modeling and Solution framework

In Chapters 2 and 3 we have described two modeling and solution techniques applicable to two specific classes of systems, a GPRS infrastructure and a hierarchical control system. If we analyze these two techniques from an abstract point of view, we realize that they have some common characteristics.

First of all, they are both based on a decomposition approach: the overall system is seen as a set of interacting subsystems (the couples of cells in the GPRS infrastructure - the hardware/software components in the multi-stage representation of the hierarchical control system) that can be modeled and solved in isolation, using some external inputs when required. Moreover, they both require the definition of a solution scheme that determines the order in which the submodels have to be solved and the intermediate results that have to be passed between them (Figure 2.4 for the GPRS infrastructure - Figure 3.9 for the hierarchical control system). Unfortunately, they also have another (negative) common property: they provide a very useful insight in their specific context, but they could be hardly reused to analyze quite different classes of systems, as they have been developed considering a domain-specific environment.

This last consideration has induced us to define a modeling and solution framework, based on a decomposition approach, whose application-domain is not restricted to any particular class of systems. Such generalization concerns both the modeling phase and the solution process: the required models are identified with respect to the functions they perform, without detailing their actual implementation or the adopted modeling formalism, and the solution process only defines the solution scheme, without detailing the type of solver to be used (analytical/numerical or simulative).

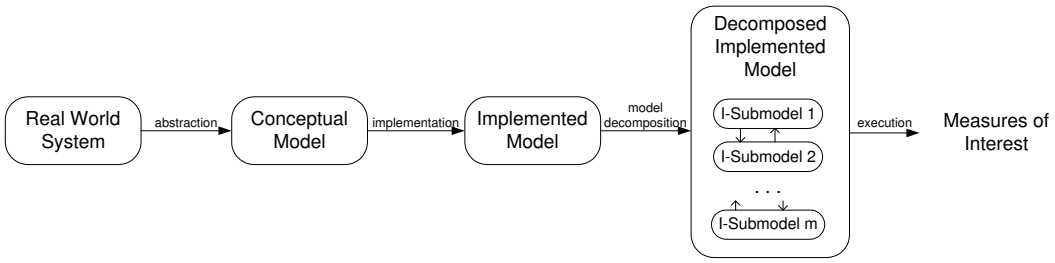

Figure 4.1: Modeling and solution process with implemental-level decomposition

4.1 Introduction

Every system can be seen, at different levels of abstraction, as composed of a number of interacting subsystems: the real world system, for example, can be seen as a set of countries exchanging goods and persons; internet can be seen as a set of computers that communicate and transfer data; a human cell consists of a number of interacting smaller units of life, and so on. Moreover, the system decomposition is not unique, as we can identify different system decompositions corresponding to different levels of abstraction: a server can be seen as a black box that produces an output for a given input, but it could also be seen as a set of interacting processes (tasks) that cooperate to produce the same output. It is clear that the higher is the level of detail required to capture the system behavior, the higher is the complexity of the system to be modeled and solved.

These considerations show the importance of the choice of a particular system decomposition, that is always a tradeoff between correctness of representation of the real system behavior (with respect to the measures of interest) and capability to solve the corresponding models.

In Figure 4.1 we illustrate the modeling and solution process adopted in most of the existing works that try to master system complexity using decomposition. The starting point is a real world system (either actual or hypothesized) that we want to model and analyze in order to obtain the measures of interest. The first step is the development of a conceptual model, a way of "thinking about" and representing the real world system. A crucial design decision is the determination of which factors influence the system behavior, the system behaviors to be incorporated into the model, and the representation of those behaviors. The choice of an appropriate conceptual model is not unique and it depends on: i) the measures of interest to be computed, ii) the computational and development resources available to build, validate, and use the model, and iii) the available data describing the real-world system and its interfaces. We refer to this determination as abstraction, since the conceptual model is a simpler representation of the more complex real-world system. The conceptual model is then implemented using an appropriate

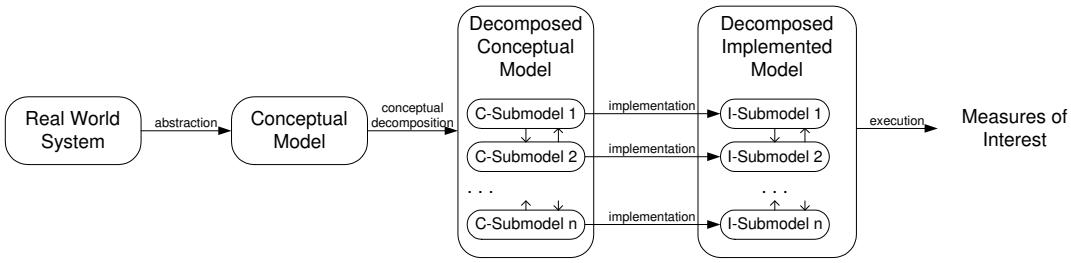

Figure 4.2: Modeling and solution process with conceptual-level decomposition

modeling formalism, thus obtaining a single implemented model. At this point an appropriate decomposition technique can be applied to decompose the original model in a set of independent and simpler to solve submodels that can be solved separately (perhaps sharing some intermediate results), finally obtaining the measures of interest.

Instead of applying the decomposition techniques at the model-level, that is decoupling parts of a pre-existing model as depicted in Figure 4.1, we aim to apply the decomposition approach at the conceptual-level, thus identifying a set of interacting conceptual submodels that can be implemented using different formalisms and solved in isolation using different appropriate solution techniques, exchanging intermediate results when required (see Figure 4.2).

The application of the decomposition technique at the conceptual-level enables us to decompose the system in a very natural way. Moreover, each conceptual submodel can be implemented using the formalism that better represents its behavior, and can be solved using an appropriate and efficient solution technique.

This Chapter is organized as follows. In Section 4.2 we formalize the system-level decomposition. The modeling approach is presented in Section 4.3: first we show how to build in a modular way the single model representing the whole system, and then we describe how it can be decomposed in a set of more simple submodels to be solved separately. In Section 4.4 we propose two algorithms implementing the solution process, and we provide some considerations about their effectiveness and about the accuracy of the final results. Section 4.5 outlines the main characteristics that an automated tool should have in order to support the proposed modeling and solution framework, while conclusions are drawn in Section 4.6.

4.2 The proposed interaction-based decomposition technique

As inspired by [8], we first analyze a system from a functional (or logical, conceptual) point of view (functional decomposition). The overall system is decomposed in a set of interacting subsystems, that we call "entities", each one corresponding to a critical system function with respect to the validation objectives. Therefore, the behavior of an entity corresponds to the critical system function it performs. Like the inputs of a function may depend on the output produced by another function, similarly the behavior of an entity may depend on the behavior of another entity. We say that there exists a "dependency relation" (or "dependency connection") from entity X to entity Y during a phase $k = [t_0; t_1]$ (interval of time), that we denote with $X \rightarrow_k Y$, when the behavior of Y depends on the behavior of X during the period starting from time t_0 and ending at time t_1.

As inspired by [14], the system lifetime can be seen as a sequence of phases in which each phase is characterized by some properties. In particular, we define a phase as the period during which the dependency relations (interactions) holding between the entities remain fixed, while vary between two successive phases. Therefore, the lifetime of the system may be seen as a sequence of phases in which two consecutive phases have at least one different dependency relation between entities (temporal decomposition).

For each phase k and for each entity X, we can identify the direct dependencies involving X with a 3-tuple $< \Omega_X; X; \Theta_X >$ in which Ω_X is the set of the entities that directly affect X (that is $\Omega_X = \{W_1, \ldots, W_m\} \rightarrow_k X$) and Θ_X is the set of entities that are directly affected by X (that is $X \rightarrow_k \{Y_1, \ldots, Y_n\} = \Theta_X$).

Therefore, we can define the set E^k as the set of all the tuples (one tuple for each entity) that do not change during phase k, that is:

$$E^k = \{< \Omega_X; X; \Theta_X > | \Omega_X \rightarrow_k X \text{ and } X \rightarrow_k \Theta_X\} \quad . \tag{4.1}$$

We say that in a phase k an entity is:

- *active* if $< \emptyset; X; \Theta_X \neq \emptyset > \in E^k$, that is X affects entities in Θ_X but it is not affected by any entity ($\Omega_X = \emptyset$);
- *passive* if $< \Omega_X \neq \emptyset; X; \emptyset > \in E^k$, that is X is affected by entities in Ω_X but it does not affect any entity ($\Theta_X = \emptyset$);
- *mixed* if $< \Omega_X \neq \emptyset; X; \Theta_X \neq \emptyset > \in E^k$, that is X is affected by entities in Ω_X and, at the same time, it affects the entities in Θ_X;

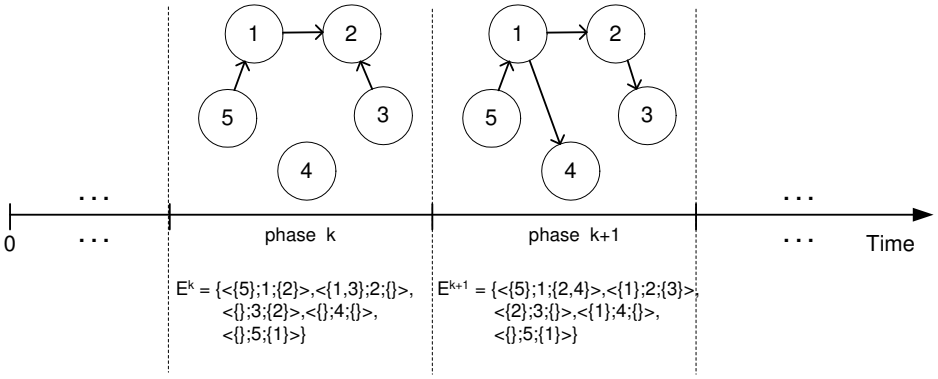

Figure 4.3: Dependency relations in two consecutive phases

- *neutral* if $< \emptyset; X; \emptyset > \in E^k$, that is X neither is affected nor affects any entity ($\Omega_X = \emptyset$ and $\Theta_X = \emptyset$).

Therefore, in each phase the entities can be partitioned with respect to the role they play during the phase.

Thus the application of the functional and temporal decomposition to a system generates another system, that we call "*phased-interacting system*", that is equivalent to the original one but it is built considering some critical subsystems (each one performing a critical function) and the interactions between them (here we consider two systems to be equivalent if they have exactly the same behavior).

In Figure 4.3 we show a simple example of phased-interacting system composed of five entities (numbered from '1' to '5'), focusing on two consecutive phases. From phase k to $k+1$, entity '1' has a new dependency connection to entity '4', while entity '3' transforms from active to passive. The corresponding set E^k and E^{k+1} are shown at the bottom of the Figure. During phase k entities '3' and '5' are active, '2' is passive, '1' is mixed and '4' is neutral. During phase $k+1$ entity '5' is active, '3' and '4' are passive while '1' and '2' are mixed.

We call "dependency connection graph" the graph describing the dependency relations holding between entities in a phase, where the nodes are the entities and the directed arcs are the dependency relations (see the example of Figure 4.3). The dependency connection graph for phase k and the set E^k are two different but equivalent ways to represent the dependency relations during the phase, as we can easily build the dependency connection graph starting from the corresponding set of tuples and viceversa.

4.3 The modeling approach for a phased-interacting system

In the previous Section we depicted how to logically represent a complex system as a set of interacting subsystems (the entities), whose interactions may change during the system lifetime. At this point we have to take advantage of the system decomposition proposing a modeling approach that also includes some capabilities in managing the system complexity.

The modeling approach consists of two steps. In the first step we build the whole model representing the behavior of the generated phased-interacting system. Although built following a modular approach, the complexity of the whole model is still huge, as the modularity does not necessarily correspond to a decomposition of the solution process and, then, to a mitigation of the model complexity. Therefore, in the second step, we properly modify the structure of the whole model, thus enabling the application of a decomposed solution process that consists in solving a set of simpler submodels in isolation, possibly sharing some intermediate results.

4.3.1 The first step of the modeling approach: the whole model structure definition

The first step in modeling a phased-interacting system, that is a system generated by the application of the functional and temporal decomposition, is to construct a whole model representing the behavior and the interactions of all the entities in all the phases.

Following a modular approach, in Figure 4.4 we identify the main submodels composing the whole model. It is important to note that we are not considering a specific modeling formalism, as each submodel could be described using an appropriate formalism different from that adopted for other submodels. Our goal is to identify the main submodels with respect to the functions they have to perform, without detailing how these functions are actually implemented.

The "Neutral Entity X" model (with $X = 1, \ldots, N$, where N is the total number of entities), denoted with X° for brevity, represents the behavior of entity X when it is neutral, that is when it is isolated, and it does not change during the system lifetime. Its behavior can be specified to be active, passive or mixed through the "Dependency Relations" model, that consists of a set of submodels representing all the possible dependency relations existing between entities during all the phases. Each submodel $\varepsilon^{(k)}_{X \to \Theta_X}$, with $X \in \{1, \ldots, N\}$ and $k \in \{1, \ldots, p\}$, represents the dependency relation $X \to_k \Theta_X$, that is it defines how entity X affects the entities belonging to Θ_X during phase k.

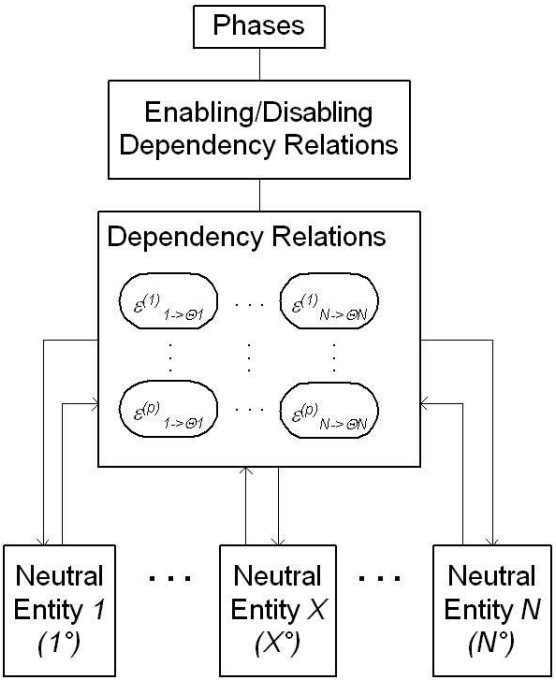

Figure 4.4: The whole model for a phased-interacting system

As previously specified, X° is a model that represents the behavior of entity X when it works as neutral, that is when it does not interact with any entity, but in general an entity can act as active, passive or mixed, in accordance with the dependency relations identified for the considered phase. Suppose for example that, during phase k, entity X has to be active and affects entities $\{Y_1, \ldots, Y_n\}$, that are passive. To do this, we modify the behavior of X° "connecting" it with the submodel $\varepsilon^{(k)}_{X \to \{Y_1,\ldots,Y_n\}}$ that modifies the behavior of the neutral entity X to be active during phase k. In turn, the $\varepsilon^{(k)}_{X \to \{Y_1,\ldots,Y_n\}}$ submodel has to be "connected" with the models $Y_1^\circ, \ldots, Y_n^\circ$, thus modifying their behavior from neutral to passive. The term "connected" is unavoidably very general and its exact meaning depends on the specific formalism used to build the submodels (e.g. using SAN formalism it may correspond to share some places between the two models applying the Join operator). At this high level of abstraction, the only issue is that the behavior of a model can be properly modified "connecting" it with another submodel. In general, if $< \Omega_X; X; \Theta_X = \{Y_1, \ldots, Y_n\} > \in E^k$, the proper behavior of the entities involved in the dependency relation is obtained connecting the model X° with the submodel $\varepsilon^{(k)}_{X \to \Theta_X}$ that, in turn, is connected with the models $Y_1^\circ, \ldots, Y_n^\circ$. Therefore, the behavior of the entities $X, Y_1, \ldots, Y_n$ during phase k is modified by the

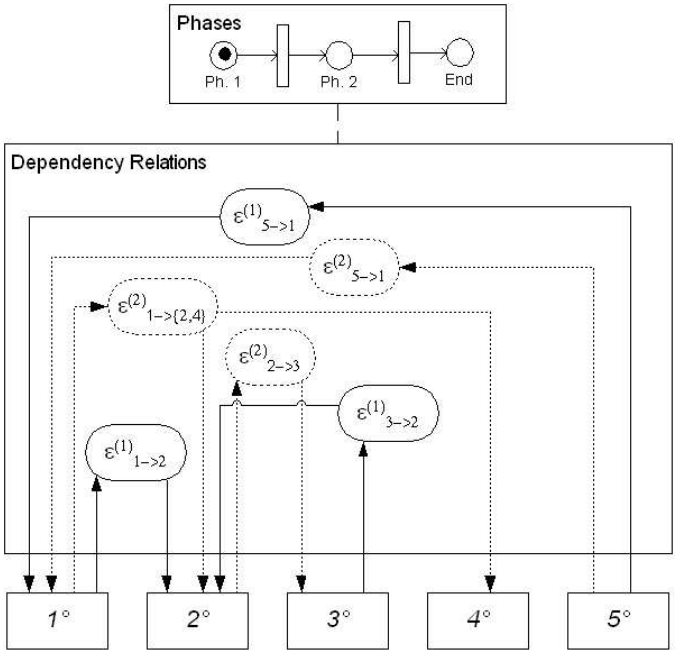

Figure 4.5: Snapshot of the whole model configuration during phase 1, an example

$\varepsilon^{(k)}_{X \to \Theta_X}$ submodel according to the existing dependency relations. Obviously not all the dependency relations hold in a phase, and then the "Enabling/ Disabling Dependency Relations" model enables only those relationships that exist in the current phase, while the others are disabled. Finally, the "Phases" model identifies the current phase of the system.

For a better understanding, in Figures 4.5 and 4.6 we specify the structure of the whole model for the example depicted in Figure 4.3, supposing that there are only two phases ($p = 2$, where p is the number of phases). For simplicity, the "Phases" model has been defined as a generalized stochastic Petri net (GSPN), and the disabled dependency relations have been depicted with a dashed line.

At this point we have identified the main submodels that have to be implemented in order to capture the behavior of a phased-interacting system. But what about complexity? It is important to underline that the modularity of the proposed modelling approach alone cannot be truly effective without a modular solution of the defined models. Therefore, as the complexity problem needs to be attacked both from the point of view of system representation and of the underlying model solution, we need to perform a further step towards the modeling approach definition in order to include some capabilities to reduce the solution complexity.

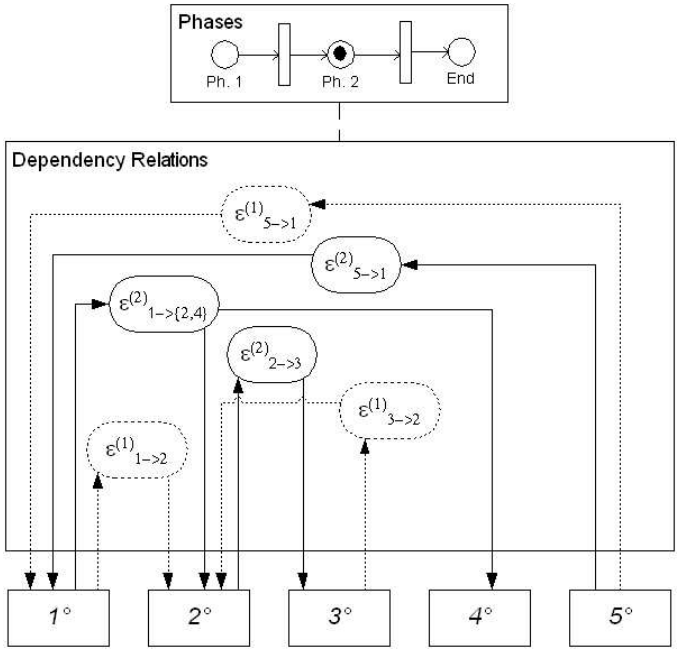

Figure 4.6: Snapshot of the whole model configuration during phase 2, an example

4.3.2 The second step towards the modeling approach: the whole model decomposition

The idea is to decompose the whole model of Figure 4.4, representing a phased-interacting system, in a set of simpler submodels to be solved separately, sharing some intermediate results that capture the effects of the dependency relations between entities.

The way in which we decompose the whole model has been inspired by [17]. In this work the authors decouple parts of a model identifying submodels that are not affected by the rest of a model (isolated submodels), and solving them separately. A result from each solved submodel is then used in the solution of the rest of the model.

If we concentrate on a single phase k, the model X° is certainly isolated when it is not connected to any other entity's model, and it happens when entity X is neutral. In this case the entity's model can be solved separately as it neither affects nor is affected by any other entity's model. If, on the contrary, entity X is active, passive or mixed, the model X° will be connected to other entities' models through the corresponding dependency relation models, and then it is clearly not isolated. In these cases, the only way to decompose the whole model is to split the dependency relations holding between entities, that is decoupling each dependency relation model in a set of interacting separated submodels. This "splitting

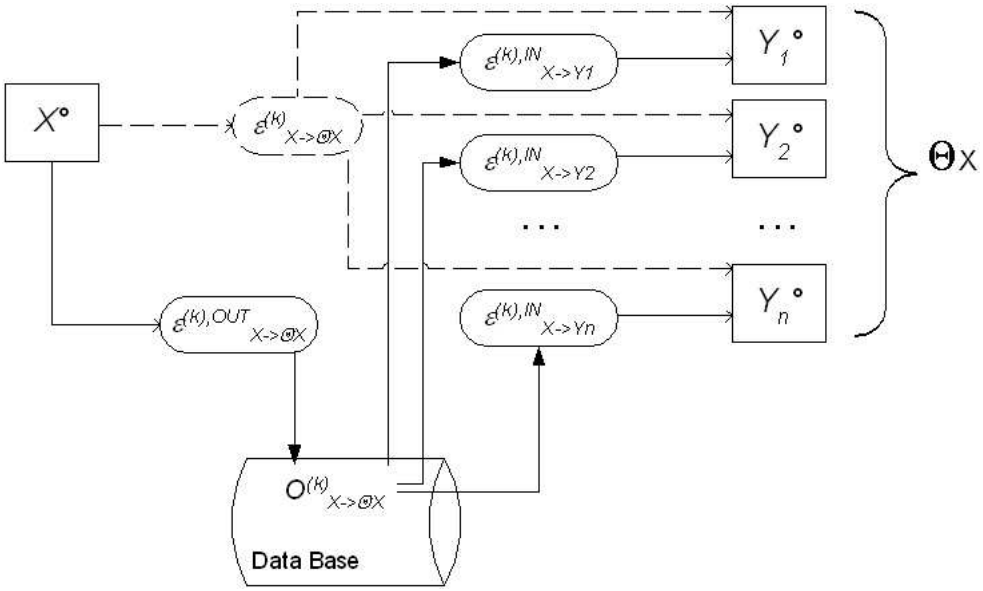

Figure 4.7: The splitting procedure

procedure" removes all interactions between the entities' models during their execution, thus enabling the separate solution process, and the interactions are replaced by exchange of numerical results (physical decomposition). Therefore, the interactions between the separated submodels must be captured in the numerical results that are passed between them.

In more detail, suppose to have an element $< \Omega_X; X; \Theta_X > \in E^k$ such that $\Theta_X = \{Y_1, \ldots, Y_n\} \neq \emptyset$. The splitting procedure consists in decoupling the model $\varepsilon^{(k)}_{X \to \Theta_X}$, representing the dependency relation $X \to_k \Theta_X$, in a set of separate submodels. Figure 4.7 graphically shows the application of the splitting procedure. The original $\varepsilon^{(k)}_{X \to \Theta_X}$ model (depicted with a dashed line), is replaced by the following models:

- $\varepsilon^{(k),OUT}_{X \to \Theta_X}$, that
  - modifies the neutral behavior of entity X during phase k according to the dependency relation $X \to_k \Theta_X$; therefore, the models $\varepsilon^{(k),OUT}_{X \to \Theta_X}$ and $\varepsilon^{(k)}_{X \to \Theta_X}$ affect the model X° in the same way;
  - includes all the information needed to produce and write on a shared data base the intermediate result $O^{(k)}_{X \to \Theta_X} = [O^{(k)}_{X \to Y_1}, \ldots, O^{(k)}_{X \to Y_n}]$, that is an array representing the effects that entity X has on $Y_1, \ldots, Y_n$ during phase k. This is the result that captures the interactions between the entities;

- $\varepsilon^{(k),IN}_{X\to Y_1}, \ldots, \varepsilon^{(k),IN}_{X\to Y_n}$, that
  - modify the neutral behavior of the entities belonging to Θ_X during phase k according to the dependency relation $X \to_k \Theta_X$; therefore, the models $\varepsilon^{(k),IN}_{X\to Y_1}, \ldots, \varepsilon^{(k),IN}_{X\to Y_n}$ and $\varepsilon^{(k)}_{X\to\Theta_X}$ affect the models $Y_1^\circ, \ldots, Y_n^\circ$ in the same way;
  - includes all the information needed to read from a shared data base and use the previously produced result $O^{(k)}_{X\to\Theta_X}$ in order to capture the effects that entity X has on the entities belonging to Θ_X during phase k. In particular, $O^{(k)}_{X\to Y_i}$, with $i \in \{1, \ldots n\}$, will be used inside the corresponding $\varepsilon^{(k),IN}_{X\to Y_i}$ model to set the value of a certain parameter in order to reconstruct the impact of entity X on Y_i.

In summary, the application of the splitting procedure produces a set of decoupled submodels that interact each other through the passing of numerical results: an intermediate result produced by the solution of a given submodel is used to set the value of an appropriate numerical parameter defined in another submodel.

The monolithic model presented in Figure 4.4 is then modified according to the splitting procedure and it results in the decomposed model structure depicted in Figure 4.8. The "Dependency Relations" model has been decomposed in two parts: the IN part for the models that read and use the intermediate results, and the OUT part for the models that produce and share the intermediate results. Again, the "Enabling/ Disabling Dependency Relations" model enables only those relationships that hold in the current phase, while the "Phases" model identifies the current phase of the system.

Finally, the "Execution Manager" can be seen as a program that implements the solution algorithm for the proposed modeling approach (e.g. it determines the sequence in which the separate submodels have to be solved), as better described in Section 4.4.

An example

Just to be concrete, we provide a very simple example that should clarify the application of the splitting procedure. Let us consider the Stochastic Petri Net (SPN) model of Figure 4.9 (upper part), representing a system composed by the entities X and Y (whose modeling details are hidden in the clouds) that, during phase k, interact through the dependency relation $X \to_k Y$. The dependency relation is modeled just considering a transition `t` that, when completes, removes a token from a place `p1` belonging to X° and, atomically, adds a token in place `p2` belonging to Y°. The "Enabling/ Disabling Dependency Relations" model (not shown in the example) enables transition `t` during phase k only.

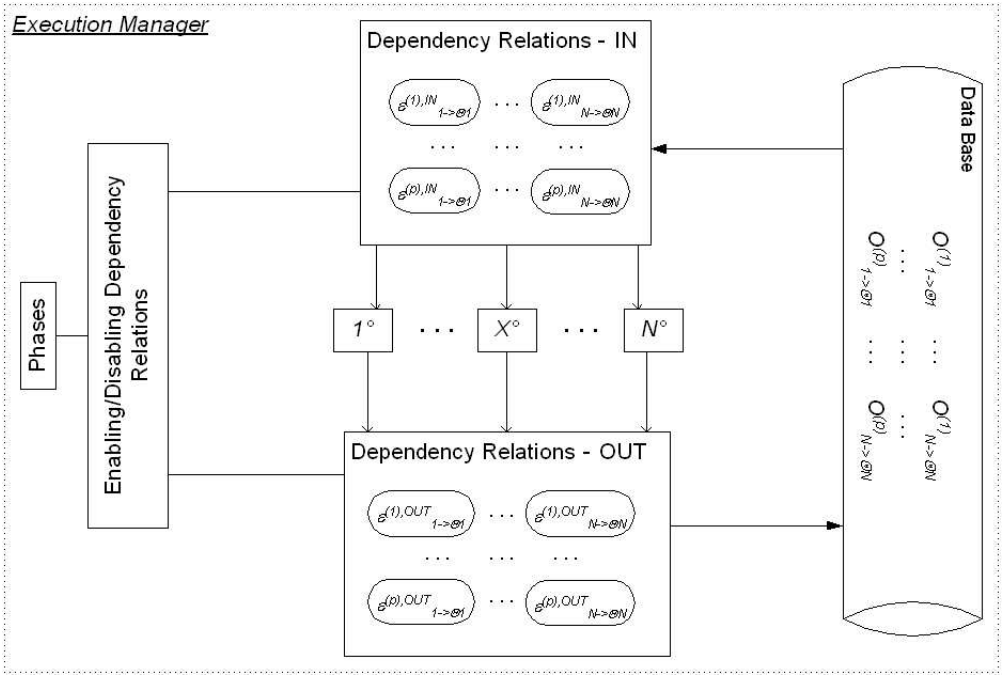

Figure 4.8: The decomposed model structure

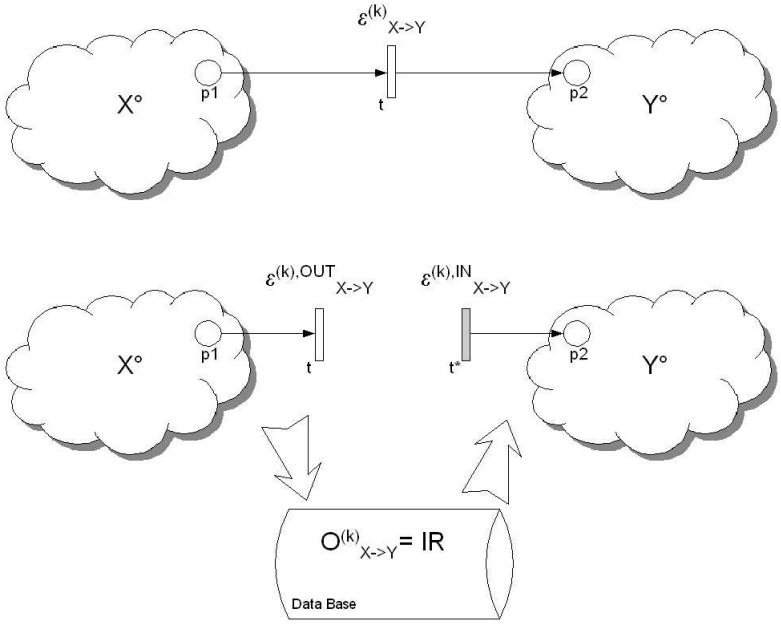

Figure 4.9: Example of application of the splitting procedure

The decomposed model structure that we obtain applying the splitting procedure is depicted in the lower part of Figure 4.9. The model $\varepsilon^{(k)}_{X\to Y}$ has been split in two parts. The $\varepsilon^{(k),OUT}_{X\to Y}$ submodel affects entity X as the $\varepsilon^{(k)}_{X\to Y}$ model does. The intermediate result $O^{(k)}_{X\to Y}$ is an array IR such that the i-th element IR[i] contains the mean time at which transition $\mathtt{t}$ fires for the i-th time. This result can be used by the $\varepsilon^{(k),IN}_{X\to Y}$ submodel to reconstruct the effects that entity X induces on Y. In this case $\mathtt{t}^*$ could be a deterministically distributed transition whose firing time is IR[0] for the first time, and IR[i]-IR[$i-1$] for the other completions (mean time elapsed between two consecutive firings). It is clear that the approximation of transition $\mathtt{t}$ with $\mathtt{t}^*$ can induce a degradation in the accuracy of the final results, and then this aspect has to be addressed very carefully. The deterministic transition $\mathtt{t}^*$ and the model Y° form, together, a Deterministic and Stochastic Petri Net (DSPN) model that will be solved using appropriate solution techniques.

4.4 The decomposed solution process

In this Section we define the solution process that can be applied to the decomposed model structure depicted in Figure 4.8. It mainly consists in a program (the "Execution Manager") that determines:

- the sequence in which the separate submodels have to be solved, in accordance with the existing dependency relations;
- the intermediate results that a separate submodel has to produce and/or use in its execution;
- the measures of interest.

Before detailing the two algorithms that we present in this dissertation (a standard version and an optimized one), we have to introduce some general definitions and notations that will be used in the algorithms' definitions.

The model for an active/passive/mixed entity

We previously defined X° as the model representing the behavior of entity X when it is neutral. Let $X^{(k)}$ be the model representing the real behavior of entity X during phase k, that is accounting for all the dependency relations involving X during phase k. Then

$$X^{(k)} = X^\circ \cup \varepsilon^{(k),OUT}_{X\to\Theta_X} \bigcup_Z \varepsilon^{(k),IN}_{Z\to X} \quad , \tag{4.2}$$

with $< \Omega_Z; Z; \Theta_Z > \in E^k$ and $X \in \Theta_Z$, where the symbol $\cup$ (join) has been here used to represent the "connections" between submodels, as described in Section 4.3.2. Equation 4.2 states, as expected, that the real behavior of entity X during phase k can be captured connecting (joining) the following submodels:

1. X°, representing the behavior of the entity when it is neutral;
2. $\varepsilon^{(k),OUT}_{X \to \Theta_X}$ (only if $< \Omega_X; X; \Theta_X > \in E^k$ with $\Theta_X \neq \emptyset$), representing the impact of the dependency relations in which X is active or mixed;
3. $\bigcup_Z \varepsilon^{(k),IN}_{Z \to X}$ (only if $< \Omega_X; X; \Theta_X > \in E^k$ with $\Omega_X \neq \emptyset$), representing the impact of the dependency relations in which X is passive or mixed.

Moreover, let $X^{(i,\ldots,j)}$ be the model representing the real behavior of entity X from phase i to phase j, thus accounting for all the dependency relations from phase i to phase j, with $i < j$. It immediately follows that

$$X^{(i,\ldots,j)} = X^{\circ} \bigcup_{u=i}^{j} (\varepsilon^{(u),OUT}_{X \to \Theta_X} \bigcup_{Z} \varepsilon^{(u),IN}_{Z \to X}) \quad . \tag{4.3}$$

Intermediate results and measures of interest

The intermediate result $O^{(k)}_{X \to \{Y_1,\ldots,Y_n\}=\Theta_X}$, produced by entity X during phase k, can be seen as a function f that takes as input the model representing the real behavior of entity X from phase 1 to phase k ($X^{(1,\ldots,k)}$) and all the intermediate results affecting X from phase 1 to phase k ($\bigcup_{j=1}^{k} O^{(j)}_{Z \to X}$), and produces as output the intermediate results for the entities $\{Y_1, \ldots, Y_n\}$ affected by X:

$$O^{(k)}_{X \to \{Y_1,\ldots,Y_n\}} = [O^{(k)}_{X \to Y_1}, \ldots, O^{(k)}_{X \to Y_n}] = f(X^{(1,\ldots,k)}, \bigcup_{j=1}^{k} O^{(j)}_{Z \to X}) \quad , \tag{4.4}$$

with $< \Omega_Z; Z; \Theta_Z > \in E^j$ and $X \in \Theta_Z$.

More generally we define $O^{(i,\ldots,j)}_X$, with $i \leq j$, as the set of all the intermediate results produced by entity X from phase i to phase j. Similarly, it can be seen as a function g that takes as input the model representing the real behavior of entity X from phase 1 to phase j ($X^{(1,\ldots,k)}$) and all the intermediate results affecting X from phase 1 to phase j ($\bigcup_{u=1}^{j} O^{(u)}_{Z \to X}$), and produces as output the set of intermediate results:

$$O^{(i,\ldots,j)}_X = \{O^{(i)}_{X \to \Theta_X}, \ldots, O^{(j)}_{X \to \Theta_X}\} = g(X^{(1,\ldots,k)}, \bigcup_{u=1}^{j} O^{(u)}_{Z \to X}) \quad , \tag{4.5}$$

with $< \Omega_Z; Z; \Theta_Z > \in E^u$ and $X \in \Theta_Z$.

Equivalently, we can define the measures of interest M_X for an entity X as a function h that takes as input the model representing the real behavior of entity X from phase 1 to phase p ($X^{(1,\ldots,p)}$, where p is the total number of phases) and all the intermediate results affecting X from phase 1 to phase p ($\bigcup_{j=1}^{p} O_{Z\rightarrow X}^{(j)}$), and produces as output the measures of interest relative to entity X:

$$M_X = h(X^{(1,\ldots,p)}, \bigcup_{j=1}^{p} O_{Z\rightarrow X}^{(j)}) \quad , \tag{4.6}$$

with $< \Omega_Z; Z; \Theta_Z > \in E^j$ and $X \in \Theta_Z$.

We note that we are assuming to be in the worst case in which the measures of interest can only be computed considering the behavior of the system during all its phases, that is during its whole lifetime. Moreover, we are supposing that the measures concerning more than one cell can be obtained through the composition of some partial measures regarding the single entities. For example, if $M_{X\&Y}$ is the measure of interest involving both the entities X and Y, we suppose that it is possible to identify and compute two different partial measures, M_X for entity X and M_Y for entity Y, that can be combined together through a function r in order to reconstruct the original measure of interest, that is $M_{X\&Y} = r(M_X, M_Y)$.

4.4.1 Standard Algorithm

The two algorithms that we present in this Chapter (a standard version and an optimized one) have been developed assuming that *the dependency connection graph in each phase is acyclic*. Actually this assumption limits the applicability of the solution processes that, therefore, can not be used for the solution of every phased-interacting system. In Section 4.4.3 we outline two possible ways to overcame this limitation.

The standard solution process consists of two consecutive and separate steps: dependencies removal and measures computation.

In the first step we remove the existing dependency relations between entities simply producing the intermediate results capturing and representing their interactions. The removal process starts from the first phase and, inside each phase, has to follow the order implicitly defined in the corresponding dependency connection graph. In particular, the dependency relation $X \rightarrow_k \Theta_X$ that corresponds to the element $< \Omega_X = \{W_1, \ldots, W_m\}; X; \Theta_X \neq \emptyset > \in E^k$ can only be removed in the following two cases:

1. if $\Omega_X = \emptyset$ (that is X is active), as it has no affecting entities;

2. if $\Omega_X \neq \emptyset$ (that is X is mixed), but all the dependency relations that affect entity X (that is the set $\{W_1 \rightarrow_k \Theta_{W_1}, \ldots, W_m \rightarrow_k \Theta_{W_m}\}$ such that $X \in \Theta_{W_1}, \ldots, X \in \Theta_{W_m}$) have already been removed.

We remove the dependency relation $X \rightarrow_k \Theta_X$ simply computing the corresponding output (intermediate result) $O^{(k)}_{X \rightarrow \Theta_X}$ and writing it to a shared data base. As previously mentioned, this computation requires to solve the model $X^{(k)}$ from phase 1 to phase k, taking as input all the intermediate results affecting X during the same period.

The measures of interest are finally computed in the second step of the algorithm, just considering each entity separated from the others (isolated). Each computation requires to solve the model $X^{(p)}$ from phase 1 to phase p, where p is the total number of phases, taking as input all the intermediate results affecting X during the same period.

In more detail, the standard solution process is described by the following algorithm in pseudo-code (p is the total number of phases in the system):

STANDARD ALGORITHM

STEP i): dependencies removal

$E'^k = E^k$;

for $k = 1, \ldots, p$ do {

  While ($\exists < \emptyset; X; \Theta_X \neq \emptyset > \in E^k$) do {

    Remove $< \emptyset; X; \Theta_X >$ from E^k;

    Read from db $\bigcup_{j=1}^{k} O^{(j)}_{Z \rightarrow X}$, with $< \Omega_Z; Z; \Theta_Z > \in E^j$ and $X \in \Theta_Z$;

    Compute $O^{(k)}_{X \rightarrow \Theta_X}$ and write it to db;

    Remove all the occurrences of X in each tuple of E^k;

  }

}

STEP ii): measures computation

$E^k = E'^k$;

for each entity X do {

  Read from db $\bigcup_{k=1}^{p} O^{(k)}_{Z \rightarrow X}$, with $< \Omega_Z; Z; \Theta_Z > \in E^k$ and $X \in \Theta_Z$;

  Compute M_X;

}

For a better understanding, in Table 4.1 we detail the execution of the algorithm for the example depicted in Figure 4.3, supposing that there are only two phases ($p = 2$). The columns indicate, from the left to the right: the step of the algorithm, the order in which the computations can be performed, the model rep-

Step	Order	$X^{(1,\ldots,k)}$	k	Inputs	Outputs
i)	1	$5^{\circ} \cup \varepsilon_{5\to1}^{(1),OUT}$	1		$O_{5\to1}^{(1)}$
i)	1	$3^{\circ} \cup \varepsilon_{3\to2}^{(1),OUT}$	1		$O_{3\to2}^{(1)}$
i)	2	$1^{\circ} \cup \varepsilon_{1\to2}^{(1),OUT} \cup \varepsilon_{5\to1}^{(1),IN}$	1	$O_{5\to1}^{(1)}$	$O_{1\to2}^{(1)}$
i)	3	$5^{\circ} \cup \varepsilon_{5\to1}^{(1),OUT} \cup \varepsilon_{5\to1}^{(2),OUT}$	2		$O_{5\to1}^{(2)}$
i)	4	$1^{\circ} \cup \varepsilon_{1\to2}^{(1),OUT} \cup \varepsilon_{1\to\{2,4\}}^{(2),OUT} \cup \varepsilon_{5\to1}^{(1),IN} \cup \varepsilon_{5\to1}^{(2),IN}$	2	$O_{5\to1}^{(1)} \cup O_{5\to1}^{(2)}$	$O_{1\to2}^{(2)} \cup O_{1\to4}^{(2)}$
i)	5	$2^{\circ} \cup \varepsilon_{2\to3}^{(2),OUT} \cup \varepsilon_{1\to2}^{(1),IN} \cup \varepsilon_{3\to2}^{(1),IN} \cup \varepsilon_{1\to2}^{(2),IN}$	2	$O_{1\to2}^{(1)} \cup O_{3\to2}^{(1)} \cup O_{1\to2}^{(2)}$	$O_{2\to3}^{(2)}$
ii)	6	$1^{\circ} \cup \varepsilon_{1\to2}^{(1),OUT} \cup \varepsilon_{1\to\{2,4\}}^{(2),OUT} \cup \varepsilon_{5\to1}^{(1),IN} \cup \varepsilon_{5\to1}^{(2),IN}$	2	$O_{5\to1}^{(1)} \cup O_{5\to1}^{(2)}$	M_1
ii)	6	$2^{\circ} \cup \varepsilon_{2\to3}^{(2),OUT} \cup \varepsilon_{1\to2}^{(1),IN} \cup \varepsilon_{3\to2}^{(1),IN} \cup \varepsilon_{1\to2}^{(2),IN}$	2	$O_{1\to2}^{(1)} \cup O_{3\to2}^{(1)} \cup O_{1\to2}^{(2)}$	M_2
ii)	6	$3^{\circ} \cup \varepsilon_{3\to2}^{(1),OUT} \cup \varepsilon_{2\to3}^{(2),IN}$	2	$O_{2\to3}^{(2)}$	M_3
ii)	6	$4^{\circ} \cup \varepsilon_{1\to4}^{(2),IN}$	2	$O_{1\to4}^{(2)}$	M_4
ii)	6	$5^{\circ} \cup \varepsilon_{5\to1}^{(1),OUT} \cup \varepsilon_{5\to1}^{(2),OUT}$	2		M_5

Table 4.1: Standard algorithm execution

resenting the real behavior of entity X from phase 1 to phase k, the number of phases under analysis (k), the intermediate results provided as input to the analyzed model, and the intermediate results produced as output.

In the first phase (when k=1) entities '3' and '5' are active, so their respective models $3^{(1)}$ and $5^{(1)}$ can be solved (in an unspecified order) to obtain the intermediate results $O_{3\to2}^{(1)}$ and $O_{5\to1}^{(1)}$, respectively. At this point the dependency relation between entity '5' and '1' (a mixed entity) has been removed, and then we can solve the model $1^{(1)}$ thus producing the output $O_{1\to2}^{(1)}$. Now we have removed all the existing dependency relations holding in phase 1, and then we pass to phase 2. The first dependency relation to be removed is that between entity '5' and '1'. Therefore we solve the model $5^{(1,2)}$ (representing the behavior of entity '5' during phase 1 and 2) obtaining the intermediate result $O_{5\to1}^{(2)}$. Then it is the turn of entity '1', and then its dependency relations with entities '2' and '4' are removed. The last model to be solved during `STEP i)` is $2^{(1,2)}$, producing the output $O_{2\to3}^{(2)}$. In `STEP ii)` of the solution process, the models representing the single entities are solved, thus producing the measures of interest.

4.4.2 Algorithm optimization

The main characteristic of the standard algorithm presented in the previous Section is to separate the phase in which the dependency relations are removed from the phase in which the measures of interest are computed. Actually we can optimize the solution algorithm computing, in some cases, the intermediate results and the measures of interest at the same time, thus reducing the total number of computations.

Suppose to execute `STEP i)` of the standard algorithm and to select the el-

ement $< \emptyset; X; \{Y_1, \ldots, Y_n\} \neq \emptyset > \in E^k$. As previously specified, it means that during phase k entity X is active, or it is mixed but all the dependency relations affecting it have already been removed. The optimized algorithm is based on the following observation:

If, from phase $k+1$ to phase p, X acts only as active or neutral,

1. then we can compute the set of intermediate results $O_X^{(k,\ldots,p)} = \{O^k_{X\rightarrow\Theta_X}, \ldots, O^p_{X\rightarrow\Theta_X}\}$ and the measure of interest M_X in the same computation, as entity X is no more affected by any other entity in the successive phases. This means that the corresponding model $X^{(1,\ldots,p)}$ already has the intermediate results required as input, and then it can be solved in isolation considering the whole system lifetime and producing both the measure of interest for entity X and the remaining intermediate results affecting the other entities.

2. otherwise, if $phase$ is the last phase starting from k in which X is active before it becomes passive or mixed, then we can obtain the intermediate results $O_X^{[k,\ldots,phase]} = \{O^k_{X\rightarrow\Theta_X}, \ldots, O^{phase}_{X\rightarrow\Theta_X}\}$ in only one computation. This means that the corresponding model $X^{(1,\ldots,phase)}$ already has the intermediate results required as input, and then it can be solved in isolation from phase 1 to phase $phase$, thus producing the intermediate results affecting the other entities from phase k to phase $phase$.

In the following we present the optimized solution technique able to reduce the total number of model execution (in pseudo-code):

OPTIMIZED ALGORITHM

STEP i): dependencies removal and measures computation

$E'^k = E^k$;

for $k = 1, \ldots, p$ do {

  While ($\exists < \emptyset; X; \Theta_X \neq \emptyset > \in E^k$) do {

    $phase = k$;

    $f = k$;

    While (($f \leq numPhases$) and ($\exists < \emptyset; X; \Theta_X > \in E^f$)) do {

      if ($\Theta_X \neq \emptyset$) then {

        $phase = f$;

        Remove $< \emptyset; X; \Theta_X >$ from E^{phase};

        Remove all the occurrences of X in each tuple of E^{phase};

      }

      $f = f + 1$;

    }

    Read from db $\bigcup_{j=1}^{phase} O^{(j)}_{Z\rightarrow X}$, with $< \Omega_Z; Z; \Theta_Z > \in E^j$ and $X \in \Theta_Z$;

Step	Order	$X^{(1,\ldots,k)}$	k	Inputs	Outputs
i)	1	$5^\circ \cup \varepsilon_{5\to1}^{(1),OUT} \cup \varepsilon_{5\to1}^{(2),OUT}$	2		$O_{5\to1}^{(1)} \cup O_{5\to1}^{(2)} \cup M_5$
i)	1	$3^\circ \cup \varepsilon_{3\to2}^{(1),OUT}$	1		$O_{3\to2}^{(1)}$
i)	2	$1^\circ \cup \varepsilon_{1\to2}^{(1),OUT} \cup \varepsilon_{1\to\{2,4\}}^{(2),OUT} \cup \varepsilon_{5\to1}^{(1),IN} \cup \varepsilon_{5\to1}^{(2),IN}$	2	$O_{5\to1}^{(1)} \cup O_{5\to1}^{(2)}$	$O_{1\to2}^{(1)} \cup O_{1\to2}^{(2)} \cup O_{1\to4}^{(2)} \cup M_1$
i)	3	$2^\circ \cup \varepsilon_{2\to3}^{(2),OUT} \cup \varepsilon_{1\to2}^{(1),IN} \cup \varepsilon_{3\to2}^{(1),IN} \cup \varepsilon_{1\to2}^{(2),IN}$	2	$O_{1\to2}^{(1)} \cup O_{3\to2}^{(1)} \cup O_{1\to2}^{(2)}$	$O_{2\to3}^{(2)} \cup M_2$
ii)	4	$3^\circ \cup \varepsilon_{3\to2}^{(1),OUT} \cup \varepsilon_{2\to3}^{(2),IN}$	2	$O_{2\to3}^{(2)}$	M_3
ii)	4	$4^\circ \cup \varepsilon_{1\to4}^{(2),IN}$	2	$O_{1\to4}^{(2)}$	M_4

Table 4.2: Optimized algorithm execution

```
      if (f > numPhases) then
         Compute M_X and O_X^(k,...,p), and write it to db;
      else
         Compute O_X^(k,...,phase), and write it to db;
   }
}
STEP ii):  remaining measures computation
E^k = E'^k;
for each entity X such that M_X has not yet been computed do {
   Read from db ⋃_{j=1}^{p} O_{Z→X}^(j), with <Ω_Z; Z; Θ_Z> ∈ E^j and X ∈ Θ_Z;
   Compute M_X;
}
```

For a better understanding, in Table 4.2 we detail the execution of the optimized algorithm for the example depicted in Figure 4.3, supposing that there are only two phases ($p = 2$). The meaning of the columns are the same as those defined for the standard algorithm.

The first two entities to be analyzed are '3' and '5' (active entities). Entity '3' becomes passive in phase 2, and then we can solve the model $3^{(1)}$ only, thus removing the dependency relation between '3' and '2' during phase 1. On the contrary, entity '5' is still active during phase 2, and then we can solve the model $5^{(1,2)}$ producing two kinds of results: the intermediate results $O_{5\to1}^{(1)}$ and $O_{5\to1}^{(2)}$, that remove the dependency relations between entity '5' and entity '1' during phase 1 and 2, and the measure of interest for entity '5' (M_5). Now the dependency relations that affect entity '1' during all the phases have been removed, and then the corresponding model $1^{(1,2)}$ can be solved in isolation producing $O_{1\to2}^{(1)}$, $O_{1\to2}^{(2)}$, $O_{1\to4}^{(2)}$ and M_1. At this point we can solve the model $2^{(1,2)}$ producing $O_{2\to3}^{(2)}$ and M_2. Finally, $3^{(1,2)}$ and $4^{(1,2)}$ can be solved in an unspecified order obtaining the corresponding measures of interest.

With respect to the application of the standard algorithm, the optimized algorithm greatly reduces the total number of required computations, and its real ef-

fectiveness mainly depends on the structure of the dependency connection graphs of each phase. In the example of Table 4.2, the optimized algorithm execution reduces the total number of computations from 11 (see Table 4.1) to 6, thus reducing the overall complexity.

Future developments

The major weakness of the proposed solution processes is that the solution of the models in each phase are computed starting from the first phase.

A very interesting research direction to remove this weakness is to compute, during the solution of the model $X^{(i,\ldots,j)}$ (with $i \leq j$), not only the intermediate results $O_X^{(i,\ldots,j)}$ (that remove the dependency relations involving entity X from phase i to phase j), but also the state of the computation at the end of phase j. For example, if the underlying stochastic process is Markovian, the state-occupancy probability vector at the end of a phase allows to completely reconstruct the past behavior of the model, and then it can be used as initial probability marking distribution for the solution of the same model in the next phases (as proposed in [66]), thus avoiding to re-compute the solution from the first phase. In case of non-Markovian models solved by simulation, the state-occupancy probability vector at the end of a phase is not sufficient to reconstruct the past behavior of the model, and then further information has to be extracted (e.g. the remaining completion time for each enabled transition).

The same problem can be seen from the following point of view. Each model $X^{(i,\ldots,j)}$ has an "implicit" dependency relation with the model $X^{(h,\ldots,i-1)}$ (with $h \leq i-1$). Using the introduced notation, it means that $X^{(h,\ldots,i-1)} \rightarrow X^{(i,\ldots,j)}$. The state of the computation at the end of phase $i-1$ can be used to remove this dependency relation just providing it as input to the model $X^{(i,\ldots,j)}$.

4.4.3 Applicability of the solution process

The solution processes detailed in Sections 4.4.1 (standard algorithm) and 4.4.2 (optimized algorithm) only work if *the dependency connection graph in each phase is acyclic*. In other words, we have assumed that in each phase there are no paths of dependency relations starting and ending in the same entity.

In fact, the two algorithms are essentially based on the identification, inside each phase, of an "independent" entity, that is an active entity or a mixed entity, provided that all the dependency relations affecting it in the current phase and in all the previous one's have already been removed. The problem immediately occurs just considering a phase k composed by two entities X and Y that interact each other through the dependency relations $X \rightarrow_k Y$ and $Y \rightarrow_k X$. In this case it is impossible to select an independent entity as they are mutually dependent.

How to proceed?

There are two main possible solutions. The first one is to develop algorithms accounting for the presence of cycles in a phase, using a fixed-point analysis ([1]). Let X and Y be two entities that interact each other during phase k through the dependency relations $X \rightarrow_k Y$ and $Y \rightarrow_k X$. The intermediate results $O^{(k)}_{X \rightarrow Y}$ and $O^{(k)}_{Y \rightarrow X}$ could be obtained as fixed points of a sequence of iterations. Let f and g be the functions computing $O^{(k)}_{X \rightarrow Y}$ and $O^{(k)}_{Y \rightarrow X}$, respectively (see Equation 4.4). Starting with an initial guess h_0, representing the hypothesized effect that entity Y has on X (hypothesized $O^{(k)}_{Y \rightarrow X}$), we could iterate in the following way:

$$
\begin{aligned}
&f(X^{(1,\ldots,k)}, h_0 \cup \textstyle\bigcup_{j=1}^{k} O^{(j)}_{Z \rightarrow X}) = h_1,\\
&g(Y^{(1,\ldots,k)}, h_1 \cup \textstyle\bigcup_{j=1}^{k} O^{(j)}_{Z \rightarrow Y}) = h_2,\\
&f(X^{(1,\ldots,k)}, h_2 \cup \textstyle\bigcup_{j=1}^{k} O^{(j)}_{Z \rightarrow X}) = h_3,\\
&g(Y^{(1,\ldots,k)}, h_3 \cup \textstyle\bigcup_{j=1}^{k} O^{(j)}_{Z \rightarrow Y}) = h_4,\\
&\ldots\\
&f(X^{(1,\ldots,k)}, h_{2i} \cup \textstyle\bigcup_{j=1}^{k} O^{(j)}_{Z \rightarrow X}) = h_{2i+1},\\
&g(Y^{(1,\ldots,k)}, h_{2i+1} \cup \textstyle\bigcup_{j=1}^{k} O^{(j)}_{Z \rightarrow Y}) = h_{2i+2},\\
&\ldots
\end{aligned}
$$

Each element of the sequence $h_0, h_2, \ldots, h_{2i}, \ldots$ represents the hypothesized effect that entity Y has on X during phase k ($O^{(k)}_{Y \rightarrow X}$) at the ith-iteration. Each element of the sequence $h_1, h_3, \ldots, h_{2i+1}, \ldots$ represents the hypothesized effect that entity X has on Y during phase k ($O^{(k)}_{X \rightarrow Y}$) at the ith-iteration. The iteration is terminated when the difference between h_{2i} and h_{2i+2} and the difference between h_{2i+1} and h_{2i+3} are below a certain tolerance level, and in this case $h_{2i} = O^{(k)}_{X \rightarrow Y}$ and $h_{2i+1} = O^{(k)}_{Y \rightarrow X}$.

Note that this iteration may not always converge and, if it converges, it may not always converge to the same value. Actually the main problem that arises in dealing with models which use fixed point iteration is the theoretical proof of existence, uniqueness, and convergence of the fixed point equations, which still remains an "art" and depends on the formalism adopted to implement the models (e.g. [67] establishes conditions for the existence of a solution in stochastic Petri nets).

The second possible solution able to deal with a cyclic dependency connection graph in a phase is to "hide" the cycles inside an appropriate larger entity. The idea is to consider the set of the cyclicly interacting entities as a single larger entity, properly modifying the interested dependency relations.

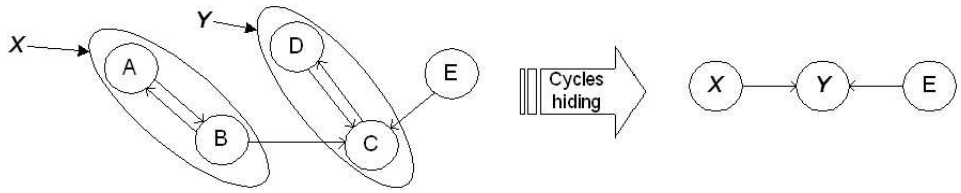

Figure 4.10: Cycles hiding

In Figure 4.10 we show the application of this simple procedure to a dependency connection graph of phase k defined by the following dependency relations: $A \rightarrow_k B, B \rightarrow_k A, C \rightarrow_k D, D \rightarrow_k C, B \rightarrow_k C$ and $E \rightarrow_k C$. The cycles disappear considering the system during phase k as the composition of three entities only, X, Y and E, interacting through the following dependency relations: $X \rightarrow_k Y$, and $E \rightarrow_k Y$. The resulting dependency connection graph during phase k is now acyclic and then the separate solution processes described in the previous Sections can be successfully applied. The entity aggregation disappears as soon as we consider another phase.

This is a very useful and elegant solution that enables us to treat all the existing phased-interacting system. The main disadvantage is that it increases the complexity of those entities built as aggregation of the simple entities (e.g. entity X and Y of Figure 4.10), thus partially disrupting the advantages of the system decomposition. In the worst case in which all the entities are cyclicly interacting in all the phases, this procedure is unapplicable as it would lead to an equivalent system composed by one entity only with a non manageable complexity.

4.4.4 About effectiveness

The major characteristic of the solution process is its capability to manage the complexity (both spatial and temporal) of the overall model, as we provide the solutions solving some separated sub-models.

In case of state-based analytical solution, the state-space explosion problem is drastically reduced thanks to the lower number of states generated for each individual sub-model (one for each entity). Moreover, both for analytical solution and simulation, other advantages are related to:

- the mitigation of the stiffness-problem, if each submodel has less time scales than the monolithic model;
- the capability to simultaneously execute some computations in `STEP i)` and `STEP ii)` of the two algorithms, thus reducing the total computational time needed to complete the measures evaluations.

Although these aspects are strictly related to the real system under analysis, in the following we try to give some useful details on these issues.

Spatial and Temporal complexity mitigation

Suppose that a system consists of N entities $(X_1, \ldots, X_N)$ and p phases, and that all the models can be solved using a state-based analytical solution. Suppose for simplicity that S is the size of state space generated for the separate solution of each entity's model. In general, the size of the state-space generated for the solution of the unique non-decomposed model can be expressed as S^r, where r is a parameter whose value depends on the actual implementation of the models (typically $1 \leq r \leq N$). In the worst case $r = N$, and it happens when the resulting state space is the cartesian product of the state spaces associated to each entity's model. In this case the application of the decomposition approach induces the maximum mitigation of the spatial complexity, since it is reduced from S^N to S.

Now we give an estimation of the time complexity of the standard algorithm described in Section 4.4.1, and we compare it with that obtained by solving the non-decomposed model of Figure 4.4 as a whole. We refer to the standard algorithm as we suppose to be in a worst case scenario in which the application of the optimized algorithm does not produce any significant advantage, although it is not always the case. We provide the time complexity estimation supposing that i) all the models are CTMC, and ii) the Standard Uniformization (SU) algorithm [68] is adopted to compute a transient-state probability vector. The SU is based on transforming a CTMC into a DTMC with uniform exponential inter-event times, and its temporal complexity is $O(3N_T M^2)$, where M is the number of states of the CTMC and N_T is the truncation point of the infinite sum computing the state-occupancy probability vector.

In the worst case, the application of the standard algorithm requires to solve each entity p times (one model solution in each phase), and then Np is the total number of model solutions. Therefore, the total time complexity of the standard algorithm is about $O(3NpN_T S^2)$. Instead, the computational cost demanded to numerically solve the system as a whole model is about $O(3N_T(S^r)^2)$, as the state-space size of the whole model is about S^r (for an appropriate value of r). Therefore, the application of the standard algorithm is useful only if

$$O(3NpN_T S^2) < O(3N_T S^{2r}) \quad ,$$

and it happens when $Np < S^{2r-2}$ (usually true if $r \geq 2$). Note that we are implicitly assuming that all the models have the same N_T value.

In addition, we want to emphasize that the total time needed to complete the algorithm execution could be greatly reduced considering that the computations

having equal order (the second column of Table 4.1, from the left) could be simultaneously computed on different computers, as they have no mutual interactions. In the example of Table 4.1, the two computations with order equal to 1 could be simultaneously executed on two different computers, while the five computations with order equal to 6 could be simultaneously executed on five different computers, thus approximately halving the required total computational time. The amount of the reduction strictly depends on the dependency connection graphs defined in each phase.

Therefore, as expected, the complexity of the algorithm is always the result of a trade-off between largeness of the single models and total number of computations, and this aspect may heavily influence the definition of the elements composing the phased-interacting system.

Stiffness mitigation

The stiffness problem arises when the range of time scales in a model is high. This problem can be mitigated applying the splitting procedure described in Section 4.3.2, when the unique model $\varepsilon^{(k)}_{X\to\Theta_X=\{Y_1,\ldots,Y_n\}}$ is replaced with the two submodels $\varepsilon^{(k),OUT}_{X\to\Theta_X}$ and $\varepsilon^{(k),IN}_{X\to\Theta_X}$. Let Δ be a function that takes as input a model and produces as output the range of the time scales of the same model. Supposing that the ranges of time scales for the two models $\varepsilon^{(k),OUT}_{X\to\Theta_X}$ and $\varepsilon^{(k),IN}_{X\to\Theta_X}$ are equal to the range of time scale for the unique model $\varepsilon^{(k)}_{X\to\Theta_X}$, then the following relations hold:

$$\Delta(X^\circ \cup \varepsilon^{(k),OUT}_{X\to\Theta_X}) \leq \Delta(X^\circ \cup \varepsilon^{(k)}_{X\to\Theta_X} \cup Y_1^\circ \cup \ldots \cup Y_n^\circ) \quad \text{, and} \tag{4.7}$$

$$\Delta(\varepsilon^{(k),IN}_{X\to\Theta_X} \cup Y_1^\circ \cup \ldots \cup Y_n^\circ) \leq \Delta(X^\circ \cup \varepsilon^{(k)}_{X\to\Theta_X} \cup Y_1^\circ \cup \ldots \cup Y_n^\circ) \quad . \tag{4.8}$$

Therefore, as a side effect, the splitting procedure can reduce the range of time scales of the analyzed model and then can mitigate the stiffness problem. This consideration could be extremely useful when the entities composing the system perform critical functions characterized by very different range of time scales, for example due to very different architectural structure (software and/or hardware).

4.4.5 About Accuracy

One of the main problem that we have to cope with when applying a model decomposition is the possibility to introduce errors that can affect the accuracy of the final measures.

Suppose that M_X is the measure of interest obtained solving the decomposed-model of Figure 4.8, using one of the algorithms previously described, and let

M'_X be the same measure obtained solving the original non-decomposed model of Figure 4.4. We say that M_X is accurate if ***Diff***$(M_X, M'_X) < \epsilon$, where ***Diff*** is a function that measures the difference of the two results (when both available), and ϵ is a pre-defined tolerance level.

The critical point is the application of the splitting procedure described in Section 4.3.2, during which the model representing each dependency relation in a phase is decomposed in two submodels (the IN part and the OUT part), interacting through the passing of a proper intermediate result. If the decomposed model is not able to perfectly reconstruct the behavior of the original non-decomposed model, then we have introduced an error. This happens, for example, when the decomposed model is meant to approximate the original model in order to decrease its complexity. The solution of an approximated decomposed model produces an approximated intermediate result that, in turn, will affect the accuracy of the final measure.

In more detail, with reference to Equation 4.6, the accuracy of a measure of interest M_X depends:

1. on the accuracy of the intermediate results $\bigcup_{j=1}^{p} O^{(j)}_{Z \to X}$, with $< \Omega_Z; Z; \Theta_Z > \in E^j$ and $X \in \Theta_Z$, where p is the total number of phases. An intermediate result is non-accurate when it does not contain adequate information that enable the "Dependency Relations - IN" and "Dependency Relations - OUT" models to perfectly reconstruct the behavior of the entity. It typically happens when the precision of the intermediate result is low, for example due to a rough approximation error that occurs during the computation.

   With reference to Equation 4.4, the computation of each partial result $O^{(k)}_{X \to \Theta_X}$ uses as input all the previously produced intermediate results affecting the entity (that is $\bigcup_{j=1}^{k} O^{(j)}_{Z \to X}$). Therefore, an error in the accuracy of $O^{(j)}_{Z \to X}$, with $j \leq k$, can affect the accuracy of all the successive intermediate results that are produced using this value as input, and finally can affect the measure of interest.

   The algorithm for calculating the number of time an intermediate result (both accurate or not) propagates through the phases corresponds to the algorithm used to compute the maximum length path in the dependency connection graph corresponding to the entire lifetime of the system.

2. on the capability of the "Dependency Relations - IN" and "Dependency Relations - OUT" submodels of Figure 4.8 to perfectly represent the interactions that involve entity X, using the previously produced intermediate results. When this capability is low, we say that the models are non-accurate.

If the submodels $\varepsilon^{(k),IN}_{X\to\Theta_X}$ and $\varepsilon^{(k),OUT}_{X\to\Theta_X}$ (the IN and OUT "Dependency Relations" models for entity X during phase k) only approximate the real behavior of the existing dependency relation $X \to_k \Theta_X$, then the intermediate result $O^{(k)}_{X\to\Theta_X}$ will be non-accurate, as produced by a non-accurate model.

This may happen, for example, when we approximate a cumulative distribution function of a random variable, representing an intermediate result passed between two entities, with its expectation. In this case the error is related to the incapability of the IN and OUT models to perfectly rebuild the real behavior of the interactions, independently from the accuracy of the expectation.

A more comprehensive analysis dealing with the errors induced by a model decomposition is reported in [16]. Using the connection formalism as a decomposition technique, the author defines the concept of accuracy for the solution of measures, for the final results and for the models, and provides a useful insight on the relationships between model decomposition and introduced approximations.

4.5 Available tools supporting the modeling and solution framework

The modeling and solution framework proposed in this Chapter demands the modeler to manually perform the following actions: i) construct the required models, ii) pass the intermediate results between the models, and iii) solve the models with an appropriate solution technique. Actually the feasibility of the methodology is also related to the capability to automatize these manual procedures, thus reducing the required modeler's effort. In this Section we outline the main characteristics that an automated modeling and solution tool should have in order to facilitate the application of the proposed methodology. A candidate tool should have the following main characteristics:

1. the capability to support different formalisms and different solution methods. Actually each model $X^{(1,\ldots,p)}$, that represents the real behavior of entity X during all the phases (see Equation 4.3), could be implemented using a different formalism, interacting with the other models through the passing of the intermediate results only, and could be solved using an appropriate solution technique.

2. the possibility to use solutions calculated from one solvable model as inputs into other solvable models (model connection), that is at the base of the decomposition technique described in Section 4.3.2.

3. the capability to automatize the splitting procedure as depicted in Figure 4.7. Starting from the "Dependency Relations model", the tool should automatically build the corresponding "Dependency Relations IN and OUT models", perhaps using some extra information (e.g. to define how the model can be decomposed with respect to the expected intermediate result that has to be produced, like a mean or a distribution).

4. the opportunity to simultaneously perform some computations on different machines. This is not a fundamental characteristic, but it would be useful in the case in which the models to be solved are mutually independent, that is they do not interact each other, and then can be solved concurrently and in isolation (see Section 4.4.4).

From these considerations it appears quite evident that we need a modeling framework that can accommodate multiple modeling formalisms, multiple ways to combine models expressed in different formalisms, and multiple model solution methods, and then we refer to the multi-formalism/multi-solution tools outlined in Section 1.7.2. Most of the these tools are able to support the proposed modeling and solution framework, as they have at least the first two characteristics (strongly requested). At the same time, there is not any tool that can accommodate the automatization of the splitting procedure (characteristic number 3) that, consequently, has to be manually performed. Since no existing tool perfectly suits our requirements, we do not find any significant reason to adopt a modeling tool different from that used during all this dissertation, and then we will still use Möbius to present the feasibility case-study in the next Chapter. In addition, Möbius includes some features that really facilitate the automatic passing of intermediate results between models (connection formalism [62]).

4.6 Summary

In this Chapter we proposed a very natural decomposition/aggregation approach based on a functional, temporal and physical decomposition. A system is seen as a set of interacting sub-systems that can work in isolation or can interact each other through some dependency relations that may change during the system's lifetime. Following a modular approach, we first modeled the system as a whole, thus identifying the main submodels with respect to the functions they perform. Then the whole model has been decomposed (through the splitting procedure) in a set of submodels to be solved in isolation, passing some intermediate results when required. We proposed two algorithms (a standard version and an optimized one) defining the order in which the submodels have to be solved, on the basis

of the dependency relation graph for each phase. Finally we provided some considerations about the effectiveness and accuracy of the proposed methodology, although these properties are strictly related to the real system under analysis. The proposed methodology, although to be further refined, shows very interesting potentiality both concerning the modeling, that facilitates modularity and scalability, and the solution process, that could be further improved considering the future developments sketched in Section 4.4.2, thus avoiding to re-compute the solution from the first phase during the dependency removal process.

Chapter 5

Feasibility case-study

In this Chapter we describe the application of the modeling and solution framework presented in Chapter 4 to the GPRS case-study detailed in Chapter 2. Here the focus is not on the QoS evaluation, but on the feasibility of the proposed decomposition approach, emphasizing the advantages and the disadvantages both with respect to the application of the particular solution technique presented in Section 2.4, fully developed for the solution of the considered class of systems, and to the application of the standard solution technique that solves the non-decomposed model as a whole. We follow the same structure of Chapter 4: first we build the whole non-decomposed model, and then we apply the splitting procedure thus obtaining the decomposed one.

The rest of this Chapter is organized as follows. Section 5.1 presents the GPRS infrastructure and its corresponding phased-interacting system. The procedure to construct the single model representing the whole GPRS infrastructure is depicted in Section 5.2, while the application of the decomposed solution process is described from Section 5.3 to Section 5.7. In Section 5.8 we introduce the connection formalism that can be used to automatize the passing of the intermediate results between models, while the numerical results of the simulation studies are presented, compared and discussed in Section 5.9. Conclusions are finally drawn in Section 5.10.

5.1 The GPRS infrastructure as a phased-interacting system

The analyzed scenario is the same as that presented in Section 2.5.1, and it consists of one central cell (CELL-0) and three partially overlapping cells (CELL-1, CELL-2 and CELL-3). We aim to analyze the behavior of the GPRS network during the following temporal events (see Figure 5.1):

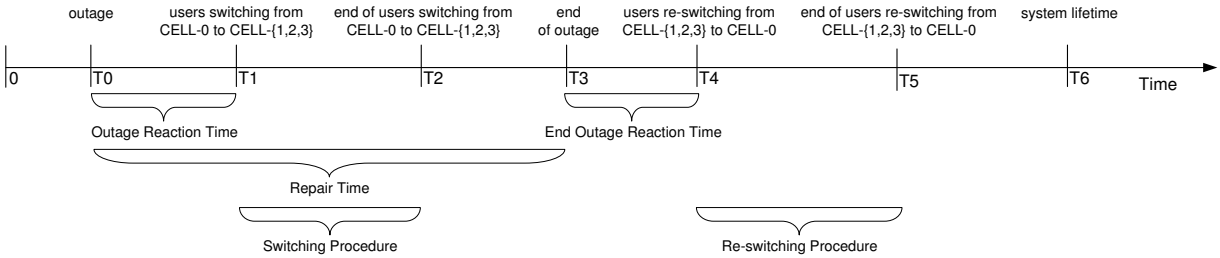

Figure 5.1: Scheduled Temporal Events

- At time T0, an outage occurs in the central cell (CELL-0), thus determining congestion some time after;
- At time T1, the switching procedure starts, causing some users to be switched from the congested cell to its adjacent ones;
- At time T2 the switching procedure ends because of i) the established number of users has been switched to all the neighbors cells, or ii) the re-switching procedure starts (T2=T4);
- At time T3, the outage ends;
- At time T4, a Resource Management System (RMS) reacts to the end of the outage and starts the users re-switching procedure from CELL-1, CELL-2 and CELL-3 to CELL-0;
- At time T5, the re-switching procedure ends, and it happens when all the users previously switched to CELL-1, CELL-2 and CELL-3 have been re-switched to the central cell;
- Time T6 identifies the end of the lifetime of the system, that corresponds to the interval of time [0;T6].

This set of temporal events is the same as that presented in Chapter 2: here we have only explicitly identified the time at which the switching procedure ends because of the established number of users has been switched to all the neighbors cells (time T2). The measures of interest are the point-wise congestion function (**PCf**) and the total congestion indicator (**TCi**), as detailed in Section 2.3.

As shown in Figure 5.2, the system under analysis can be seen as a phased-interacting system in which:

- each entity corresponds to a GPRS cell;

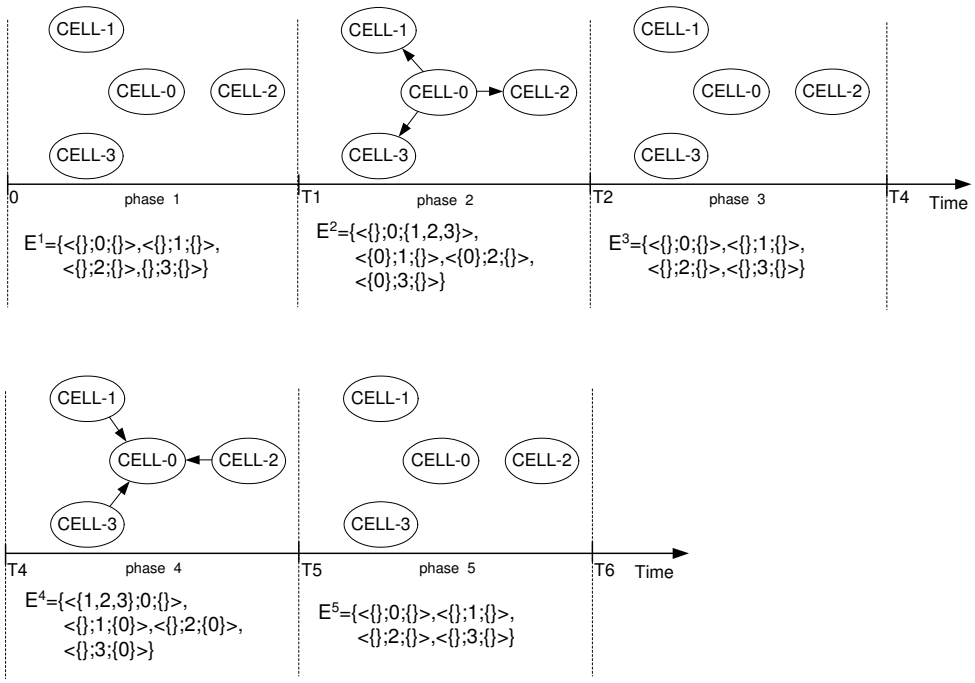

Figure 5.2: The GPRS network behavior as a phased-interacting system

- an entity X is connected (there exists a dependency relation) to an entity Y during phase p ($X \rightarrow_p Y$) if, during phase p, X is a sending cell and Y is a receiving cell.

Therefore an entity CELL-X affects an entity CELL-Y through a dependency relation when some users move from CELL-X to CELL-Y due to a cell resizing. The dependency relation is only in one direction (from CELL-X to CELL-Y) because we are supposing that a receiving cell can not refuse an incoming user, and then the switching (or re-switching) procedure only depends on the behavior of the cells that act as a sender during the phase. The lifetime of the system consists of 5 phases:

Phase 1- It is defined in the interval of time [0;T1], that is from the beginning of the system lifetime until the switching procedure starts. During this phase all the cells are neutral, that is they work in isolation. We note that the 'outage' event for CELL-0 at time T0 (with T0<T1) does not define a new phase as it only affects the internal behavior of the central cell.

Phase 2- It is defined in the interval of time [T1;T2], that is from the time in which the switching procedure starts until it ends. We assume that the switching procedure lasts until i) the established number of users has been switched to

all the neighbor cells, or ii) the re-switching procedure starts (T2=T4). This means that CELL-0 is considered to be active if it sends the users to *at least* one adjacent cell, and CELL-1,...,CELL-N are passive if *at least* one of them receives some users. In the case in which the single switching procedure between CELL-0 and CELL-i (with $i \in \{1, 2, 3\}$) completes before time T2 (e.g. at a time T2$_x$ <T2), then during the period [T2$_x$;T2] no users will be actually moved from CELL-0 to CELL-i, although we still consider them to be active and passive, respectively.

Phase 3- It is defined in the interval of time [T2;T4], that is from the time in which the switching procedure ends until the re-switching procedure starts. During this phase all the cells are neutral, that is they work in isolation. We note that the 'end of outage' event for CELL-0 at time T3 does not define a new phase because it only affects the internal behavior of the central cell.

Phase 4- It is defined in the interval of time [T4;T5], that is from the time in which the re-switching procedure starts until it ends. We assume that the re-switching procedure lasts until *all* the users previously switched to CELL-1, CELL-2 and CELL-3 have been re-switched to the central cell. This means that CELL-1, CELL-2 and CELL-3 are considered to be active if *at least* one of them sends the users to the central cell, and CELL-0 is passive if it receives some users from *at least* one neighbor cell. In the case in which the single re-switching procedure between CELL-i (with $i \in \{1, 2, 3\}$) and CELL-0 completes before time T5 (e.g. at a time T5$_x$ <T5), then during the period [T5$_x$;T5] no users will be actually moved from CELL-i to CELL-0, although we still consider them to be active and passive, respectively.

Phase 5- It is defined in the interval of time [T5;T6], that is from the time in which the re-switching procedure ends until the end of the system lifetime (possibly T6=∞ for the steady-state behavior). During this phase all the cells are neutral, that is they work in isolation.

5.2 The whole non-decomposed model

In this Section we describe the main submodels composing the whole model for the considered system, as described in Section 4.3.1. We need to implement the following models:

1. "Phases model", that identifies the current phase of the system;
2. $CELL - X^{\circ}$ model, representing the neutral behavior of a generic GPRS cell CELL-X (with $X = 0, 1, 2, 3$) during all the system lifetime;

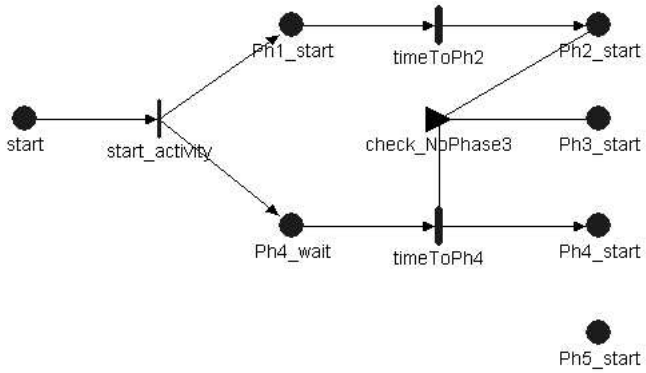

Figure 5.3: The "phases model"

3. $\varepsilon^{(2)}_{CELL-0 \to \{CELL-1, CELL-2, CELL-3\}}$ model, representing the dependency relation $CELL-0 \to_2 \{CELL-1, CELL-2, CELL-3\}$, that is defining how entity CELL-0 affects the entities CELL-1, CELL-2 and CELL-3 during phase 2 (switching procedure);

4. $\varepsilon^{(4)}_{CELL-X \to \{CELL-0\}}$ model (with $X = 1, 2, 3$), representing the dependency relation $CELL-X \to_4 \{CELL-0\}$, that is defining how entity CELL-X affects the entity CELL-0 during phase 4 (re-switching procedure);

For the sake of clarity in the models definitions, we suppose that the switching and re-switching procedure can affect the *idle* users only, that is the users that are not requiring any service to the network. This assumption has been introduced only to provide a more clear understanding of the presented models, but the same models could be easily extended to deal with the switching (and re-switching) of active users, and also with more than three adjacent overlapping cells.

5.2.1 "Phases model"

In Figure 5.3 we show the model representing the phases of the system. The marking of place `start` is set to 1, while the other places are set to 0. All the timed activities are deterministic. The `timeToPh2` activity fires after a time *T1*, thus determining the beginning of the switching procedure (one token in place `Ph2_start`). When the switching procedure ends, the token is removed from place `Ph2_start` and added to place `Ph3_start` (this is done by another model that shares these places), thus identifying the beginning of phase 3. In the meantime, after a time *T4*, the activity `timeToPh4` fires and the re-switching procedure starts (one token in place `Ph4_start`). The output gate `check_NoPhase3` removes the token from place `Ph2_start` in the case in which the switching

procedure has not yet completed: in this case it is interrupted and than the re-switching procedure can take place (in this case there is no phase 3). When the re-switching procedure ends the token is removed from place `Ph4_start` and added to place `Ph5_start` (this is done by another model that shares these places).

5.2.2 $CELL - X^{\circ}$ model

An abstract view of the GPRS cell model for a generic CELL-X (with $X = 0, 1, 2, 3$) working in isolation is presented in Figure 5.4. It can be used both in the case in which CELL-X is a sending cell (CELL-0) and when it is one of the receiving cells (CELL-1, CELL-2, CELL-3).

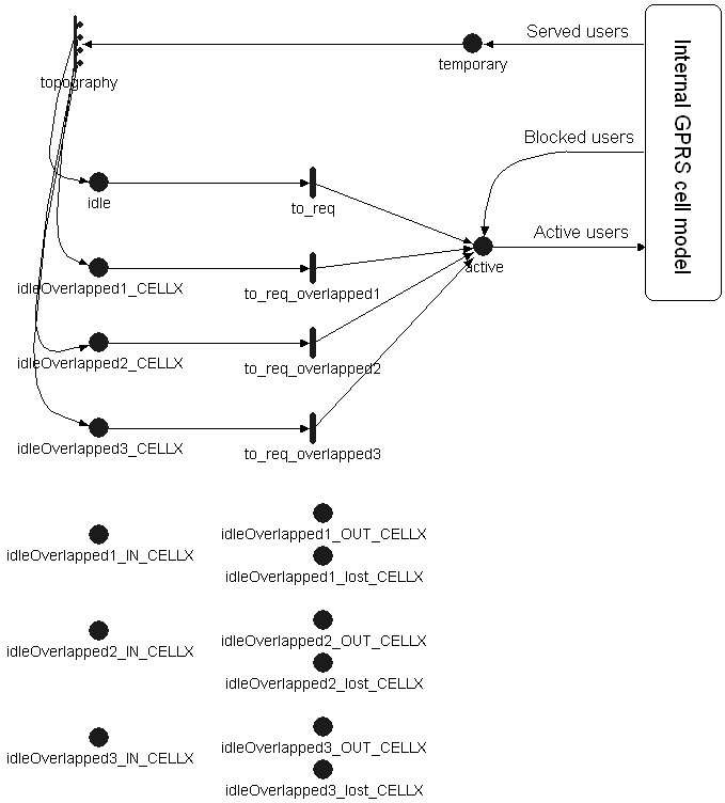

Figure 5.4: CELL-X: a neutral GPRS cell with three overlapping areas

The generic model of Figure 5.4 works as follows. When a user has been served, a token exits from the "internal GPRS cell model". The "internal GPRS cell model" has been deeply described in Section 2.4.2, and then it is not reported here for the sake of brevity. The generic user has to be mapped (using the `topography` activity) in the non overlapping area (place `idle`), in the area

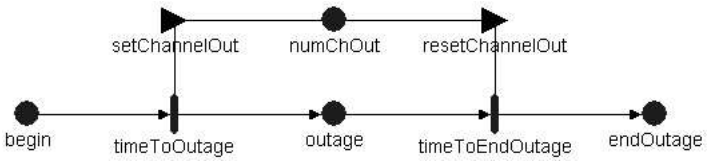

Figure 5.5: The outage model

partially overlapped with C1 (place `idleOverlapped1_CELLX`), in the area partially overlapped with C2 (place `idleOverlapped2_CELLX`), or in the area partially overlapped with C3 (place `idleOverlapped3_CELLX`) in accordance with the topography of the network. The probability that a generic user is mapped in the overlapping areas is dynamically calculated considering the original number of users camped in the non overlapping zone and in the three overlapping areas, the users that have been switched (or lost) to the other cells (if CELL-X is a sending cell) and the incoming users received from the adjacent cells (if CELL-X is a receiving cell). When an idle user requests a new service, he/she becomes active and enters in the "internal GPRS cell model" that executes the random access procedure of a GPRS cell. With respect to Figure 2.5, we note that we have explicitly identified the different overlapping areas between CELL-X and its adjacent cells.

Places `idleOverlapped`*Y*`_OUT_CELLX` and `idleOverlapped`*Y*`_lost_CELLX` represent the users belonging to CELL-X that, respectively, have been successfully switched and lost during the switching procedure to C*Y*, with *Y*=1,2,3. Finally, places `idleOverlapped1_IN_CELLX`, `idleOverlapped2_IN_CELLX` and `idleOverlapped3_IN_CELLX` represent, respectively, the users that have been re-switched from C1, C2 and C3 to CELL-X. When the cell is neutral, that is it works in isolation, the marking of these places remains fixed, while varies during the switching and re-switching procedure.

"Outage model" for CELL-0

In Figure 5.5 we show the model to be added (joined) to CELL-0° in order to account for the outage period of the central cell. Place `begin` contains one token and all the timed activities are deterministic. When `timeToOutage` fires (after a time *T0*), the outage begins and the output gate `setChannelOut` sets a number of tokens in place `numChOut` that corresponds to the number of unavailable traffic channels. When the `timeToEndOutage` activity fires (after a time *T3-T0*, that is the outage duration), the output gate `resetChannelOut` sets the number of tokens in place `numChOut` to 0 and then the outage ends.

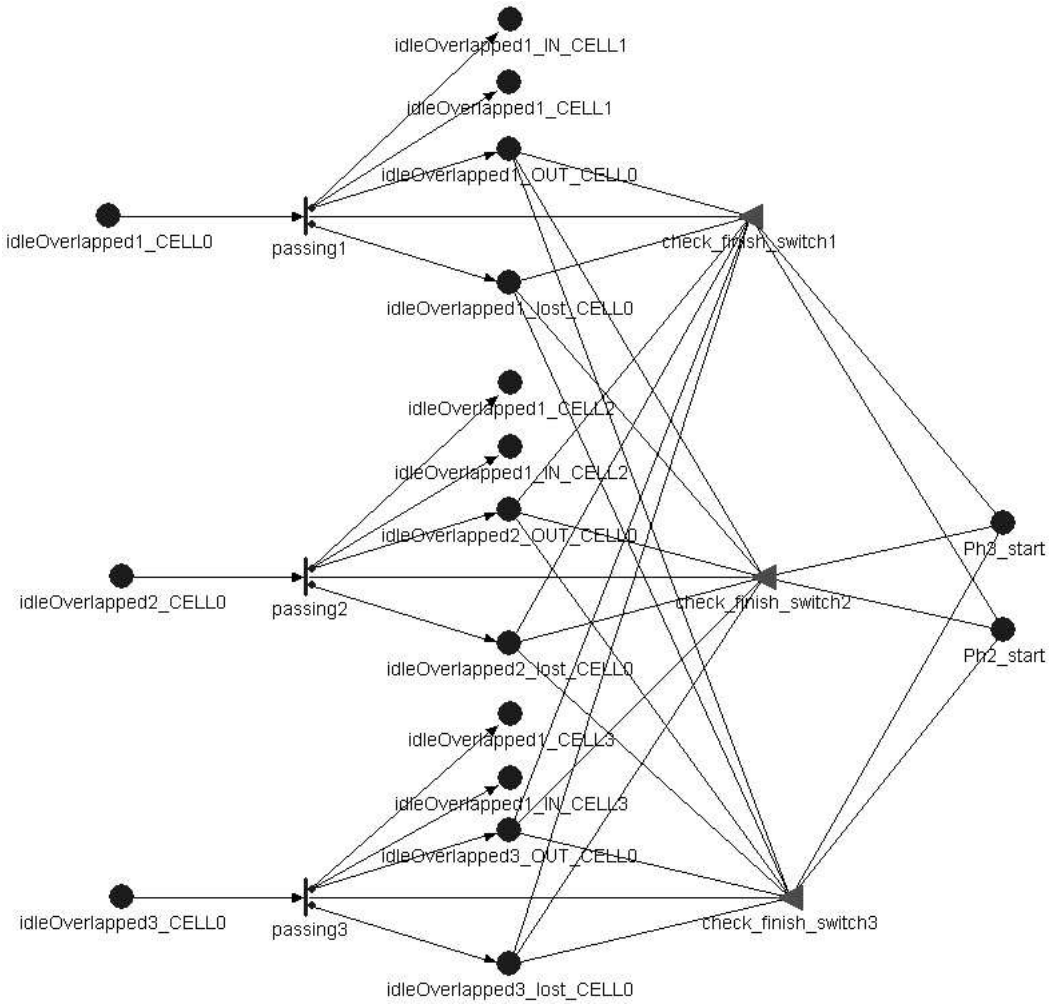

Figure 5.6: The $\varepsilon^{(2)}_{CELL-0\rightarrow\{CELL-1,CELL-2,CELL-3\}}$ model

5.2.3 $\varepsilon^{(2)}_{CELL-0\rightarrow\{CELL-1,CELL-2,CELL-3\}}$ model

The template of the model is shown in Figure 5.6 and it is used during the switching procedure (phase 2) to specify the behavior of CELL-0 to be active (sending cell), and the behavior of the adjacent cells CELL-1, CELL-2 and CELL-3 to be passive (receiving cells). Therefore, it will be joined with the models $CELL-0^{\circ}$, $CELL-1^{\circ}$, $CELL-2^{\circ}$ and $CELL-3^{\circ}$.

With reference to the upper part of Figure 5.6 only (the other parts behave similarly), the model works as follows. When the switching procedure starts (`Ph2_start` contains one token) the available idle users in the overlapping areas

between CELL-0 and CELL-1 (tokens in place `idleOverlapped1_CELL0`) are instantaneously switched. During the switching procedure, some of them are lost (tokens in place `idleOverlapped1_lost_CELL0`) and some other are successfully switched. The switched users (tokens) are added to three different places:

- places `idleOverlapped1_IN_CELL1` and `idleOverlapped1_OUT_CELL0`, whose markings represent the number of idle users that have been switched from CELL-0 to CELL-1 during the switching procedure;
- place `idleOverlapped1_CELL1`, whose marking represents the number of switched users that are actually in idle mode.

The input gate `check_finish_switch1` enables the instantaneous activity `passing1` if the switching procedure is started and there are other users to switch. When the established number of idle users has been switched from CELL-0 to CELL-1, CELL-2 and CELL-3, then the marking of place `Ph2_start` is set to zero and one token is added to place `Ph3_start` (phase 2 ends and phase 3 begins).

5.2.4 $\varepsilon^{(4)}_{CELL-1\rightarrow\{CELL-0\}}$ model

The template of the model is shown in Figure 5.7, and it is used during the re-switching procedure (phase 4) to specify how the neutral cell CELL-1 modifies to become active (sending cell), and how the neutral cell CELL-0 modifies to become passive (receiving cell). It will be joined with the models $CELL-0^{\circ}$, $CELL-1^{\circ}$, $CELL-2^{\circ}$ and $CELL-3^{\circ}$.

The model works as follows. The input gate `check_reswitch` enables the connected instantaneous activities when place `Ph4_start` contains one token (the re-switching procedure is started). When it happens, all the users previously lost (`idleOverlapped1_lost_CELL0`) are instantaneously re-camped to the central cell, and all the previously switched users are re-switched as soon as they become available. A token in place `Ph4_end_CELL`*Y*, with *Y*=1,2,3 , means that all the idle users previously switched from CELL-0 to CELL-*Y* have been re-switched to the central cell. When all the three places contain a token, the marking of place `Ph4_start` is set to zero and one token is added to place `Ph5_start` (phase 4 ends and the last phase begins).

The same model can be trivially modified to obtain the model $\varepsilon^{(4)}_{CELL-2\rightarrow\{CELL-0\}}$ that accounts for the dependency relation between CELL-2 and CELL-0 during

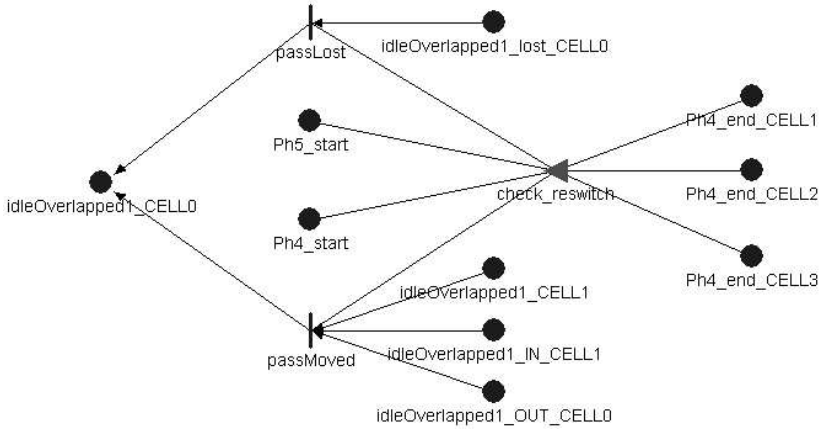

Figure 5.7: The $\varepsilon^{(4)}_{CELL-1\rightarrow\{CELL-0\}}$ model

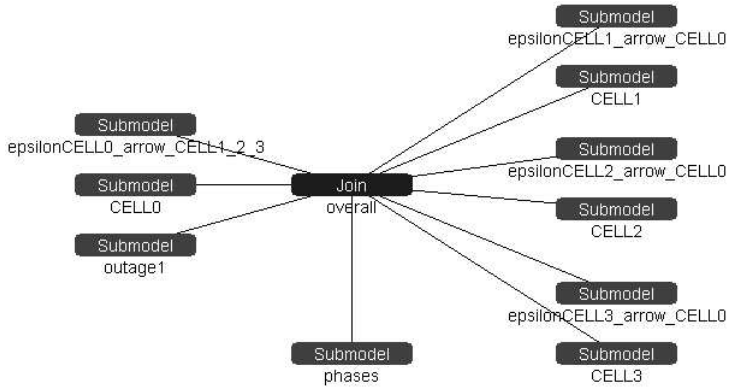

Figure 5.8: The whole non-decomposed model for CELL-0, CELL-1, CELL-2 and CELL-3

the re-switching procedure: it corresponds to the same model in which the suffix "_CELL1" is replaced with "_CELL2". The same considerations can be done to build the $\varepsilon^{(4)}_{CELL-3\rightarrow\{CELL-0\}}$ model.

5.2.5 The overall model

The whole model representing the behavior of the considered GPRS infrastructure is shown in Figure 5.8. CELL0, CELL1, CELL2 and CELL3 are the names of the models $CELL-0°$, $CELL-1°$, $CELL-2°$ and $CELL-3°$, respectively. outage1 is the name of the "outage model", and phases is the name of the "phases model". epsilonCELL0_arrow_CELL1_2_3 is the $\varepsilon^{(2)}_{CELL-0\rightarrow\{CELL-1,CELL-2,CELL-3\}}$ model, while

epsilonCELL1_arrow_CELL0, epsilonCELL2_arrow_CELL0 and epsilonCELL3_arrow_CELL0 are, respectively, the names of the models $\varepsilon^{(4)}_{CELL-1\rightarrow\{CELL-0\}}$, $\varepsilon^{(4)}_{CELL-2\rightarrow\{CELL-0\}}$ and $\varepsilon^{(4)}_{CELL-3\rightarrow\{CELL-0\}}$. These models are composed together using the Join operator ([4]), sharing:

- all the places having the same name and suffix (*name_of_the_place*_CELL0, *name_of_the_place*_CELL1, *name_of_the_place*_CELL2 and *name_of_the_place*_CELL3);
- place numChOut among CELL0 and outage1;
- places Ph2_start and Ph3_start among epsilonCELL0_arrow_CELL1_2_3 and phases.
- places Ph4_start and Ph5_start among epsilonCELL1_arrow_CELL0, epsilonCELL2_arrow_CELL0, epsilonCELL3_arrow_CELL0 and phases.

5.3 The decomposed model

As described in Section 4.3.2, the splitting procedure is applied to the whole non-decomposed model producing a set of submodels that can be solved in isolation. Most of the models that we built for the non-decomposed case can be reused in the decomposed one. In particular, the "phases model", the "outage model" and the $CELL-0°$, $CELL-1°$ and $CELL-2°$ models can be used as they are. The models developed for the specifically tailored methodology of Chapter 2 can not be easily reused in this context, because they have been constructed in a non-modular way: for example, the model of the phases is fully embedded in the "users switching/reswitching sub-model".

The splitting procedure only affects the models representing the dependency relations between the entities (cells), and then we decompose

1. the $\varepsilon^{(2)}_{CELL-0\rightarrow\{CELL-1,CELL-2,CELL-3\}}$ model in two submodels: the OUT part ($\varepsilon^{(2),OUT}_{CELL-0\rightarrow\{CELL-1,CELL-2,CELL-3\}}$ model) and the IN part ($\varepsilon^{(k),IN}_{CELL-0\rightarrow\{CELL-1,CELL-2,CELL-3\}}$ model);
2. the $\varepsilon^{(4)}_{CELL-1\rightarrow\{CELL-0\}}$ model in two submodels: the OUT part ($\varepsilon^{(4),OUT}_{CELL-1\rightarrow\{CELL-0\}}$ model) and the IN part ($\varepsilon^{(4),IN}_{CELL-1\rightarrow\{CELL-0\}}$ model).
3. the $\varepsilon^{(4)}_{CELL-2\rightarrow\{CELL-0\}}$ model in two submodels: the OUT part ($\varepsilon^{(4),OUT}_{CELL-2\rightarrow\{CELL-0\}}$ model) and the IN part ($\varepsilon^{(4),IN}_{CELL-2\rightarrow\{CELL-0\}}$ model).

Row	Step	Order	$X^{(1,\ldots,k)}$	k	Inputs	Outputs
1	i)	1	$0^{\circ} \cup \varepsilon^{(2),OUT}_{0\to\{1,2,3\}}$	2		$O^{(2)}_{0\to\{1,2,3\}}$
2	i)	2	$1^{\circ} \cup \varepsilon^{(4),OUT}_{1\to\{0\}} \cup \varepsilon^{(2),IN}_{0\to\{1\}}$	4	$O^{(2)}_{0\to\{1\}}$	$O^{(4)}_{1\to\{0\}} \cup M_1$
3	i)	2	$2^{\circ} \cup \varepsilon^{(4),OUT}_{2\to\{0\}} \cup \varepsilon^{(2),IN}_{0\to\{2\}}$	4	$O^{(2)}_{0\to\{2\}}$	$O^{(4)}_{2\to\{0\}} \cup M_2$
4	i)	2	$3^{\circ} \cup \varepsilon^{(4),OUT}_{3\to\{0\}} \cup \varepsilon^{(2),IN}_{0\to\{3\}}$	4	$O^{(2)}_{0\to\{3\}}$	$O^{(4)}_{3\to\{0\}} \cup M_3$
5	ii)	3	$0^{\circ} \cup \varepsilon^{(2),OUT}_{0\to\{1,2,3\}} \cup \varepsilon^{(4),IN}_{1\to\{0\}} \cup \varepsilon^{(4),IN}_{2\to\{0\}} \cup \varepsilon^{(4),IN}_{3\to\{0\}}$	4	$O^{(4)}_{1\to\{0\}} \cup O^{(4)}_{2\to\{0\}} \cup O^{(4)}_{3\to\{0\}}$	M_0

Table 5.1: Optimized algorithm execution for the case study

4. the $\varepsilon^{(4)}_{CELL-3\to\{CELL-0\}}$ model in two submodels: the OUT part ($\varepsilon^{(4),OUT}_{CELL-3\to\{CELL-0\}}$ model) and the IN part ($\varepsilon^{(4),IN}_{CELL-3\to\{CELL-0\}}$ model).

Since each model to be solved is composed by only one cell at a time, there are no ambiguities and then we remove the suffix "_CELLX" (with X=0,1,2,3) from the name of all the places.

5.4 Applying the optimized solution algorithm

In Table 5.1 we detail the execution of the optimized algorithm for the example depicted in Figure 5.2. In the first column from the left we identify the rows of the Table, and the meaning of the other columns are the same as those defined for the standard algorithm in Section 4.4. In order to have a compact Table, we denote an entity CELL-i with "i", for $i = 0, 1, 2, 3$.

Entity CELL-0 is active and the model $CELL{-}0^{(1,2)}$ is solved to obtain the intermediate results $O^{(2)}_{CELL-0\to\{CELL-1\}}$, $O^{(2)}_{CELL-0\to\{CELL-2\}}$ and $O^{(2)}_{CELL-0\to\{CELL-3\}}$, thus removing the dependency relation $CELL-0 \to_2 \{CELL-1, CELL-2, CELL-3\}$. Now the models $CELL-1^{(1,2,3,4,5)}$, $CELL-2^{(1,2,3,4,5)}$ and $CELL-3^{(1,2,3,4,5)}$ can be solved, obtaining i) the measures of interest for CELL-1, CELL-2 and CELL-3, and ii) the intermediate results $O^{(4)}_{CELL-1\to\{CELL-0\}}$, $O^{(4)}_{CELL-2\to\{CELL-0\}}$ and $O^{(4)}_{CELL-3\to\{CELL-0\}}$, that remove the dependency relations $CELL{-}1 \to_4 \{CELL{-}0\}$, $CELL{-}2 \to_4 \{CELL{-}0\}$ and $CELL-3 \to_4 \{CELL-0\}$. Finally, the model $CELL-0^{(1,2,3,4,5)}$ can be solved producing the measures of interest for CELL-0.

In the following Sections we show the models required for each computation, following the order of the rows defined in Table 5.1.

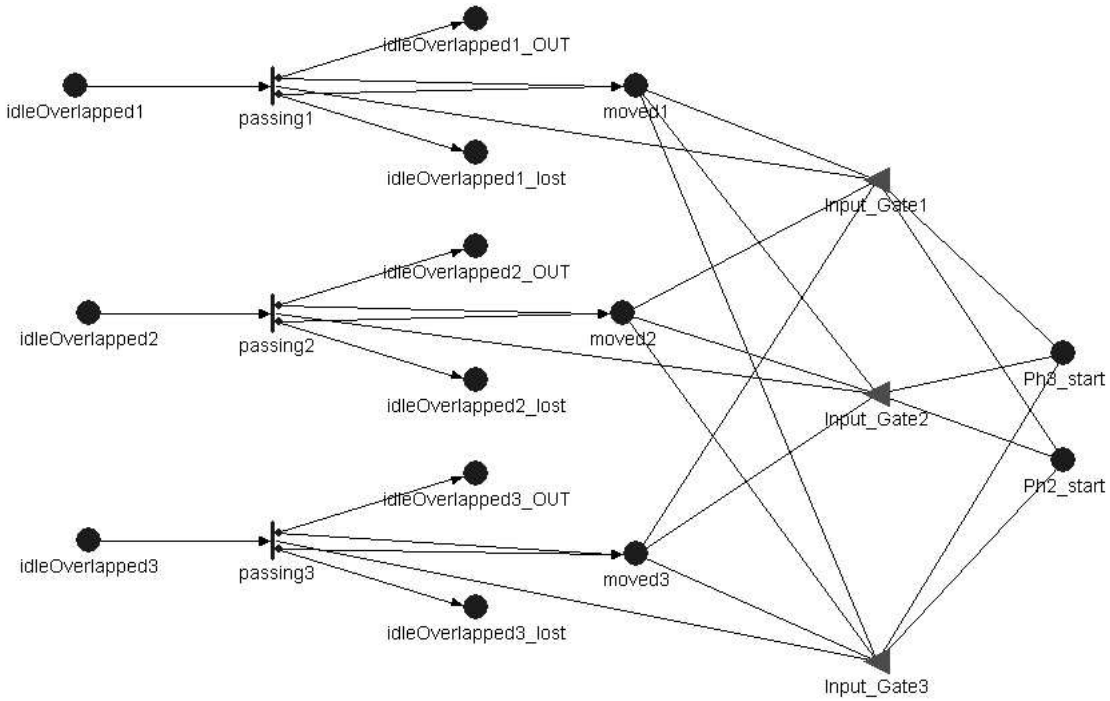

Figure 5.9: The $\varepsilon^{(2),OUT}_{CELL-0\rightarrow\{CELL-1,CELL-2,CELL-3\}}$ model

5.5 Row 1

Model required: $CELL{-}0^{(1,2)} = CELL{-}0^{\circ} \cup \varepsilon^{(2),OUT}_{CELL-0\rightarrow\{CELL-1,CELL-2,CELL-3\}}$;

Input: no;

Output: $O^{(2)}_{CELL-0\rightarrow\{CELL-1,CELL-2,CELL-3\}} = [O^{(2)}_{CELL-0\rightarrow\{CELL-1\}}, O^{(2)}_{CELL-0\rightarrow\{CELL-2\}}, O^{(2)}_{CELL-0\rightarrow\{CELL-3\}}]$;

5.5.1 $\varepsilon^{(2),OUT}_{CELL-0\rightarrow\{CELL-1,CELL-2,CELL-3\}}$ model

The template of the "Dependency Relations - OUT" submodel is shown in Figure 5.9 and it is used during the switching procedure (phase 2) to specify the behavior of the neutral cell CELL to be active (sending cell).

The model is the same as the one presented in Section 5.2.3 (Figure 5.6) with only one difference: the switched users are not directly moved to CELL-1, CELL-2 and CELL-3 but they are collected in three places, `moved1`, `moved2` and `moved3`.

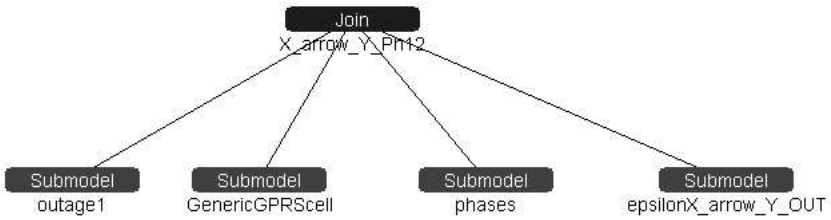

Figure 5.10: The overall model for $CELL - 0^{(1,2)}$

5.5.2 $CELL - 0^{(1,2)}$ model

In Figure 5.10 we show the overall model `X_arrow_Y_Ph12` representing the real behavior of CELL-0 during phase 1 and 2, thus accounting for the switching procedure. It is obtained joining the following submodels: `outage1` ("outage model"), `phases` ("phases model"), `GenericGPRScell` ($CELL - 0°$) and `epsilonX_arrow_Y_OUT` ($\varepsilon^{(2),OUT}_{CELL-0\rightarrow\{CELL-1,CELL-2,CELL-3\}}$). The models share all the places having the same name.

5.5.3 $O^{(2)}_{CELL-0\rightarrow\{CELL-1,CELL-2,CELL-3\}}$ computation

The goal is to compute the intermediate result $O^{(2)}_{CELL-0\rightarrow\{CELL-1,CELL-2,CELL-3\}} = [O^{(2)}_{CELL-0\rightarrow\{CELL-1\}}, O^{(2)}_{CELL-0\rightarrow\{CELL-2\}}, O^{(2)}_{CELL-0\rightarrow\{CELL-3\}}]$ that captures the effect that CELL-0 induces on CELL-1, CELL-2 and CELL-3 during phase 2. In general, two cells interact when a user is switched (or re-switched) from a cell to another, and this happens when a given activity fires, thus changing the states of the corresponding two models. The behavior of an activity can be captured defining a measure for the time of each activity completion: one measure for the time of the first completion, one for the time of the second completion, and so on. The set of these solutions would form a random process that exactly captures the effect of the action on the other model. The problem is that it is not possible to pass a random process as an intermediate result, and then we can only compute some results that approximatively represent the random process. In our case-study, we approximate each random variable with its mean.

For this purpose we define a set of 50 rate reward variables[1] for each overlapping cell, and we accumulate the rewards over the period [0;T4] (until re-switching time, as the time T4 the switching procedure is certainly stopped). The rate reward for the variable number i (with $i = 0, \ldots, 49$) concerning the overlap-

[1]We are here assuming that no more than 50 users can be switched from CELL-0 to CELL-j (with $j = 1, 2$), but this value can be increased as required.

ping cell j (with $j = 1, 2, 3$), that we call "rew$j$DTi", is defined as follows:

```
if (epsilonX_arrow_Y_OUT->moved$j$->Mark()<=$i$) return(1);
else return(0);
```

This means that rew$j$DTi accumulates a rate reward equal to 1 until activity passing$j$ of model epsilonX_arrow_Y_OUT has completed for the i-th time, and it corresponds to the time elapsed until the i-th user has been switched from CELL-0 to CELL-j. The users that never switch have a corresponding rew$j$DTi value equal to T4. Therefore, the intermediate result consists of three arrays (double-precision values) such that the i-th element is the expectation ($E[\cdot]$) of the i-th reward variable:

$$O^{(2)}_{CELL-0\rightarrow\{CELL-1\}} = double[50] = \{E[\texttt{rew01DT0000}], ..., E[\texttt{rew01DT0049}]\},$$

$$O^{(2)}_{CELL-0\rightarrow\{CELL-2\}} = double[50] = \{E[\texttt{rew02DT0000}], ..., E[\texttt{rew02DT0049}]\},$$

$$O^{(2)}_{CELL-0\rightarrow\{CELL-3\}} = double[50] = \{E[\texttt{rew03DT0000}], ..., E[\texttt{rew03DT0049}]\}.$$

5.6 Row 2 (or 3, or 4)

Model required: $CELL - 1^{(1,2,3,4,5)} = CELL - 1^{\circ} \cup \varepsilon^{(4),OUT}_{CELL-1\rightarrow\{CELL-0\}} \cup \varepsilon^{(2),IN}_{CELL-0\rightarrow\{CELL-1\}}$;

Input: $O^{(2)}_{CELL-0\rightarrow\{CELL-1\}}$;

Output: $O^{(4)}_{CELL-1\rightarrow\{CELL-0\}} \cup M_{CELL-1}$;

5.6.1 $\varepsilon^{(2),IN}_{CELL-0\rightarrow\{CELL-1\}}$ model

The template of the "Dependency Relations - IN" submodel is shown in Figure 5.11 and it is used during the switching procedure (phase 2) to specify how the neutral cell CELL-1 modifies to become passive (receiving cell).

We define a set of 50 global variables, DepartureTime0000, ..., DepartureTime0049 (double-precision values), such that the i-th variable contains the mean time elapsed until the i-th idle user has been switched from CELL-0 to CELL-1. These values can be obtained from the previously produced intermediate result $O^{(2)}_{CELL-0\rightarrow\{CELL-1\}}$ through the following (manually made) settings:

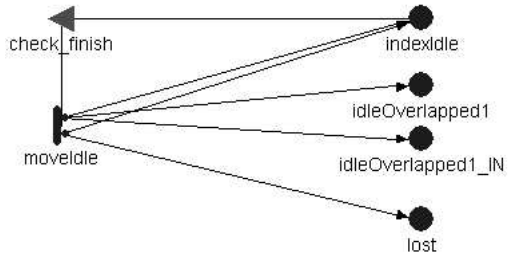

Figure 5.11: The $\varepsilon^{(2),IN}_{CELL-0\rightarrow\{CELL-1\}}$ model

`DepartureTime00`i = $O^{(2)}_{CELL-0\rightarrow\{CELL-1\}}[i]$, with $i = 00, 01, \ldots, 49$.

These global variables are then put in the local array `Dep`, defined inside the input gate `check_finish`, such that `Dep[`i`]=DepartureTime00`i (with $i = 00, 01, \ldots, 49$). The effect of the switching procedure from CELL-0 to CELL-1 is then reconstructed through the deterministic activity `moveIdle` that fires at time `Dep[0]` for the first time, and at time `Dep[indexIdle->Mark()]-Dep[indexIdle->Mark()-1]` for the other completions (mean time elapsed between two consecutive firings). If `Dep[indexIdle->Mark()]` is equal to T4 (the re-switching time), no more users must be switched and then `check_finish` disables the deterministic activity `moveIdle`.

The $\varepsilon^{(2),IN}_{CELL-0\rightarrow\{CELL-2\}}$ model is built in the same way: we just consider that the i-th global variable contains the mean time elapsed until the i-th idle user has been switched from CELL-0 to CELL-2, that is `DepartureTime00`i = $O^{(2)}_{CELL-0\rightarrow\{CELL-2\}}[i]$, with $i = 00, 01, \ldots, 49$. The same considerations hold for the $\varepsilon^{(2),IN}_{CELL-0\rightarrow\{CELL-3\}}$ model.

5.6.2 $\varepsilon^{(4),OUT}_{CELL-1\rightarrow\{CELL-0\}}$ model

The template of the model is shown in Figure 5.12, and it is used during the re-switching procedure (phase 4) to specify the behavior of the neutral cell CELL-1 to be active (sending cell).

The model works as follows. The input gate `Input_Gate1` enables the instantaneous activity `pass_Y` if the re-switching procedure is started (one token in place `Ph4_start`). Therefore, all the users previously switched from CELL-0 to CELL-1 (corresponding to the number of tokens in place `idleOverlapped1_IN`) are re-switched to the original cell as soon as they become available (that is, as soon as the tokens arrive in place `idleOverlapped1`).

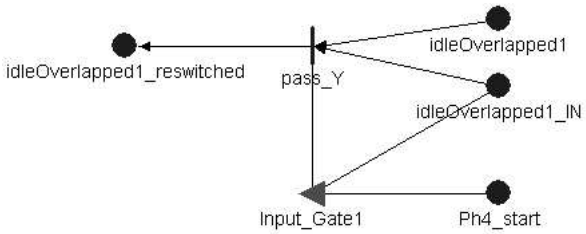

Figure 5.12: The $\varepsilon^{(4),OUT}_{CELL-1\rightarrow\{CELL-0\}}$ model

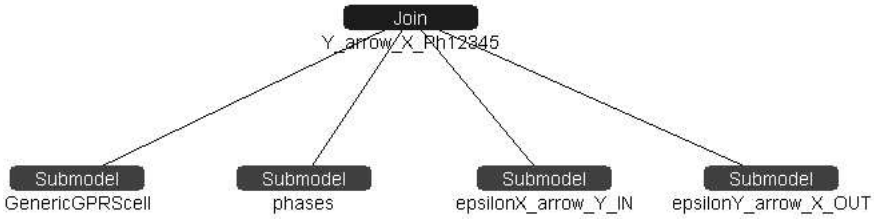

Figure 5.13: The overall model for $CELL-1^{(1,2,3,4,5)}$

The models $\varepsilon^{(4),OUT}_{CELL-2\rightarrow\{CELL-0\}}$ and $\varepsilon^{(4),OUT}_{CELL-3\rightarrow\{CELL-0\}}$ are built similarly.

5.6.3 $CELL-1^{(1,2,3,4,5)}$ model

In Figure 5.13 we show the overall model `Y_arrow_X_Ph12345` representing the real behavior of CELL-1 during all the system lifetime (from phase 1 to phase 5), thus accounting for the switching and re-switching procedures. It is obtained joining the following submodels: `phases` ("phases model"), `GenericGPRScell` ($CELL-1^{\circ}$), `epsilonX_arrow_Y_IN` ($\varepsilon^{(2),IN}_{CELL-0\rightarrow\{CELL-1,CELL-2,CELL-3\}}$) and `epsilonY_arrow_X_OUT` ($\varepsilon^{(4),OUT}_{CELL-1\rightarrow\{CELL-0\}}$). The models share all the places having the same name.

The models $CELL-2^{(1,2,3,4,5)}$ and $CELL-3^{(1,2,3,4,5)}$ are built similarly.

5.6.4 $O^{(4)}_{CELL-1\rightarrow\{CELL-0\}}$ computation

The goal is to compute the intermediate result $O^{(4)}_{CELL-1\rightarrow\{CELL-0\}}$ capturing the effect that CELL-1 induces on CELL-0 during the re-switching procedure. As described in Section 5.5.3, we approximate the random variable representing the i-th completion time of a given activity with its mean.

The intermediate result $O^{(4)}_{CELL-1\rightarrow\{CELL-0\}}$ is computed as here described.

We define a set of 50 rate reward variables[2] and we accumulate the rewards over the period [0;T6] (T6 is chosen such that the re-switching procedure is certainly completed). The rate reward for the variable number i (with $i = 0, \ldots, 49$), that we call "rewDTi", is defined as it follows:

```
if (epsilonY_arrow_X_OUT->idleOverlapped1_reswitched->Mark()<=i)
return(1);
else return(0);
```

This means that rewDTi accumulates a rate reward equal to 1 until activity pass_Y of model epsilonY_arrow_X_OUT has completed for the i-th time, and it corresponds to the time elapsed until the i-th user has been re-switched from CELL-1 to CELL-0. If n is the total number of re-switched users (with $n < 50$), then the last $50 - n$ reward variables will be equal to T6. Therefore, the intermediate result is an array of doubles such that the i-th element is the expectation ($E[\cdot]$) of the i-th reward variable:

$$O^{(4)}_{CELL-1 \rightarrow \{CELL-0\}} = double[50] = \{E[\texttt{rewDT0000}], ..., E[\texttt{rewDT0049}]\} \quad .$$

The intermediate results $O^{(4)}_{CELL-2 \rightarrow \{CELL-0\}}$ and $O^{(4)}_{CELL-3 \rightarrow \{CELL-0\}}$ are computed similarly solving the correspondent models $CELL{-}2^{(1,2,3,4,5)}$ and $CELL{-}3^{(1,2,3,4,5)}$.

5.6.5 M_{CELL-1} computation

M_{CELL-1} represents the Pointwise Congestion function (PCf) perceived by the users camped in CELL-1 at several instants of time. The value PCf(*t*), where *t* is the instant of time in which it is calculated, is computed using the following rate reward variable definition:

```
double users = totalNumUsersCELL1 + GenericGPRScell->idleOverlapped1_IN->Mark();

double usersSatisfied = GenericGPRScell->idle->Mark() +
GenericGPRScell->idleOverlapped1->Mark()+ GenericGPRScell->ch1->Mark() +
GenericGPRScell->ch2->Mark() + GenericGPRScell->ch3->Mark()+
GenericGPRScell->ch4->Mark() + GenericGPRScell->ch5->Mark() +
GenericGPRScell->ch6->Mark()+ GenericGPRScell->ch7->Mark() +
GenericGPRScell->a1->Mark() + GenericGPRScell->a2->Mark()+
```

[2]We are here assuming that no more than 50 users can be re-switched from CELL-j (with $j = 1, 2, 3$) to CELL-0, but this value can be increased as required.

```
GenericGPRScell->a3->Mark() + GenericGPRScell->a4->Mark() +
GenericGPRScell->a5->Mark()+ GenericGPRScell->a6->Mark() +
GenericGPRScell->a7->Mark();

double reward = (users - usersSatisfied) / users * 100;
return (reward);
```

The total number of users attached to CELL-1 (`users`) is the sum of the original number of users camped in CELL-1 (`totalNumUsersCELL1`) and the number of users that have been switched from CELL-0 to CELL-1 (`GenericGPRScell->idleOverlapped1_IN->Mark()`). The satisfied users (`usersSatisfied`) are those that are not requiring any service or are currently being served, therefore the number of unsatisfied users is equal to `users` - `usersSatisfied`.

The computations for M_{CELL-2} and M_{CELL-3} are performed in the same way.

5.7 Row 5

Model required: $CELL - 0^{(1,2,3,4,5)} = CELL - 0^{\circ} \cup \varepsilon^{(2),OUT}_{CELL-0 \rightarrow \{CELL-1,CELL-2,CELL-3\}} \cup \varepsilon^{(4),IN}_{CELL-1 \rightarrow \{CELL-0\}} \cup \varepsilon^{(4),IN}_{CELL-2 \rightarrow \{CELL-0\}} \cup \varepsilon^{(4),IN}_{CELL-3 \rightarrow \{CELL-0\}}$;

Input: $O^{(4)}_{CELL-1 \rightarrow \{CELL-0\}} \cup O^{(4)}_{CELL-2 \rightarrow \{CELL-0\}} \cup O^{(4)}_{CELL-3 \rightarrow \{CELL-0\}}$;

Output: M_{CELL-0};

5.7.1 $\varepsilon^{(4),IN}_{CELL-1 \rightarrow \{CELL-0\}}$ model

The template of the "Dependency Relations - IN" submodel is shown in Figure 5.14 and it is used during the re-switching procedure (phase 4) to specify the behavior of the neutral cell CELL-0 to be passive (it receives the users from CELL-1).

We define a set of 50 global variables, `DepartureTime01_0000`, ..., `DepartureTime01_0049` (double-precision values), such that the i-th variable contains the mean time elapsed until the i-th idle user has been re-switched from CELL-1 to CELL-0. These values can be obtained from the previously produced intermediate result $O^{(4)}_{CELL-1 \rightarrow \{CELL-0\}}$ through the following (manually made) settings:

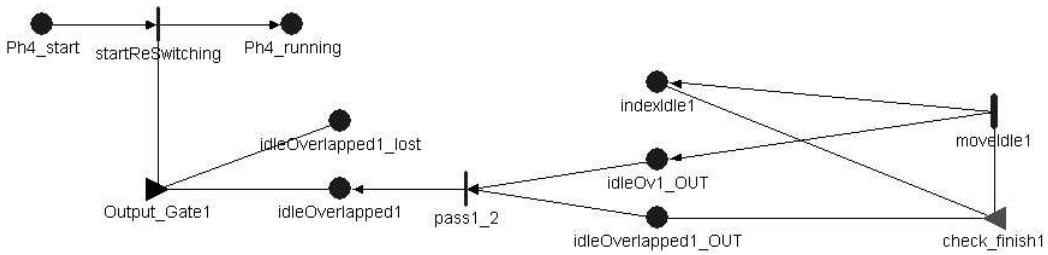

Figure 5.14: The $\varepsilon^{(4),IN}_{CELL-1\rightarrow\{CELL-0\}}$ model

`DepartureTime01_00`i = $O^{(4)}_{CELL-1\rightarrow\{CELL-0\}}[i]$, with $i = 00, 01, \ldots, 49$.

These global variables are then put in the local array `Dep`, defined in the input gate `check_finish1`, such that `Dep[`i`]=DepartureTime01_00`i (with $i = 00, 01, \ldots, 49$). The effect of the re-switching procedure from CELL-1 to CELL-0 is then reconstructed through the deterministic activity `moveIdle1` that fires at time `Dep[0]` for the first time, and at time `Dep[indexIdle->Mark()]-Dep[indexIdle->Mark()-1]` for the other completions (mean time elapsed between two consecutive firings). If `Dep[indexIdle->Mark()]` is equal to T6 (the system lifetime), no more users must be re-switched and then the the gate `check_finish1` disables the activity `moveIdle1`.

The model works as follows. When the re-switching procedure starts (`Ph4_start` contains one token), the instantaneous activity `startReSwitching` fires and the output gate `Output_Gate1` executes the following actions: i) tokens in place `idleOverlapped1_lost` are added to place `idleOverlapped1`, and ii) the marking of `idleOverlapped1_lost` is set to zero. This means that all the idle users previously lost during the switching procedure from CELL-0 to CELL-1 have been re-camped in the central cell. The users previously switched from CELL-0 to CELL-1 are re-switched following the distribution defined in the `moveIdle1` activity.

The model $\varepsilon^{(4),IN}_{CELL-2\rightarrow\{CELL-0\}}$ is built in the same way: we just consider that the i-th global variable contains the mean time elapsed until the i-th idle user has been re-switched from CELL-2 to CELL-0, that is `DepartureTime01_00`i = $O^{(4)}_{CELL-2\rightarrow\{CELL-0\}}[i]$, with $i = 00, 01, \ldots, 49$. The same considerations hold for the $\varepsilon^{(4),IN}_{CELL-3\rightarrow\{CELL-0\}}$ model.

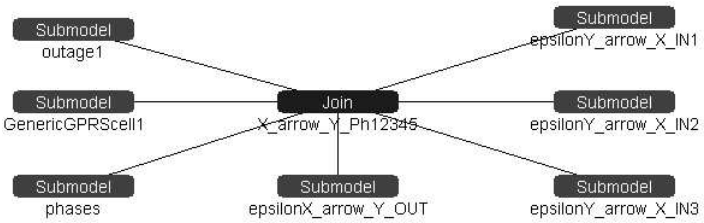

Figure 5.15: The overall model for $CELL - 0^{(1,2,3,4,5)}$

5.7.2 $CELL - 0^{(1,2,3,4,5)}$ model

In Figure 5.15 we show the overall model `X_arrow_Y_Ph12345` representing the real behavior of CELL-0 during all the system lifetime (from phase 1 to phase 5), thus accounting for the switching and re-switching procedures. It is obtained joining the following submodels: `outage1` ("outage model"), `phases` ("phases model"), `GenericGPRScell` ($CELL - 0°$), `epsilonX_arrow_Y_OUT` ($\varepsilon^{(2),OUT}_{CELL-0\rightarrow\{CELL-1,CELL-2,CELL-3\}}$), `epsilonY_arrow_X_IN1` ($\varepsilon^{(4),IN}_{CELL-1\rightarrow\{CELL-0\}}$), `epsilonY_arrow_X_IN2` ($\varepsilon^{(4),IN}_{CELL-2\rightarrow\{CELL-0\}}$) and `epsilonY_arrow_X_IN3` ($\varepsilon^{(4),IN}_{CELL-3\rightarrow\{CELL-0\}}$). The models share all the places having the same name.

5.7.3 M_{CELL-0} computation

M_{CELL-0} represents the Pointwise Congestion function (PCf) perceived by the users camped in CELL-0 at several instants of time. The value PCf(*t*), where *t* is the instant of time in which it is calculated, is computed using the following rate reward variable definition:

```
double users = totalNumUsersCELL0 - GenericGPRScell->idleOverlapped1_OUT->Mark() -
GenericGPRScell->idleOverlapped2_OUT->Mark() -
GenericGPRScell->idleOverlapped3_OUT->Mark();

double usersSatisfied = GenericGPRScell->idle->Mark() +
GenericGPRScell->idleOverlapped1->Mark()+ GenericGPRScell->idleOverlapped2->Mark()+
GenericGPRScell->idleOverlapped3->Mark() + GenericGPRScell->ch1->Mark()+
GenericGPRScell->ch2->Mark() + GenericGPRScell->ch3->Mark()+
GenericGPRScell->ch4->Mark() + GenericGPRScell->ch5->Mark() +
GenericGPRScell->ch6->Mark()+ GenericGPRScell->ch7->Mark() +
GenericGPRScell->a1->Mark() + GenericGPRScell->a2->Mark()+
GenericGPRScell->a3->Mark() + GenericGPRScell->a4->Mark() +
```

```
GenericGPRScell->a5->Mark()+ GenericGPRScell->a6->Mark() +
GenericGPRScell->a7->Mark();

double reward = (users - usersSatisfied) / users * 100;
return (reward);
```

The total number of users attached to CELL-0 (`users`) is the difference between the original number of users camped in CELL-0 (`totalNumUsersCELL0`) and the number of users that have been switched from CELL-0 to CELL-1, CELL-2 and CELL-3 (`GenericGPRScell->idleOverlapped`i`_OUT->Mark()`, with $i = 1, 2, 3$). The satisfied users (`usersSatisfied`) are those that are not requiring any service or are currently being served, therefore the number of unsatisfied users is equal to `users` - `usersSatisfied`.

5.8 The "connected model" to automatically pass the intermediate results between models

A weakness of the presented methodology is that it requires to manually pass the intermediate results between models, writing the expectations of the reward variables in the corresponding global variables. In order to avoid this manual transcription and to speed up the solution process, we use the connection formalism [62, 16] that has been recently implemented in the Möbius [61] tool.

A connected model is a graph consisting of a set of model nodes (called "solvable models"), arcs (called "conduits"), and transformation nodes (called "connection functions"). A solvable model is a model with a specified reward structure that can be solved, a conduit transfers results, while a connection function takes a set of results to generate an input parameter for another model. The connection graph is traversed in a certain order. First a solvable model is solved, and it writes the results to a results database. The output conduits for that solvable model transfer specific results from the model to a connection function. Then the connection function performs some mathematical transformation on all the results passed to it, and a conduit provides this value to another solvable model. The value is passed into the solvable model via a global variable.

In Figure 5.16 we show the connected model that we build to automatically pass the intermediate results between models.

Here we describe the order in which the connection graph has to be traversed.

1. We solve the solvable model `CELL0arrowCELL1_2_3_Ph12`, that corresponds to the `X_arrow_Y_Ph12` model of Section 5.5.2 ($CELL - 0^{(1,2)}$),

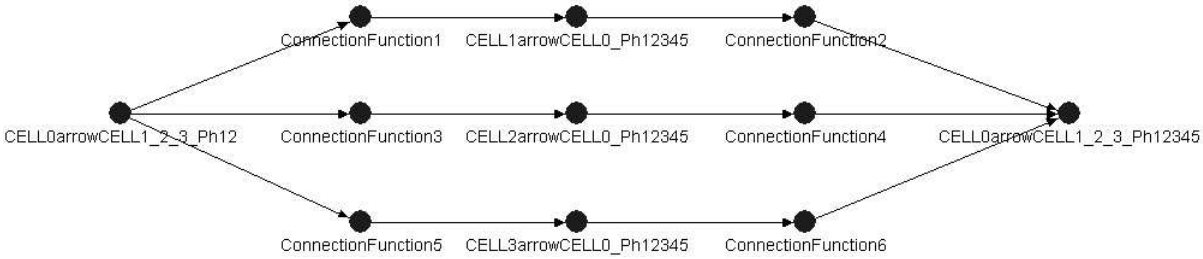

Figure 5.16: The connected model

and the intermediate results $O^{(2)}_{CELL-0\rightarrow\{CELL-1\}}$, $O^{(2)}_{CELL-0\rightarrow\{CELL-2\}}$ and $O^{(2)}_{CELL-0\rightarrow\{CELL-3\}}$ are saved to the results database.

2. We solve the solvable model `ConnectionFunction1`, that automatically executes the following operations:

   `DepartureTime0000 =` E`[rew01DT0000];`
   ...
   `DepartureTime0049 =` E`[rew01DT0049];`

3. We solve the solvable model `CELL1arrowCELL0_12345`, that corresponds to the `Y_arrow_X_Ph12345` model of Section 5.6.3 ($CELL-1^{(1,2,3,4,5)}$), thus producing the measure of interest M_{CELL-1} and the intermediate result $O^{(4)}_{CELL-1\rightarrow\{CELL-0\}}$, that is saved to the results database.

4. We solve the solvable model `ConnectionFunction2`, that automatically executes the following operations:

   `DepartureTime01_0000 = rewDT0000;`
   ...
   `DepartureTime01_0049 = rewDT0049;`

5. We solve the solvable model `ConnectionFunction3`, that automatically executes the following operations:

   `DepartureTime0000 =` E`[rew02DT0000];`
   ...
   `DepartureTime0049 =` E`[rew02DT0049];`

6. We solve the solvable model `CELL2arrowCELL0_12345`, that corresponds to the `Y_arrow_X_Ph12345` model of Section 5.6.3 ($CELL-$

$2^{(1,2,3,4,5)}$), thus producing the measure of interest M_{CELL-2} and the intermediate result $O^{(4)}_{CELL-2 \to \{CELL-0\}}$, that is saved to the results database.

7. We solve the solvable model `ConnectionFunction4`, that automatically executes the following operations:

   ```
   DepartureTime02_0000 = rewDT0000;
   ...
   DepartureTime02_0049 = rewDT0049;
   ```

8. We solve the solvable model `ConnectionFunction5`, that automatically executes the following operations:

   ```
   DepartureTime0000 = E[rew03DT0000];
   ...
   DepartureTime0049 = E[rew03DT0049];
   ```

9. We solve the solvable model `CELL3arrowCELL0_12345`, that corresponds to the `Y_arrow_X_Ph12345` model of Section 5.6.3 ($CELL-3^{(1,2,3,4,5)}$), thus producing the measure of interest M_{CELL-3} and the intermediate result $O^{(4)}_{CELL-3 \to \{CELL-0\}}$, that is saved to the results database.

10. We solve the solvable model `ConnectionFunction6`, that automatically executes the following operations:

    ```
    DepartureTime03_0000 = rewDT0000;
    ...
    DepartureTime03_0049 = rewDT0049;
    ```

11. We solve the solvable model `CELL0arrowCELL1_2_3_Ph12345`, that corresponds to the `X_arrow_Y_Ph12345` model of Section 5.7.2 ($CELL-0^{(1,2,3,4,5)}$), thus producing the measure of interest M_{CELL-0}.

5.9 Model evaluation

A simulation approach is here used to numerically solve the sub-models obtained applying the methodology, using the simulator offered by the Möbius tool. For each study we execute a minimum of 1000 simulation runs (batches), thus the mean of each random variable is computed from a set of at least 1000 samples. Moreover, we set the relative confidence interval to 0.1 and the confidence level to 0.95. This means that the stopping criteria will not be satisfied until the confidence interval is within 10% of the mean estimate 95% of the time.

Symbol	Description	Values
totalNumUsersCELL0	Total number of users camped in CELL-0	180
numUsersCELL0_OvCELL1	Number of users camped in the overlapping area between CELL-0 and CELL-1	60
numUsersCELL0_OvCELL2	Number of users camped in the overlapping area between CELL-0 and CELL-2	50
numUsersCELL0_OvCELL3	Number of users camped in the overlapping area between CELL-0 and CELL-3	40
totalNumUsersCELL1	Total number of users camped in CELL-1	140
totalNumUsersCELL2	Total number of users camped in CELL-2	170
totalNumUsersCELL3	Total number of users camped in CELL-3	200
numIdleUsersToSwitchToCELL1	Number of idle users to switch from CELL-0 to CELL-1	42
numIdleUsersToSwitchToCELL2	Number of idle users to switch from CELL-0 to CELL-2	35
numIdleUsersToSwitchToCELL3	Number of idle users to switch from CELL-0 to CELL-3	28

Table 5.2: Some parameters and their values

5.9.1 Settings for the numerical evaluation and the Analyzed Scenario

As in Chapter 2, we analyze a GPRS network composed by one central cell (CELL-0) and three partially overlapping cells (CELL-1, CELL-2 and CELL-3). The values we assigned to the main parameters of each cell are the same as those detailed in Table 2.1, with the difference that now the switching/re-switching procedure involves idle users only. For the sake of clarity, we report in Table 5.2 the settings for the parameters defining the topology of the infrastructure.

As in Chapter 2, we suppose that 10% of idle users to switch are lost during the reconfiguration action, that the switching and re-switching procedures are instantaneous, and that the partial outage affecting the central cell consists of a software error that reduces the number of available traffic channels from 3 to 1. We set the outage duration to 120 seconds (average time needed to restart the software), the *outageReactionTime* parameter is set to 30 seconds (the time that occurs between the outage and the beginning of the users switching procedure), and the *outageEndReactionTime* parameter is set to 15 seconds (the time that occurs between the end of the outage and the users re-switching).

5.9.2 Numerical evaluation: solving the whole non-decomposed model

We first solve the whole non-decomposed model built as presented in Section 5.2, and the obtained results are considered to be "exact" (within the specified confidence interval), since no approximation has been introduced due to the application of the solution technique.

In Figure 5.17 we show the Pointwise Congestion function (PCf) at several instants of time (25 points, starting from time 200 with steps of 15 seconds). As expected, the congestion perceived by the users camped in CELL-0 rapidly increases at time T0 due to the outage, decreases at time T1 (switching time) and

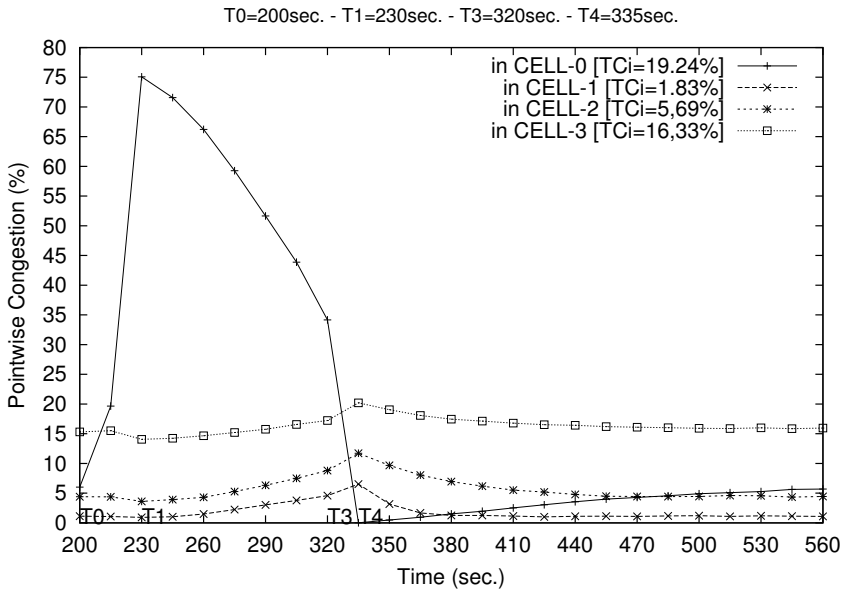

Figure 5.17: PCf for each cell

increases again at time T4 (re-switching time). On the other hand, the congestion perceived in each receiving cell increases at time T1 and decreases at time T4.

Concerning computational time, the simulator has provided the results within the desired accuracy in **8769** seconds.

5.9.3 Numerical evaluation: solving the decomposed models

The same GPRS scenario has been solved using the modeling and solution approach defined in Chapter 4. In Table 5.3 we show the absolute and relative errors affecting the final results in each of the 25 sampling points (instants of time), with respect to the "exact" solution of Figure 5.17. The absolute error is always less than 0.81 (corresponding to a relative error equal to 5.79%), and the maximum relative error is equal to 15% (corresponding to an absolute error of 0.14). The average (absolute and relative) errors are even lower. Although most of the errors are very low, some of them seem to be not negligible. The problem is that the errors of Table 5.3 are the result of two types of approximations: the first induced by the decomposed solution process, and the second caused by the simulation approach that only provides an estimation of the measures. In addition, the "exact" solution that we use to evaluate the accuracy of the results is, in turn, another approximated solution, since obtained through simulation. For these considerations all the errors of Table 5.3 can be still considered acceptable.

Point	CELL-0	CELL-1	CELL-2	CELL-3
1	0.33 (5.43%)	0.05 (4.34%)	0.07 (1.61%)	0.06 (0.40%)
2	0.45 (2.28%)	0.06 (5.86%)	0.13 (2.96%)	0.17 (1.12%)
3	0.39 (0.52%)	***0.14 (15.00%)***	0.39 (10.91%)	***0.81 (5.79%)***
4	0.75 (1.05%)	0.08 (7.61%)	0.01 (0.20%)	0.27 (1.93%)
5	0.68 (1.03%)	0.15 (10.15%)	0.07 (1.57%)	0.43 (2.91%)
6	0.55 (0.93%)	0.15 (6.61%)	0.23 (4.38%)	0.43 (2.86%)
7	0.32 (0.63%)	0.07 (2.42%)	0.19 (2.98%)	0.36 (2.31%)
8	0.01 (0.02%)	0.10 (2.60%)	0.17 (2.24%)	0.26 (1.58%)
9	0.05 (0.16%)	0.15 (3.22%)	0.01 (0.16%)	0.16 (0.93%)
10	0.00 (6.30%)	0.08 (1.30%)	0.17 (1.44%)	0.13 (0.64%)
11	0.01 (2.31%)	0.04 (1.13%)	0.23 (2.36%)	0.12 (0.64%)
12	0.01 (0.75%)	0.07 (4.13%)	0.35 (4.41%)	0.10 (0.57%)
13	0.03 (1.76%)	0.00 (0.15%)	0.12 (1.77%)	0.13 (0.76%)
14	0.16 (8.09%)	0.11 (8.92%)	0.13 (2.11%)	0.04 (0.26%)
15	0.14 (5.63%)	0.06 (5.62%)	0.18 (3.35%)	0.09 (0.51%)
16	0.29 (9.44%)	0.09 (9.13%)	0.08 (1.53%)	0.24 (1.43%)
17	0.33 (9.25%)	0.00 (0.35%)	0.23 (4.72%)	0.08 (0.49%)
18	0.28 (6.93%)	0.07 (6.37%)	0.34 (7.62%)	0.02 (0.15%)
19	0.29 (6.70%)	0.02 (1.54%)	0.27 (6.02%)	0.16 (0.97%)
20	0.13 (2.83%)	0.06 (4.76%)	0.18 (4.14%)	0.16 (1.00%)
21	0.02 (0.50%)	0.00 (0.16%)	0.11 (2.41%)	0.17 (1.06%)
22	0.15 (2.92%)	0.14 (13.20%)	0.15 (3.35%)	0.13 (0.79%)
23	0.05 (0.97%)	0.04 (3.28%)	0.16 (3.56%)	0.07 (0.42%)
24	0.18 (3.18%)	0.00 (0.32%)	0.10 (2.32%)	0.34 (2.17%)
25	0.01 (0.16%)	0.05 (4.84%)	0.01 (0.26%)	0.24 (1.52%)
Average	***0.22 (3.19%)***	***0.07 (4.92%)***	***0.16 (3.14%)***	***0.21 (1.33%)***

Table 5.3: Absolute (and relative) errors for CELL-0, CELL-1, CELL-2 and CELL-3, using the general approach of Chapter 4

The computational time needed to produce the results are shown in Table 5.4. The total time is equal to 6669 seconds that is about 24% less than the computational time required to solve the model as a whole (8769 seconds). Moreover, if we suppose to execute the simulations on different machines, the total time needed to have the solution is about 4663 seconds, since $CELL-1^{(1,2,3,4,5)}$, $CELL-2^{(1,2,3,4,5)}$ and $CELL-3^{(1,2,3,4,5)}$ can be computed concurrently, and then we take their maximum computational time only.

As a final study, the same scenario has been solved using the methodology

Model solved	Elapsed running time (in seconds)
$CELL - 0^{(1,2)}$	1215
$CELL - 1^{(1,2,3,4,5)}$	775
$CELL - 2^{(1,2,3,4,5)}$	1231
$CELL - 3^{(1,2,3,4,5)}$	2941
$CELL - 0^{(1,2,3,4,5)}$	507
Total	6669
Total (with parallel executions)	1215+2941+507=4663

Table 5.4: Elapsed running time using the general approach of Chapter 4

	CELL-0	CELL-1	CELL-2	CELL-3
Average	***0.33 (5.62%)***	***0.07 (4.57%)***	***0.13 (2.48%)***	***0.19 (1.13%)***

Table 5.5: Average absolute (and relative) errors for CELL-0, CELL-1, CELL-2 and CELL-3, using the specific approach of Chapter 2

specifically tailored for the GPRS case study. The obtained measures have the same average accuracy as the previous results, as shown in Table 5.5, but the total computational cost slightly decreases, as shown in Table 5.6. The single execution time for each couple of models [CELL-0;CELL-*i*] is slightly higher than the $CELL - i^{(1,2,3,4,5)}$ model, but it has not to account for the solution of the $CELL - 0^{(1,2)}$ model, thus reducing the overall computational time. If we suppose to execute the simulations of each couple [CELL-0;CELL-*i*] on different processors, the total time needed to compute the solution is about 3655 seconds. This result is of course not surprising, considering that the methodology we are evaluating was defined as an ad-hoc solution for the specific application domain.

Model solved	Elapsed running time (in seconds)
[CELL-0;CELL-1]	996
[CELL-0;CELL-2]	1456
[CELL-0;CELL-3]	3151
CELL (type C model)	504
Total	5603
Total (with parallel executions)	3151+504=3655

Table 5.6: Elapsed running time using the specific approach of Chapter 2

5.10 Summary

In this Chapter we demonstrated the use of the interaction-based modeling approach defined in Chapter 4, applying the methodology to the GPRS case study previously solved with a specific solution technique in Chapter 2. With respect to the previous analysis, the modularity of the model representing the system has been enhanced. Moreover, the application of the splitting procedure does not induce an unacceptable degradation on the accuracy of the final results, while the computational complexity is substantially reduced with respect to solve the whole non-decomposed model, and slightly increases with respect to the application of the specific solution technique. A more detailed analysis about the accuracy of the final results should be performed considering analytical/numerical solutions that eliminate the non-accuracy induced by the simulation approach.

Chapter 6

Conclusion

Availability, reliability and quality of service are only few system's properties whose evaluation has become a very critical issue in the last decades. One of the most widely used techniques for their assessment is the model-based approach: we first build a model representing the system at the desired level of detail, and then we solve it by simulation or using some analytical/numerical methods. Unfortunately, the models can be very complex and their solutions can demand a huge amount of memory and/or time to complete, and this severely limits the class of systems that can be really studied using the model-based approach. In the literature there are several techniques that try to manage the complexity problem, and most of them are very useful in their application-domain but they can be hardly re-used in other contexts. The contribution given by this dissertation is twofold: on one hand we developed two modeling and solution technique for the analysis of two specific (although well populated) class of systems, and on the other hand we exploited this knowledge in order to define a modeling and solution framework potentially applicable to all existing systems.

The first study has concerned a GPRS infrastructure composed of a number of partially overlapping cells, and the goal was to analyze the impact of outage phenomenons and reconfiguration actions on the quality of service perceived by the users camped in the network. The overall system complexity, that increases with the increment of the number of cells composing the infrastructure, has been attacked though the definition of a specific decomposition approach that considers the cells in pairs. Using the proposed methodology, a fine-tuning activity has been performed in order to determine i) appropriate values for the number of users to switch from the central cell to the others, and ii) the maximum amount of time that the system can spend for its decision-making process if the congestion must not exceed a given threshold level. These are two critical aspects that must be always evaluated when a cell resizing is performed in a mobile telephone environment.

Although the modeling approach has been developed for the GPRS technology, it shows very attractive potentialities, being it suitable to be employed in the analysis of other similar problems, for example to analyze the behavior of a heterogeneous infrastructure where different network technologies (e.g., GPRS and UMTS) co-operate to reduce a congestion situation.

In the second study we have analyzed a class of hierarchical control systems, in which the functionalities of the whole system are partitioned among a number of subsystems working at different levels of a hierarchy. The dependability of the whole system is enhanced considering, at each level, both internal checks (to cope with faults during the internal computation) and interface checks (to cope with erroneous inputs and/or outputs). A proper modeling methodology based on a decomposition approach has been defined, able to reduce the system complexity. We first decomposed a model starting from its functional specification and applying a stepwise refinement to decompose it in small sub-models (decomposition). Then, the modular model solution has been carried out in a bottom-up fashion (aggregation). The methodology has been applied to evaluate some dependability attributes of the trial developed in the context of the European project CAUTION++. The obtained results have allowed us to understand the impact of several factors contributing to the dependability of the single CAUTION++ components on the overall system instance. Moreover, this study could be useful to guide implementation choices addressing dependability, by providing comparative quantitative assessment of possible alternatives.

The knowledge derived by the two previous described experiences has been exploited to define a general modeling and solution framework for the solution of complex systems. Here the term "general" means that we do not want to limit the analysis neither to a specific class of systems, nor to a particular modeling formalism. The methodology is based on a functional, temporal and physical decomposition. The whole system is decomposed in a set of sub-systems that can work in isolation or can interact each other through some dependency relations (functional decomposition). The lifetime of the system is then decomposed in a sequence of phases such that two consecutive phases have at least one different dependency relation (temporal decomposition). Following a modular approach, we have first modeled the system as a whole, thus identifying the main submodels with respect to the functions they perform. Then the whole model has been decomposed (through the splitting procedure) in a set of submodels to be solved in isolation, passing some intermediate results when required (physical decomposition). We proposed two algorithms (a standard version and an optimized one) that, basing on the dependency relation graph for each phase, define i) the order in which the submodels have to be solved, ii) the intermediate results to be produced

by the model solutions, and iii) the intermediate results to be used as input in the model definitions. The decomposition approach is able to reduce the spatial and temporal complexity. On the other side, the accuracy of a final result has to be carefully treated and it mainly depends i) on the accuracy of the intermediate results used to produce it, both in terms of machine precision (for analytical solver) and confidence interval (in case of simulation), and ii) on the capability of the "Dependency Relations - IN and OUT" models to perfectly represent the interactions that involve an entity, using the previously produced intermediate results.

Finally, the general modeling and solution approach has been applied to the previously described GPRS case-study to prove its feasibility. The application of the methodology enhances the modularity of the models that perform well-separated functions. The obtained results are accurate as those produced by the application of the modeling and solution technique specifically developed for the GPRS infrastructure, and the computational time is substantially reduced with respect to solve the whole non-decomposed model.

Although not shown in this dissertation, the general modeling and solution framework introduced in Chapter 4 can also be applied to the case study concerning the hierarchical control systems of Chapter 3. With respect to the overall solution scheme of Figure 3.9, we can identify two types of entities: those that correspond to the detailed model of a component (or of a set of identical components) in a stage, and one corresponding to the high level architectural model of the system. The exchange of parameters between two entities represents a dependency relation. The corresponding phased-interacting system is then composed of $N + 1$ entities, if N is the total number of stages, and one phase only, since the dependency relations do not change in time.

A significant contribution of this dissertation is the definition of the general modeling and solution framework, and some considerations can be done comparing it with the decomposition techniques outlined in Section 1.6. All these techniques manage the system complexity decoupling a model in simpler and more tractable submodels, and their solutions are combined to get the solution of the whole system. Most of these techniques can be seen as instances of the proposed framework just identifying an appropriate conceptual-level system decomposition, that is identifying an appropriate set of interacting conceptual submodels (entities) that can be solved in isolation, exchanging intermediate results when required (dependency relations).
For example, the time scale decomposition approach outlined in Section 1.6.2 decomposes a model into a hierarchical sequence of aggregated sub-nets (these are

the entities), each of which characterized by a certain time scale. At each level of the hierarchy, the current marking of an aggregated sub-net determines the number of tokens in the sub-net at the lower level (these are the dependency relations). The modular hierarchical decomposition approach of Section 1.6.4 models a system at various levels of abstraction (the entities), and the output values from one layer are used as parameters of the next higher layer (the dependency relations). The interaction-based decomposition technique of Section 1.6.6 decomposes a large stochastic Petri net in a set of parameterized stochastic Petri net submodels (the entities) that interact through the passing of a parameter value (the dependency relations).

Therefore, several existing decomposition techniques can be mapped in the proposed framework just identifying the entities with respect to an appropriate logic. In the presented examples, the interactions between entities (that concretely consist of some numerical results passed between models) remain fixed during the whole system lifetime, and then they all correspond to a phased-interacting system with several entities and one phase only.

Viceversa, the phased decomposition approach outlined in Section 1.6.5 includes the concept of multiple phases and it deserves special attention. Phased Mission Systems (PMSs) are characterized by a sequence of phases in which the system configuration can change during operations. Conceptually, a PMS could be modeled using a single general model M (the entity) able to represent the specific behavior of the system inside each phase. Let M_i be the specification of the model M representing the behavior of the PMS during phase i, then the model M_i provides to the next model M_{i+1} an intermediate result (the state-occupancy probability vector at the end of phase i) that enables the separate solution of the model M_{i+1} starting from the end of phase i, thus avoiding to re-compute the solution from the first phase. At the moment, this technique can not be confined in the proposed modeling and solution environment, mainly for two reasons: i) the dependency relations among entities have been defined inside each phase (Section 4.2), while the dependency relations required for the application of the phased decomposition approach link entities belonging to two consecutive phases (the model M_i affects the behavior of the model M_{i+1} through the passing of an intermediate result); ii) the proposed algorithms implementing the decomposed solution process (Section 4.4) compute the solution of the models in each phase starting from the first phase, while in the phased decomposition approach the model in a phase is solved just starting from the end of the previous phase. Actually, as sketched in Section 4.4.2, we are planning to modify the modeling and solution framework in order to accommodate this kind of decomposition techniques.

Other future works concern: i) the automation of the splitting procedure, currently done by hand, and the integration of the methodology in the existing modeling tools; ii) a more detailed analysis of the accuracy of the final results with respect to the errors induced by the application of the interaction-based decomposition approach; iii) the application of the methodology to very large scale systems in order to better understand how the methodology is scalable and effective considering very complex case studies.

List of Figures

List of Tables

Bibliography

[1] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. John Wiley and Sons, New York, 2002.

[2] M. Rausand, and A. Høyland. *System Reliability Theory: Models, Statistical Methods, and Applications*. John Wiley and Sons, New Jersey, 2004.

[3] K. Kanoun, and M. Ortalo-Borrel. Fault-tolerant system dependability-explicit modeling of hardware and software component-interactions. *IEEE Transactions on Reliability*, 49(4):363–376, December 2000.

[4] W. H. Sanders. *Construction and solution of performability models based on stochastic activity networks*. PhD thesis, University of Michigan, Michigan, 1988.

[5] W. H. Sanders, and J. F. Meyer. Reduced base model construction methods for stochastic activity networks. *IEEE Journal on Selected Areas in Communications*, 9(1):25–36, January 1991.

[6] D. Obal. *Measure-adaptive state-space construction methods*. PhD thesis, University of Arizona, Arizona, 1998.

[7] M. Rabah, and K. Kanoun. Performability evaluation of multipurpose multiprocessor systems: the "separation of concerns" approach. *IEEE Transactions on Computers*, 52(2):223–236, 2003.

[8] C. Betous-Almeida, and K. Kanoun. Stepwise construction and refinement of dependability models. In *Proc. IEEE International Conference on Dependable Systems and Networks DSN 2002*, Washington D.C., 2002.

[9] S. Bernardi, and S. Donatelli. Stochastic petri nets and inheritance for dependability modelling. In *Proceedings of the 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'04)*, pages 363–372, March 2004.

[10] M. Balakrishnan, and K.S. Trivedi. Componentwise decomposition for an efficient reliability computation of systems with repairable components. In *Int. IEEE Symp. Fault-Tolerant Computing (FTCS-25)*, pages 259–268, 1995.

[11] H. H. Ammar, and S. M. Rezaul Islam. Time scale decomposition of a class of generalized stochastic petri net models. *IEEE Transactions on Software Engineering*, 15(6):809–820, June 1989.

[12] C. M. Woodside, J. E. Neilson, D. C. Petriu, and S. Majumdar. The stochastic rendezvous network model for performance of synchronous client-server-like distributed software. *IEEE Transactions on Computers*, 44(1):20–34, January 1995.

[13] A. Bondavalli, M. Nelli, L. Simoncini, and G. Mongardi. Hierarchical modelling of complex control systems: dependability analysis of a railway interlocking. *International Journal of Computer Systems Science & Engineering*, 4:249–261, 2001.

[14] I. Mura, and A. Bondavalli. Markov regenerative stochastic petri nets to model and evaluate phased mission systems dependability. *IEEE Transactions on Computers*, 50(12):1337–1351, December 2001.

[15] G. Ciardo, and K. S. Trivedi. A decomposition approach for stochastic petri net models. In *Proc. of the Fourth International Workshop on Petri Nets and Performance Models (PNPM91)*, pages 74–83, December 1991.

[16] D. Daly. *Analysis of connection as a decomposition technique*. PhD thesis, University of Illinois, Illinois, 2001.

[17] D. Daly, and W. H. Sanders. A connection formalism for the solution of large and stiff models. In *Proc. 34th Annual Simulation Symposium*, pages 258–265, April 2001.

[18] P. Lollini, A. Bondavalli, and F. Di Giandomenico. Evaluation of the impact of congestion on service availability in gprs infrastructures. In *Lecture Notes in Computer Science*, number 3694, pages 180–195. M. Malek et al., 2005.

[19] S. Porcarelli, F. Di Giandomenico, A. Bondavalli, M. Barbera, and I. Mura. Service level availability estimation of gprs. *IEEE Transactions on Mobile Computing*, 2(3):233–247, 2003.

[20] P. Lollini, A. Bondavalli, F. Di Giandomenico, and S. Porcarelli. Congestion analysis during outage, congestion treatment and outage recovery for simple gprs networks. In *Proc. of the Ninth IEEE Symposium On Computers And Communications (ISCC 2004)*, volume 2, pages 772–778, Alexandria, Egypt, July 2004.

[21] S. Porcarelli, F. Di Giandomenico, P. Lollini, and A. Bondavalli. A modular approach for model-based dependability evaluation of a class of systems. In *International Service Availability Symposium (ISAS)*, Munich, Germany, May 2004.

[22] S. Porcarelli, F. Di Giandomenico, A. Bondavalli, and P. Lollini. Model-based evaluation of a radio resource management system for wireless networks. In *Computing Frontiers (CF)*, pages 51–59, Ischia, Italy, April 2004.

[23] P. Lollini, A. Bondavalli, and F. Di Giandomenico. A modeling methodology for hierarchical control systems and its application. *Journal of the Brazilian Computer Society, Special Issue on Dependable Computing*, 10(3):57–69, 2005.

[24] CAUTION++ IST Project. Capacity utilization in cellular networks of present and future generation++. http://www.telecom.ece.ntua.gr/cautionplus/.

[25] J. C. Laprie. Dependability - its attributes, impairments and means. In H. Kopetz B. Littlewood J.C. Laprie, B. Randell, editor, *Predictably Dependable Computing Systems*, pages 3–24. Springer Verlag.

[26] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, January-March 2004.

[27] D. Avresky, J. Arlat, J. C. Laprie, and Y. Crouzet. Fault injection for formal testing of fault tolerance. *IEEE Transactions on Reliability*, 45(3):443–455, September 1996.

[28] G. Balbo. Introduction to stochastic petri nets. In *Lectures on Formal Methods and Performance Analysis*, volume 2090 of *Lecture Notes in Computer Science*, pages 84–155. Springer Verlag, 2001.

[29] M. Nelli, A. Bondavalli, and L. Simoncini. Dependability modelling and analysis of complex control systems: an application to railway interlocking. In *EDCC-2 European Dependable Computing Conference*, pages 93–110, Taormina, Italy, 1996.

[30] A. Bobbio, and K. S. Trivedi. An aggregation technique for the transient analysis of stiff markov chains. *IEEE Transactions on Computers*, 35(9):803–814, 1986.

[31] A. Bondavalli, S. Chiaradonna, F. Di Giandomenico, and I. Mura. Dependability modeling and evaluation of multiple-phased systems using deem. *IEEE Transactions on Reliability*, 53(4):509–522, December 2004.

[32] D. Cox. A use of complex probabilities in theory of stochastic processes. In *Proc. of the Cambrige Philosophical Society*, volume 53, pages 313–319, 1955.

[33] M. A. Marsan, G. Balbo, and G. Conte. A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. *ACM Transaction on Computer System*, 2(2):93–122, 1984.

[34] M. K. Molloy. Performance analysis using stichastic petri nets. *IEEE Transaction on Computers*, 31:913–917, 1982.

[35] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modeling with Generalized Stochastic Petri Nets*. J. Wiley, 1995.

[36] H. Choi, V. Kulkarni, and K. S. Trivedi. Markov regenerative stochastic petri nets. *Performance Evaluation*, 20:337–356, 1994.

[37] M. A. Marsan, and G. Chiola. On petri nets with deterministic and exponentially distribuited firing times. In *Lecture Notes in Computer Science*, volume 266, pages 132–145. Springer Verlang, 1987.

[38] R. German. Transient analysis of deterministic and stochastic petri nets by the method of supplementary variables. In *Third International Workshop on Modeling, Analysis, and Simulation On Computer and Telecommunication Systems (MASCOTS)*, pages 394–398. IEEE Computer Society, 1995.

[39] A. Movaghar, and J. F. Meyer. Performability modelling with stochastic activity networks. In *Proc. of the Real-Time Systems Symposium*, pages 215–224, 1984.

[40] W. H. Sanders, and J. F. Meyer. Stochastic activity networks: Formal definitions and concepts. In *Lectures on Formal Methods and Performance Analysis*, volume 2090 of *Lecture Notes in Computer Science*, pages 315–343. Springer Verlag, 2001.

[41] W. H. Sanders, and J. F. Meyer. A unified approach for specifying measures of performance, dependability and performability. In *Dependable Computing for Critical Applications*, volume 4 of *Dependable Computing and Fault-Tolerant Systems*, pages 215–237. Springer Verlag, 1991.

[42] W. Van der Aalst, and T. Basten. Life-cycle inheritance: A petri-net based approach. *Application and Theory of Petri Nets*, 1248:62–81, 1997.

[43] W. H. Sanders. Integrated frameworks for multi-level and multi-formalism modeling. In *Proc. the 8th International Workshop on Petri Nets and Performance Models*, pages 2–9, September 1999.

[44] C. Lindemann, A. Reuys, and A. Thümmler. The dspnexpress 2.000 performance and dependability modeling environment. In *Proc. of the 29th Annual Int. Symp. on Fault-Tolerant Computing*, pages 228–231, Madison, Wisconsin, USA, June 1999.

[45] G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo. Greatspn 1.7: Graphical editor and analyzer for timed and stochastic petri nets. *Performance Evaluation*, 24(1-2):47–68, November 1995.

[46] C. Béounes et al. Surf-2: A program for dependability evaluation of complex hardware and software systems. In *Proc. of the 23rd Int. Symp. on Fault-Tolerant Computing*, pages 668–673, Toulouse, France, 1993.

[47] A. Bondavalli, I. Mura, S. Chiaradonna, R. Filippini, S. Poli, and F. Sandrini. Deem: a tool for the dependability modeling and evaluation of multiple phased systems. In *Proc. of the Int. Conference on Dependable Systems and Networks (DSN2000)*, pages 231–236, New York, USA, June 2000.

[48] R. German, C. Kelling, A. Zimmermann, and G. Hommel. Timenet: A toolkit for evaluating non-markovian stochastic petri-nets. *Performance Evaluation*, 24:69–87, 1995.

[49] W. H. Sanders, W. D. Obal II, M. A. Qureshi, and F. K. Widjanarko. The ultrasan modeling environment. *Performance Evaluation*, 24(1):89–115, 1995.

[50] K. K. Goswami, and R. K. Iyer. A simulation-based environment for system level dependability analysis. *IEEE Trans. on Computers*, 46(1):60–74, January 1997.

[51] M. Sridharan, S. Ramasubramanian, and A. K. Somani. Himap: Architecture, features, and hierarchical model specification techniques. In *Lecture*

Notes in Computer Science, number 1469, pages 348–351. R. Puigjaner, N. N. Savino, and B. Serra, 1998.

[52] R. M. L. R. Carmo, L. R. de Carvalho, E. de Souza e Silva, M. C. Diniz, and R. R. R. Muntz. Tangram-ii: A performability modeling environment tool. In *Lecture Notes in Computer Science*, number 1245, pages 6–18. R. Marie, B. Plateau, M. Calzarossa, and G. Rubino, 1997.

[53] R. J. Pooley. The integrated modelling support environment: A new generation of performance modelling tools. In *Proc. of the 5th International Conference in Computer Performance Evaluation: Modelling Techniques and Tools*, pages 1–15, Torino, Italy, February 1991. G. Balbo and G. Serazzi.

[54] R. Fricks, C. Hirel, S. Wells, and K. Trivedi. The development of an integrated modeling environment. In *Proceedings of the World Congress on Systems Simulation (WCSS '97)*, pages 471–476, Singapore, September 1997.

[55] Aad P. A. van Moorsel and Yiqing Huang. Reusable software components for performability tools, and their utilization for web-based con guration tools. In *Proceedings of the 10th International Conference in Computer Performance Evaluation: Modelling Techniques and Tools*, pages 37–50, Palma de Mallorca, Spain, September 1998. R. Puigjaner, N. N. Savino, and B. Serra.

[56] G. Franceschinis, M. Gribaudo, M. Iacono, N. Mazzocca, and V. Vittorini. Drawnet++: Model objects to support performance analysis and simulation of complex systems. volume 2324 of *Lecture Notes in Computer Science*, pages 233–238. Springer Verlag, April 2002.

[57] K. S. Trivedi. Sharpe 2002: symbolic hierarchical automated reliability and performance evaluator. In *IEEE Int. Conference on Dependable Systems and Networks (DSN 2002)*, page 544, Washington, DC, 2002.

[58] G. Ciardo, and A. S. Miner. Smart: Simulation and markovian analyzer for reliability and timing. In *IEEE International Computer Performance and Dependability Symposium (IPDS'96)*, page 60, Urbana-Champaign, IL, USA, September 1996. IEEE Comp. Soc. Press.

[59] F. Bause, P. Buchholz, and P. Kemper. A toolbox for functional and quantitative analysis of deds. In *Lecture Notes in Computer Science*, number 1469, pages 356–359. R. Puigjaner, N. N. Savino, and B. Serra, 1998.

[60] V. S. Adve, R. Bagrodia, J. C. Browne, E. Deelman, A. Dube, E. Houstis, J. Rice, R. Sakellariou, D. Sundaram-Stukel, P. J. Teller, and M. K. Vernon.

Poems: End-to-end performance design of large parallel adaptive computational systems. *IEEE Transactions on Software Engineering, Special Section of invited papers from the WOSP '98 Workshop*, 26(11):1027–1048, November 2000.

[61] D. Daly, D. D. Deavours, J. M. Doyle, P. G. Webster, and W. H. Sanders. Möbius: An extensible tool for performance and dependability modeling. In *11th International Conference, TOOLS 2000*, volume Lecture Notes in Computer Science, pages 332–336, Schaumnurg, IL, 2000. B. R. Haverkort, H. C. Bohnenkamp, and C. U. Smith (Eds.).

[62] A. Christensen. Result specification and model connection in the möbius modeling framework, 2001. (master thesis).

[63] GSM 04.60 version 8.3.0 Release 1999 ETSI. Digital cellular telecommunication system (phase 2+); general packet radio service (gprs); mobile station (ms)- base station system (bss) interface; radio link control/medium access control (rlc/mac) protocol.

[64] Chang-Yu Wang, D. Logothetis, K. S. Trivedi, and I. Viniotis. Transient behavior of atm networks under overloads. In *Proc. of the Fifteenth Annual Joint Conference of the IEEE Computer Societies, Networking the Next Generation (INFOCOM '96)*, volume 3, pages 978–985, March 1996.

[65] S. Porcarelli. *Analysis and Modeling of Dependability and Performability of Telecommunication Systems*. PhD thesis, University of Pisa, Italy, 2003.

[66] I. Mura, and A. Bondavalli. Hierarchical modeling and evaluation of phased-mission systems. *IEEE Transactions on Reliability*, 48(4):360–368, December 1999.

[67] V. Mainkar, and K. S. Trivedi. Sufficient conditions for existence of a fixed point in stochastic reward net-based iterative models. *IEEE Transactions on Software Engineering*, 22(9):640–653, September 1996.

[68] W. K. Grassmann. Finding transient solutions in markovian event systems through randomization. *Numerical Solution of Markov Chains*, pages 357–371, 1991.