Visualization of the Dynamic Analysis of Business Models Based on the Petri Net Formalism

Marianne Ozkan

Department of Computer Science, McGill University, Montreal

July 1993

Thesis submitted to the Faculty of Graduate Studies and Research in partial fulfilment of the requirements of the degree of Master.

Visualizing the Dynamic Analysis of Models Based on Petri Nets

Marianne Ozkan

Department of Computer Science, McGill University, Montreal

July 1993

Thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements of the degree of Master.

Table of Contents

Abstract

Dynamic analysis plays a increasing role in software development as it helps assess model behaviour. Techniques involved in dynamic analysis include graphical simulation and performance analysis. While the former principally addresses the qualitative behaviour of models, the latter produces quantitative performance indices. The resolution methods involved in both analyses require formal modelling techniques. Models based on the Petri net formalism have been extensively used to assess models, principally due to their descriptive power.

In this research, we review the Petri net formalism as well as the resolution methods involved in graphical simulation and performance analysis and introduce generic concepts that enable users to visualize the results of dynamic analysis through a consistent user interface. From the user's point of view, our goal is to facilitate the usage of dynamic analysis and ease the interpretability of its results. This is achieved by hiding complex dynamic analysis resolution methods behind an attractive user interface layer, while providing the user with full control over both the methods' execution and the display of their results. From the designer's point of view, we introduce generic visualization concepts that can be tailored to specific application domains.

In the past year, we developed a toolset called Macrotec [1][36], which integrates the modelling activity and both aspects of the dynamic analysis activity, namely graphical simulation and performance analysis. Macrotec's user interface was designed according to our dynamic analysis visualization concepts. Its development allowed us to validate the genericity of these concepts by tailoring them to the business domain.

1. Macrotec was developed as part of a joint CRIM/DMR Group Inc. project, which is part of the IT MACROSCOPE project.

Our experience shows that dynamic analysis promotes system improvement by enriching the modelling activity and thus, we strongly believe in the benefits of its usage in software development. To promote this, we undertook the rationalization of the results produced by dynamic analysis and the elaboration generic concepts permitting their visualization.

Résumé

L'analyse dynamique joue un rôle accru dans le développement de systèmes car elle permet d'évaluer le comportement de modèles. Parmi les techniques d'analyse dynamique, on trouve la simulation graphique et l'analyse de performance. Tandis que la première a trait principalement au comportement qualitatif de modèles, la seconde fournit des mesures de performance quantitatives. Les méthodes de résolution utilisées dans ces analyses nécessitent une technique formelle de modélisation. Dûs principalement à leur pouvoir descriptif, les modèles basés sur le formalisme des réseaux de Pétri sont fréquemment utilisés afin d'évaluer un système.

Dans cet ouvrage, nous exposons les méthodes de résolution de la simulation graphique et de l'analyse de performance telles qu'appliquées au formalisme des réseaux de Pétri. De plus, nous introduisons des concepts génériques qui permettent la visualisation des résultats d'analyse dynamique selon une interface usager cohérente. Du point de vue de l'usager, nous tentons d'augmenter l'accessibilité de l'analyse dynamique et de faciliter l'interprétation de ses résultats et ce, en camouflant ses méthodes de résolution derrière une interface attrayante, tout en laissant à l'usager le plein contrôl sur l'exécution de l'analyse et l'affichage des résultats. Du point de vue du développeur, nous décrivons des concepts de visualisation généraux pouvant aisément s'appliquer à quelconque domaine d'application.

Au cours de la dernière année, nous avons développé l'outil Macrotec [1] [36] qui intègre à la fois l'activité de modélisation et les deux aspects ci-haut mentionnés de l'analyse dynamique, soit la simulation et l'analyse de performance. L'interface usager de Macrotec concrétise nos concepts de visualization d'analyse dynamique. Son développement nous a

1. Macrotec est issu d'un projet conjoint entre le CRIM et Le Groupe DMR Inc. et fait partie du projet MACROSCOPE FT

permis de valider la généralité de ces concepts puisque Macrotec s'applique essentiellement à l'organization du travail.

Notre expérience démontre que, par le fait d'enrichir la modelisation de systèmes, l'analyse dynamique promeut leur raffinement. Ainsi, nous sommes profondément convaincu de son effet bénéfique lors du développement de systèmes. Dans le but de l'appliquer de facon efficace, nous avons donc entrepris la rationalisation des resultats produits par l'analyse dynamique ainsi que des concepts permettant leur visualisation.

Introduction

Modelling and model analysis have become an integral part of software engineering. More particularly, dynamic analysis, which is the study of the behavioural aspects of systems, is a powerful instrument of system design and refinement. Various formalisms and methodologies have come to support dynamic analysis in system development and evolution. Advances in Petri net theory [34] have made a significant breakthrough in drawing together classical analysis methods and powerful modelling formalisms.

We describe in this work widely-used dynamic analysis methods used on Petri-Net-based models. We focus on two types of analysis, namely simulation and performance analysis. The first one is the direct execution of a system as it is virtually "brought to life". The second demonstrates the quantitative aspect of the system's performance through time. We suggest in both cases visualization techniques, incorporating state-of-the-art graphics and animation techniques, that make use of dynamic analysis methods and present model behaviour as high-level graphical abstractions. Furthermore, we present the design of an integrated environment supporting the tasks of modelling, simulation, and performance analysis where a loose coupling between visualization schemes and analysis models favours reusability and evolution.

Chapters 2 and 3 are dedicated to graphical simulation and performance analysis respectively, describing possible underlying models and visualization techniques. Chapter 4 describes the Macrotec toolset [36] which integrates both aspects of dynamic analysis into a coherent environment targeted at business modelling. Chapter 5 draws a parallel between Macrotec simulation and algorithm animation and proposes extensions that would ultimately enable Macrotec to support visualization of dynamic analysis in the general sense of algorithm animation. Finally, the last chapter surveys ongoing and future work in the dynamic analysis visualization domain.

This research provides the following original contributions to the dynamic analysis visualization domain:

- Generic visualization concepts are described that reflect the quantitative and qualitative temporal behaviour of systems through an intuitive graphical user interface.

- Generic visualization concepts are described that allow for the customization of dynamic analysis resolution methods as well as for the tailoring of the resolution methods' results. These concepts are illustrated through a graphical user interface that augments the transparency of the underlying analysis engine.

- Simulation and performance analysis visualization concepts are clearly distinguished as these two complementary aspects of dynamic analysis achieve different goals through their own specific resolution methods.

- A precise vocabulary describing dynamic analysis results is introduced which promotes a taxonomy for both dynamic analysis findings, and the visualization repertoire addressing each of them.

- Generic dynamic analysis visualization concepts are validated and tailored to the business modelling domain through the Macrotec toolset which supports integrated simulation and performance analysis.

- Finally, graphical simulation is described in terms of general algorithm animation. This approach provides a solid base on which to construct, evaluate, and extend graphical simulation as it promotes synergy with a well-established related discipline.

Chapter 1

Background and Related Work

1.1 - Dynamic Analysis and Business Modelling

1.1.1 - Problem Statement

Dynamic analysis is being successfully applied at improving the performance and reliability of computer systems. This technique aims at decreasing a computer system's overall uncertainty and cost, and this, by facilitating detection and elimination of unanticipated performance and reliability bottlenecks in both its software and hardware components.

The term *dynamic analysis* is used here in a generic way, to include performance assessment, reliability assessment and fault-tolerance analysis [17]. This research focuses on performance assessment, both qualitative and quantitative. Moreover, it concentrates on a particular type of application, namely business modelling.

Business modelling is a technique for specifying and analysing an enterprise's infrastructure. Its overall goal is to identify, design and evaluate value-adding opportunities for business improvement. To this end, business modelling approaches should support the design and analysis of an enterprise's architectural structures and their information technology components as well as the analysis of their dynamic aspects. Moreover, these approaches should encompass the evolution of the architectural structure of business processes [5].

Dynamic aspects of business modelling address the behaviour of systems, people and the organization as a whole as well as the process control and physical resource structures of the business [5]. In the area of business modelling, key performance indicators would therefore include items such as resource utilization, and dynamics of product and information flow.

1.1.2 - Envisioned Solution

In light of the above problem domain, it is clear that a business modelling approach should provide a formalism with precise semantics. This formalism should be highly expressive, yet intuitive to non-specialists. Moreover, to support the dynamic aspects of business modelling, it should be well-suited for representing concurrency, synchronization, communication, and cooperation among model components. A well-defined formalism, together with a visually attractive representation for the resulting models, will enforce a common language among participants and may further communication and training in the enterprise.

Furthermore, such a formalism should support logical and structural assessment, as well as analytical performance evaluation. Specifically, the methodology and its formalism should provide support for the analysis of causal relationships between processes, for the investigation of undesirable system properties such as deadlocks, and for the automatic generation of quantifiable timed performance measures. Also, the methodology should favour prototyping to gain scalable optimization of the models.

The next section describes prominent modelling formalisms and details the one we feel best suited for business modelling.

1.2 - Evolution of Modelling Approaches

1.2.1 - Simulation, Queuing Networks, and Petri Nets

Simulation techniques such as Monte Carlo and analytical models such as Markov chains [50] are useful in describing phenomena of probabilistic nature. They have been widely used for modelling systems, evaluating performances, as well as studying sensitivity to parameter variations. They hold, however, a major drawback in that their usage is restricted to performance specialists. Clearly, higher-level techniques would bridge the gap between clients on one side, concerned with the performance of a real system, and modellers on the other, unfamiliar with the system domain.

These concerns triggered the development of two important abstract formalisms, namely queuing networks and Petri nets. We discuss these formalisms in the rest of this section.

A queuing network is a set of interconnected queues in which customers circulate and possibly arrive from and leave to the outside world [48]. Defined routing probabilities determine the path followed by customers through the network. Many packages make use of the queuing theory, presenting resolution techniques such as simulation and exact or approximate analytical methods. One such tool, AT&T's Performance Analysis Workstation [51] makes use of the visual attractiveness of the queuing formalism to simulate and present performance indices through a graphical user interface.

Petri Nets (PNs)[1] have gained wide acceptance as a powerful modelling tool due to their descriptive power in presence of phenomena such as concurrency and synchronization. PNs, however, do not support temporal specifications, and therefore, no time-related performance measures can be obtained from analysis. Since their original elaboration in the 60's, many extensions of PNs have been proposed in an attempt to associate timing to net elements. E-nets [59], for example, introduce fixed time delays between the enablement of a transition and its firing. Another variation of PNs associates minimum and maximum firing time durations to each transition [53]. In this research, however, we focus on yet another class of Petri Nets, namely the Stochastic Petri Net (SPNs) for reasons which will become clear later. At this point, let us formally define PNs.

Basically, a Petri Net consists of a set of places P, a set of transitions T, a set of directed input I and output O arcs connecting places to transitions and transitions to places respectively, and a marking M. The marking or state of a PN is defined by the number of tokens in each place. The marking is represented by a vector whose *ith* component represents the number of tokens in the net's *ith* place. A PN can be formally described as [49]:

$$PN = (P, T, I, O, M)$$
$$P = \{p_1, p_2, \quad , p_n\}$$
$$T = \{t_1, t_2, \quad , t_m\}$$
$$I \subset P \times T$$
$$O \subset T \times P$$
$$M' = \{m_1', m_2', \quad , m_n'\}$$

where M' is the initial marking of the net.

SPNs are extensions of PNs where random, exponentially distributed firing times are associated to transitions. An important breakthrough in PN theory occurred when Molloy, the father of SPNs, proved SPNs to be isomorphic to Continuous Time Markov Chains (CTMC), SPN markings thus corresponding to states of this particular type of Markov chain [56]. This property implies that a Markovian model can be automatically derived from a SPN and that, conversely, performance indices produced from the resolution of a

1. Readers may refer to Appendix A, "Glossary of Basic Petri Net Terminology".

Markov chain. The performance indices in question may be the average number of tokens in a place, the frequency of firing of a transition, the average delay of a token, etc. The advent of SPNs therefore bridged the gap between graph models and probabilistic models familiar to performance analysts.

Extensions of SPNs soon followed Molloy's initial work They attempt to improve the expressive power of SPNs by introducing new classes of transitions, while retaining the behavioural equivalent to continuous-time stochastic processes. Generalized Stochastic Petri Nets (GSPNs) [50], for example, support timed and immediate transitions. Immediate transitions, by definition, fire in zero time once they are enabled, whereas timed transitions fire after a random, exponentially distributed time. Deterministically Timed Petri Nets (DTPNs) support a mix of exponentially distributed and fixed transition time delays. Fixed time delays may be useful in evaluating the performance of hardware components where transition timings can hardly be imagined to be exponentially distributed random variables.

1.2.2 - Discussion

Queuing networks have become widely used in modelling stochastic processes due to the reduced complexity of their solution at the network's equilibrium state. In effect, the *steady-state solution* (cf. Section 2.2.3, "Execution Options") of most queuing networks can be factored into the product of the steady-state solutions of individual queues [48].

The major drawback of queuing networks is their lack of expressiveness in modelling synchronization, blocking and splitting customers. Moreover, these phenomena destroy the product form property of the queuing network, rendering necessary the conversion of the simplest queuing model into a continuous time Markov chain.

The interest in Petri nets can be justified by their descriptive power. However, they must be converted into Markovian models which suffer from exponential state-space explosion [1]. In case of intractable solutions, performance results must be produced by simulation techniques at high processing cost, due to long programs and large amounts of input data [1].

SPNs in particular are restricted by the limited-size systems they can graphically represent and the complexity of their analysis. This last factor is due to the fact that activities in the same model may execute rapidly compared to others which are critically time consuming. Furthermore, some activities may even be inserted for some pure logic aspects, making it difficult to define their timing and resulting in a system of equations which is difficult to solve [48]. GSPNs and DTPNs have reduced resolution and representation complexity due to the introduction of new classes of timed transitions.

From the above remarks, we deduce that Petri nets, and in particular GSPNs and DTPNs, are the most adequate formalism for business modelling, performance evaluation, and sensitivity analysis. This research therefore concentrates on Petri nets.

1.3 - Support for Modelling Approaches

1.3.1 - Existing Packages

Dozens of packages have been reported to make use of Petri Net formalisms in the

modelling and analysis of real systems. Feldbrugge's most recent overview [25] of Petri net tools for example, lists 23 such tools, each offering diverse functionalities. In what follows, we look at three tools which we feel well-suited for business modelling, namely Design/CPN, GreatSPN and SPNP.

Design/CPN by the Meta Software Corporation [54] makes use of Coloured Petri Nets (CPNs). CPNs allow tokens to be of different types and to take on values specific to those types. With CPNs, the size of models can be kept much smaller than with classical Petri nets [54]. Design/CPN supports hierarchical models and temporal specification. It has an open architecture and may therefore be expanded to support other types of Petri nets, new analysis tools, or to interface with other modelling techniques. Design/CPN contains a powerful graphical editor which performs semantic checks. It supports graphical simulation as well as the detection of potentially unsafe situations. Design/CPN enables users to program code segments which are executed when the specified transition fires. Many predefined functions are available for computing performance-related measures by means of these code segments, however analytical resolution methods are not supported.

GreatSPN [15], a tool developed at the University of Turin, is based on Generalized Stochastic Petri Nets. Although GSPNs support two types of timed transitions, immediate and exponentially distributed, GreatSPN offers solution algorithms supporting deterministic timed transitions as well. The package includes a graphical editor for net specification. It permits validation of the qualitative behaviour of the net by means of invariant computations, detection of structural properties, and graphical simulation. GreatSPN supports exact analytical solutions as well as Monte Carlo simulation to derive performance results. Results can be computed at steady-state (equilibrium) or at specific time points according to the Markovian transform solution. Results may be displayed on the edited network under histogram representation.

The Stochastic Petri Net Package (SPNP) [16], developed at Duke University, is a

tool for the solution of Stochastic Reward Nets (SRN), which are extensions of stochastic Petri nets. The steady-state and transient solution algorithms make use of the Markov reward chain, a Markov chain in which rewards have been assigned to states and transitions. SPNP also allows resolution by simulation. In addition to the standard set of measures already available, rewards allow associating custom measures to markings of the SRN. Logical assertion of the net can be performed by associating logical assertions to specific markings. SPNP also supports sensitivity analysis.

1.3.2 - Towards the Solution

In developing a business modelling approach supported by a convenient performance evaluation package, three alternatives command attention. The first one is simply selecting an existing general-purpose package such as the ones described in the preceding section. This approach, although advantageous in the short term, implies blurring out application specifics, with all the disadvantages we know. The second alternative is to develop an application-specific - business-oriented in our case - formalism while reusing resolution models of existing tools. In this case, a dedicated user interface must be developed, accompanied by a transformation component in order to translate the formalism into the formalism of the package used. Benefits of a direct mapping from the model formalism to the business "jargon" must be carefully weighed against the overhead resulting from the additional transformation component. Moreover, this approach implies another challenge, that of converting general performance indices into relevant business figures. Lastly, the third alternative is to develop a complete methodology tailored to the particular application domain with a formalism and resolution models specific to that domain.

With any of these alternatives, the target system should exhibit three important fea-

tures. The first is a user-friendly user interface that allows both edition and automatic validation of the network as well as the graphical edition of performance results. The second is the support for structural analysis of the model by means of graphical simulation and possibly invariant computation. Finally, the third feature is the support of an analytical network resolution method which dispenses users from the overhead of simulation. Furthermore, these features should be fully integrated at the interface level, allowing access to the full functionality of the modelling and analysis components in a consistent and transparent way.

1.4 - Research Content

Chapter 2 addresses the first feature mentioned above by describing a Petri net editor allowing for both network specification and validation. This editor constitutes the basis for the visualization concepts subsequently described. Chapter 2 pursues with an extensive discussion of the second feature mentioned above, namely graphical simulation, whereas chapter 3 addresses the third feature, performance analysis. Note that chapters 2 and 3 concentrate on the dynamic analysis visualisation aspects and do not intend to describe a full modelling and analysis tool.

Both chapter 2 and 3 describe generic visualization concepts applied to simulation and performance analysis and which have been elaborated with two main concerns in mind, namely ease of tailoring to various application domains and ease of integration into one single edition and dynamic analysis tool. These concepts promote customization of simulation and performance analysis resolution methods, as well as the interpretation of their results. Furthermore, they are illustrated through a state-of-the-art graphical user interface which combines the highest abstraction-level features found in various existing tools considered in this study, tools such as [54], [14], and [26]. This user interface is augmented with original concepts, identified as such in the research, which we feel furthers the interpretability

of dynamic analysis.

Chapter 4 describes the Macrotec[1] toolset and in so doing, meets three specific purposes. The first one is the validation of the general applicability intended by the generic visualization concepts of the preceding two chapters. Through Macrotec, these generic concepts are sucessfully tailored to the business domain, the scope of the Macrotec project.

The second purpose of chapter 4 is the verification of the feasability of integrating graphical simulation and performance analysis into one single system, under a consistent user interface. This has effectively been achieved in Macrotec, the resulting user interface being described in chapter 4. By means of such extensible integration, Macrotec surpasses in some respect the partial functionality offered by various tools considered in this research such as RDD100, Eval [73], MetaDesign, Design/CPN [54], and Voltaire [60].

Finally, chapter 4 illustrates the design trade-offs involved in developing a business modelling approach supported by a performance evaluation package, as mentioned in the preceding section. In fact, at an early project stage, we considered the three alternatives described above. Since we developed a business-specific methodology, we did not pursue the first alternative. For lack of resources, we discarded the third alternative. We finally settled for the middleground, alternative two, and undertook to develop our own original formalism, yet using an existing performance analysis package.

Finally, chapter 5 takes on a new approach to graphical simulation as it describes it in terms of algorithm animation. This chapter describes Macrotec's graphical simulation design in terms of existing algorithm animation packages such as [46] and addresses, according to this point of view, the extensibility of Macrotec in order to support to the very detail the visualization concepts described in chapter 3.

1. Macrotec has been elaborated as part of the MACroscope Architecture project (MACA), a joint project between CRIM and the DMR Group Inc.

To conclude, chapter 6 outlines the present and future work pursued in the field of dynamic analysis visualization and the last part of this research emphasizes its major findings.

Chapter 2

Graphical Simulation

2.1 - Definition and Purpose

Graphical simulation, also known as "the token game", is the visual execution of the network. It promotes comprehensive structural and behavioural assessment of a network by visualization of the causal relationships between its components. Thus, it enables one to answer questions such as: Which tokens are required for a transition firing? In which order do transitions occur? What outputs are obtained? What are the necessary intermediate transitions to be performed? What inputs are required for a specified output?

In this chapter, we first discuss the back-end of simulation. We define the required inputs to the simulation engine, propose a simulation algorithm, and describe its expected outputs. We then turn to the front-end of graphical simulation as we propose a basic graphical Petri net editor and particularly focus our attention on a set of graphics primitives used to visualize net behaviour and simulation results.

Our discussion is based on a class of Petri nets with multiplicity, deterministic timed transitions, and immediate transitions. The visualization concepts introduced, however, may as well be applied to Coloured Petri Nets, Generalized Petri Nets, or any other class of PNs. Taking into account Petri net elements such as guards, colorsets, inhibitor arcs, etc. would indeed refine the modelling approach, but would considerably complicate the resulting simulation engine. This would, however, add nothing to the discussion of visualization, the main focus of this chapter.

2.2 - Simulation Back-End

2.2.1 - Input Parameters

The inputs of a graphical simulation engine are the following[1]:

- the set of places, transitions, and arcs
- the initial marking of the network
- the firing time of transitions
- the weight of transitions
- the multiplicity of arcs

Figure 1 shows a network ready for simulation. The network consists of four places p1, p2, p3, and p4, one immediate transition t1, and three deterministic transitions t2, t3, t4 with fixed firing times of 2, 2, and 4 time units respectively. T2 has a weight of 4 and t3, a weight of 6. The multiplicity of t2's output arc is 2.

2.2.2 - Execution Model

The execution of a network from an initial marking is called *forward simulation*. In a timed forward simulation, a global clock keeps track of the simulated time units elapsed since the beginning of the simulation. The global clock starts at time zero and is incremented, each time a transition fires, by the firing time of that transition.

1 Readers unfamiliar with basic Petri net terminology may refer to Appendix A.

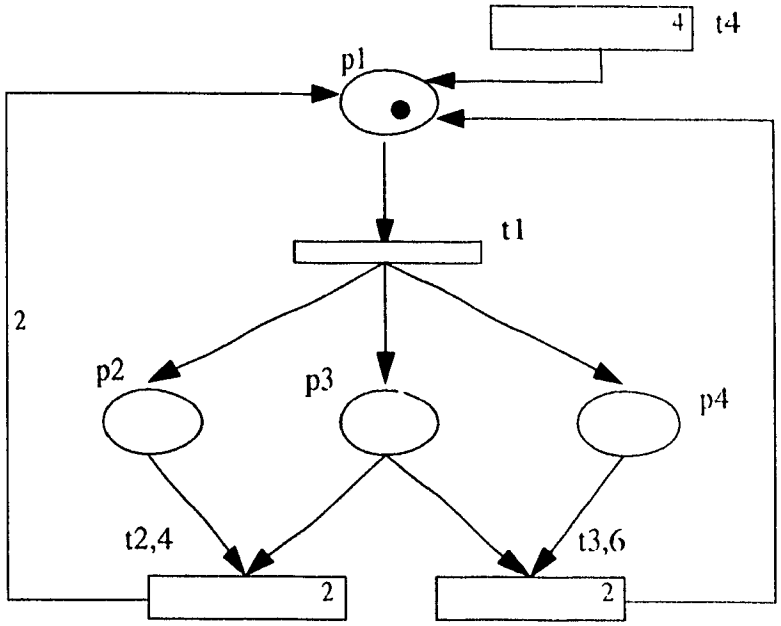

Figure 1 - A Petri net ready to be simulated

The general algorithm of a timed simulation is the following:

While Enable_transitions do {
Adjust_global_time
Resolve_conflicts
Fire_transitions
}

Enable_transitions An inactive transition is enabled if its input places each contain a number of tokens equal to or greater than the multiplicity of the arc joining that place to the transition. A transition remains enabled for a number of time units equal to its firing time.

Adjust_global_time Many transitions may be enabled simultaneously. The global

simulated clock is incremented by the shortest firing time of all enabled transitions.

Resolve_conflicts Enabled transitions which have completed their execution become firable. Conflicts between firable transitions are resolved according to the weight proportion of the conflicting transitions.

Fire_transitions When a transition is fired, m_i tokens are removed from each input place of the transition and m_o tokens are added to each output place of the transition, m_i and m_o being the multiplicities of the input and output arcs, respectively.

2.2.3 - Execution Options

The basic simulation algorithm described in the previous section may be refined to account for user-specific goals. We propose here, different facets of simulation which can be integrated into an interactively-customizable simulation algorithm.

Timed vs untimed simulations. A first aspect is to allow both timed and untimed simulations. Timed simulations imply a clear distinction between the enablement of a transition and its firing, the former occurring at simulated time $t_i < t_j$, t_j being the simulated time at which the latter takes place. Clearly, $t_j - t_i$ is the firing time of the transition. Untimed simulations execute transitions without delay, therefore, as if they had immediate firing time.

Manual vs automatic conflict resolution. Conflict resolution may be manual or automatic. Automatic conflict resolution is performed without user intervention, by means of transition weights, as we have already seen. Conflicts may also be resolved interactively by selecting the transition for firing from the conflicting set.

Automatic vs step-by-step execution. The simulation engine may be left to run unattended in what is called *automatic execution*, or it may be forced to pause and restarted again upon user request, enabling in this way such manipulations as manual conflict resolution and editing. The latter type of execution is known as *step-by-step simulation*, where a step is the interval between consecutive pauses. Intuitively, a step corresponds to one iteration of the central while loop of the simulation algorithm. Logically, a step corresponds to the time lapse between consecutive transition firings, since between these firing events, "nothing" is considered to happen at the behavioural net level, and time simply skips to the next transition firing.

Automatic execution requires specifying the number of simulation steps to execute as well as the real time lapse between each step. Both execution modes, automatic or manual, should terminate and prompt the user when the network has reached a deadlock (i.e no transitions can be enabled).

Cost analysis. Simulation may be a useful tool for cost analysis. A cost may be associated to transitions. The simulation engine adds the cost of a fired transition to the overall cost of the process. The transition's contribution to the overall process can be calculated as a percentage of the total cost and presented as a graph.

Backward simulation. Backward simulation is used to find if a certain marking is possible from specified input places and, if it is, find the different paths leading to this marking. Backward simulation helps answer questions such as: Which alternatives produce a specified output? Which alternative has lowest delay and lowest cost?

2.2.4 Desired Outputs

Simulation promotes comprehensive structural and behavioural assessment of a network. Specifically, simulation helps realize three goals which can be associated to concrete simulation results.

1- Simulation enables users to understand, assess, and debug:

the sequence of transitions
- the requirements of a transition
- the outputs of a transition
- the role of a transition
- the role of the model as a whole
- the dynamics of the model (the possible sequences and parallelism)

The simulation results needed to achieve this goal are of qualitative nature:

- which and when do transitions become firable and enabled
- which markings permit a transition's enablement and which sequence of firings leads to them
- which places and how many tokens are involved in transition firings
- which paths lead to a particular firing or marking

2- Simulation allows for model evaluation in terms of performance. It helps to identify model characteristics which are not evident during design. It enables the identification of deadlock and bottleneck conditions.

To realize this goal, quantitative results are required such as:

- the average number of tokens residing in a place during a simulation time interval
- the simulated time required for a firing or a marking to occur
- the critical path, time-wise, to achieve a specific marking
- the average percentage simulation time spent executing a transition during a simulation time interval
- the average throughput of a transition during a simulation time interval (i.e. the average number of tokens flowing through the transition per simulation time unit)

As we will see in the following chapter, these and other results can be achieved through performance analysis. Note however, that the results computed during performance analysis are derived analytically, whereas in graphical simulation, they are updated incrementally at each simulation step.

3- Simulation helps develop new models.

To promote this, simulation furnishes ways of comparing two models both in terms of functionality and performance, using the qualitative and quantitative results of 1 and 2

2.3 - Simulation Front-End

2.3.1 - Network Edition

We now tackle the front-end of graphical simulation, as we lay out the basic editing functions of the user interface. We propose a generic Petri net editor and suggest editing concepts we feel important and note-worthy. This section provides the "look and feel" of the network, from the user's point of view, and provides the basis for the visualization of simulation results, a topic handled in the following section.

Basically, the editor provides a four-region window as illustrated in figure 2. The first region is the drawing area, the second is a tool palette, the third is a menu bar, and finally, the fourth is an information region which provides feedback to the user. The palette is constituted of icons representing the model's basic building blocks. The top-most icon, when selected, allows users to enter text, for example, the labels of places and transitions. The other icons represent the places, the transitions, timed and immediate, and the arcs used to draw the structural component of the net. The menu bar offers standard file operations such as *New*, *Open*, *Save*, and *Save As*, as well as standard editing operations such as *Undo*, *Copy*, *Cut*, and *Paste*. Moreover, all simulation commands are accessed through the *Simulation* command of the menu bar.

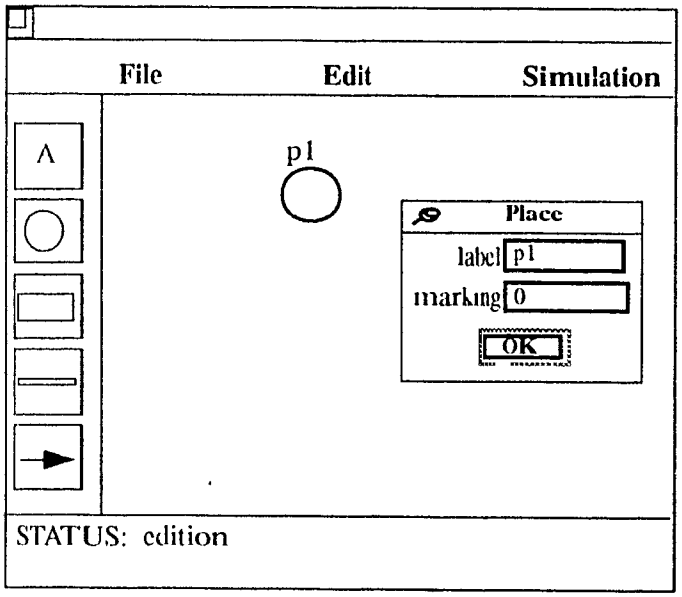

Figure 2 - Specification of a place's label and initial marking

The editor provides different modes of network specification, depending on the user's familiarity with the editor. An example of such mode is the attribute specification described hereafter.

The basic building blocks of the models are characterized by diverse *attributes*. Timed transitions are characterized, for example, by a firing time, arcs are characterized by their multiplicity, and places by the number of tokens they initially contain. We name objects characterized in this way the *owners* of their respective attributes. The editor supports two modes of attribute specification. Experienced users may type in attribute values in the drawing area, since attributes are textual objects that, upon creation of their owner, are displayed and given default values. Novices, on the other hand, may specify attribute values through the *Attribute* menu, which, for each potential owner, displays a dialogue box prompting the user for attribute values. The dialogue box shown in figure 2, for example, is used to specify the attribute of a place object.

The manipulation of graphical objects is semantic-dependent. For example, dragging an owner from position (x,y) to position (x',y') will automatically redisplay its attributes from their relative position to (x,y) to the same relative position to (x',y')

Semantic correctness is conserved throughout model edition. Arcs, for example, cannot connect two places or two transitions. Constraints between graphical objects are enforced.

Displayed objects belong to one of two classes. The first class is composed of objects together with their attributes which are relevant to the simulation engine. The second class is composed of pure graphical objects. These objects are the standard objects one usually finds in drawing applications. They enable users to annotate the net in order to increase its understandability.

Attribute display may easily overload a window, as network complexity increases. Furthermore, pure graphical objects may possibly lead to visual confusion. The editor therefore allows for selective display of information (attributes and pure graphical objects)

through simple *Show* and *Hide* commands.

2.3.2 - Simulation Results

2.3.2.1 - General Considerations

The results we expect from a simulation were discussed in section 2.2.4, "Desired Outputs". We now concentrate on their presentation to the user or, in other words, on visualizing simulation. We first introduce the basic graphical primitives used to display simulation results. Then, we present a scenario where snapshots are used to illustrate the integration of these primitives.

A transition may be in one of three states - inactive, enabled or firable. Furthermore, firable transitions may be conflicting. Accordingly, the simulation user interface provides graphical distinction between these states and depicts conflicting situations. The time required for a transition to become firable after its enablement is also made available to the user. The user can display this figure as a real number, as continuous qualitative feedback, or as both. Continuous qualitative feedback, as described in details in the next section, is applied to graphical simulation in an original way.

Upon transition firing, tokens removed from the transition's input place are shown travelling along the path dictated by the input arc, so as to show they are "absorbed" or "used" by the fired transition. In the same way, output tokens are shown as travelling along the output arc to their assigned output place.

Smoothing out graphical transitions by way of numerous intermediate displays is essential in rendering a sense of visual continuity which increases the overall understanding of model behaviour. For example, continuous token movement helps to distinguish which tokens are used and created by transitions, and is particularly useful in those cases where many firing transitions share input places. There is, however, a performance trade-off between the frequency of redrawing and the speed of simulation. Smooth transitions necessitate numerous displays and may run slowly, due to limited computing power. On the other hand, fewer displays produce a quick animation but may also produce abrupt changes which dazzle inexperienced users

The simulation user interface supports two simulation refresh modes, one showing continuous movement and the other showing only the origin and destination of movement. Users may alternate between these modes during simulation. Moreover, the speed of simulation or real time between subsequent displays, is adjustable via a scrollbar.

2.3.2.2 - Scenario

A scenario of step-by-step simulation is described hereafter. The scenario is illustrated with figures 3 to 7.

Different quantitative data may be calculated as simulation proceeds. Before launching the simulation engine, users may specify which data to compute through a standard dialogue box such as that shown in Figure 3. Figure 3 shows examples of different statistics such as the throughput and the percentage utilization time of transitions as well as the percentage time a place is empty and its number of tokens. These statistics may be computed as an *average* between the *start* and *stop* time points and for specific *time points* with a constant *increment* between the start and stop points.

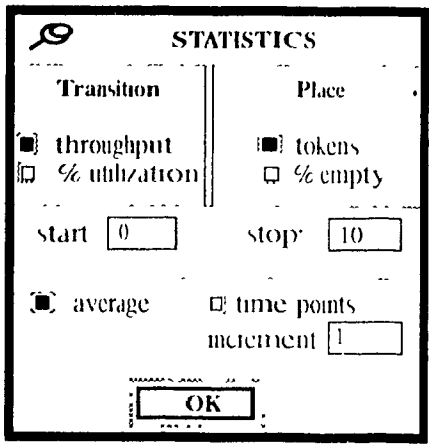

Figure 3 - A dialogue box presenting possible statistics to be computed during simulation

The user launches simulation on the network as shown in Figure 4. A control box immediately pop-ups showing the simulated global clock and the number of steps executed. The control box enables users to take one simulation step forward (*PLAY*) and backward (*REWIND*). Users may cancel the simulation and restart it from the last network state. They may also restart the simulation from the network's initial state. The *REWIND* button of the simulation control box, which accomplishes the step backward, is an original concept which proves useful in highly concurrent networks, where action firing is rather scattered on the screen. It also serves to redirect a path when a conflicting action has been chosen to fire. Since multiple rewinds are possible, the history of simulation must be conserved. This can easily be achieved by storing into memory the history of markings.

In figure 4, immediate transition t1 is firable since it contains enough entities in its input place. This is shown by the thickness of its rectangle's pen pattern. Transition t4 is enabled and will become firable at global time 4 as shown by a full slide bar inside the transition. The slide bar decreases continuously, showing the time remaining until the transition becomes firable. The slide bar can be understood with the thermometer metaphor. The black portion of the bar represents mercury on a horizontal thermometer belong-

ing to a transition that fires at "temperature zero". When the *PLAY* button is clicked, the entity in p1 will be removed and one entity will be created in each of p2, p3, and p4.

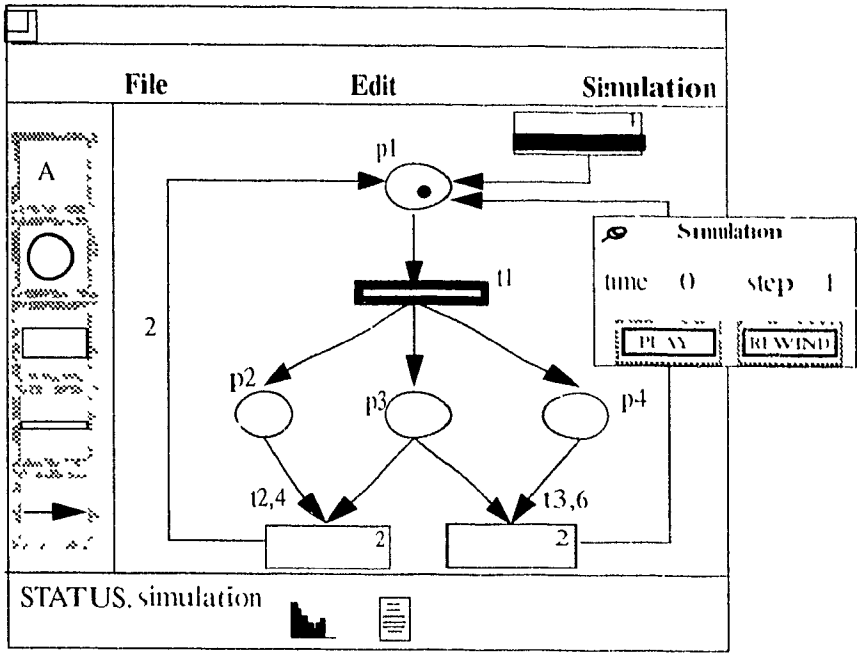

Figure 4 - Simulation snapshot at step 1

Figure 5 shows the display at step 2, therefore after the *PLAY* button is clicked. Transitions t2 and t3 are conflicting since only one token is present in their common input place p3 and their firing times are equal. Conflicts can be made apparent by flashing the conflicting transitions, for example, or by using a different colour to highlight them. We also see that t4 has completed half its execution as shown by a half-full slide bar.

Figure 6 shows two tokens in p1 at simulation step 3. We assume that t2 fired since its output arc has a multiplicity of 2. The user may have selected explicitly t2 for firing, or the simulation engine may have automatically selected t2 for firing according to the weight ratios of t2 and t3. T1 is firable since it is immediate and has precedence over timed transitions.

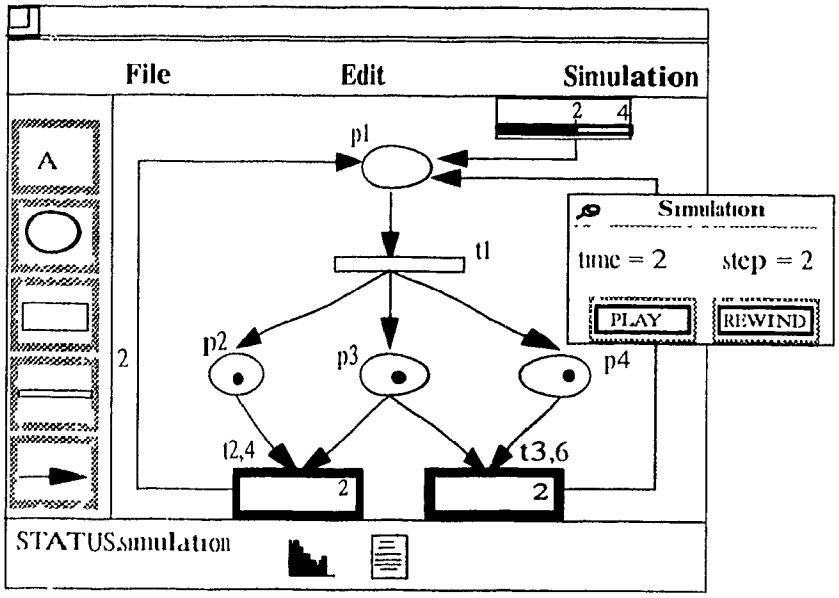

Figure 5 - Simulation snapshot at step 2

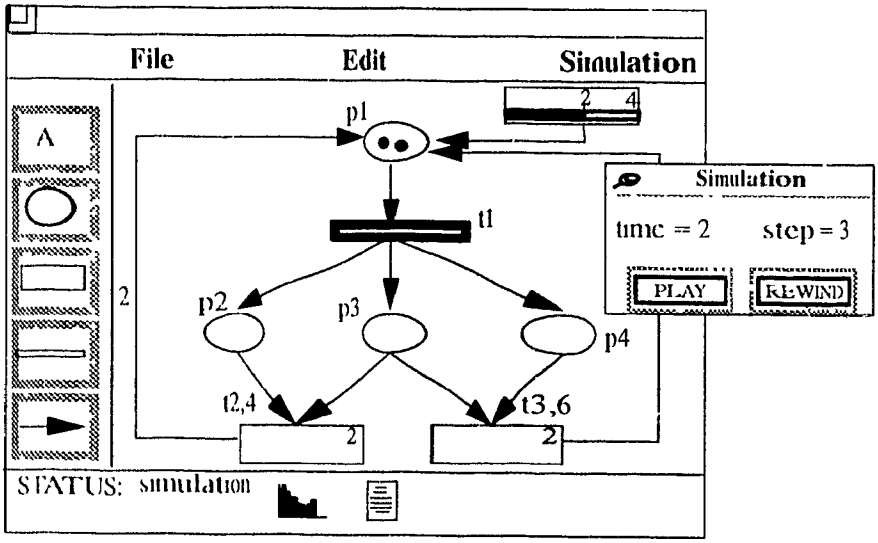

Figure 6 - Simulation snapshot at step 3

Splitting up steps into substeps would allow finer-grained graphical simulation, e.g. smooth token movement and smooth thermometer decrease. For example, figure 7a illus-

trates the network after t1 has fired, as it shows a snapshot of the tokens travelling in continuous fashion to t1 's output places. Figure 7b, shows a snapshot, taken at a later real time than the preceding one, showing the continuous decrease of t4's thermometer as the transition executes. t2 and t3 are enabled, however, they are not yet firable at simulation time 1, as shown in figure 7b.

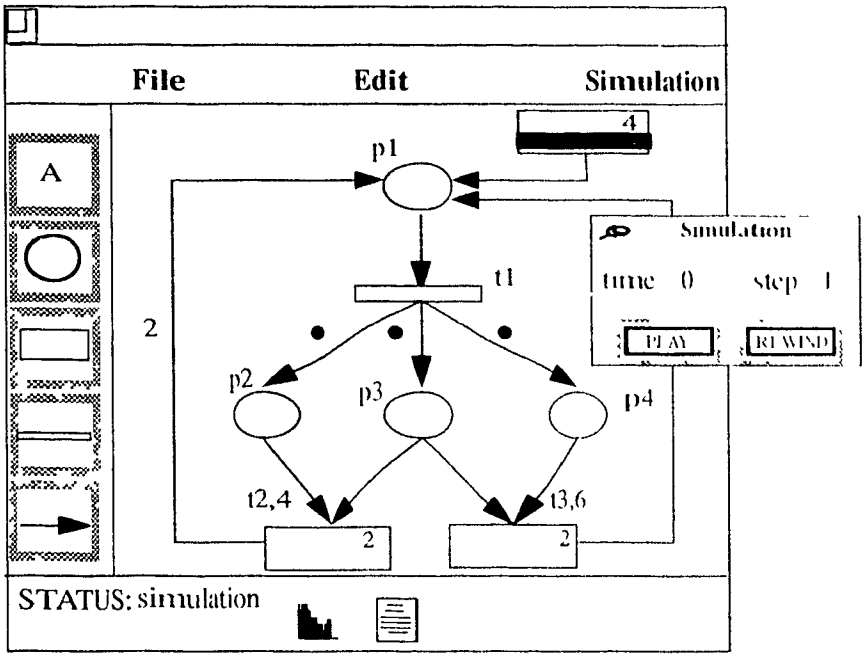

Figure 7a - Token movement between step 1 and step 2

In order to free window space, the user has the option to selectively display information. For instance, the slide bar or labels of places and transitions may be hidden. Another example is the forementioned icons in the status region which expand into files containing statistics the user has chosen to compute during the simulation. The user may specify which information to display whenever the simulation engine pauses. The left hand icon of the status region represents a result file containing tables and graphs. An example shown in figure 8 contains a table of the average throughput of transitions and a line graph of the average throughput versus time. The right-hand icon of the status region represents the simulation trace where, for each step, the simulated time, the firable transitions, and the marking are specified.

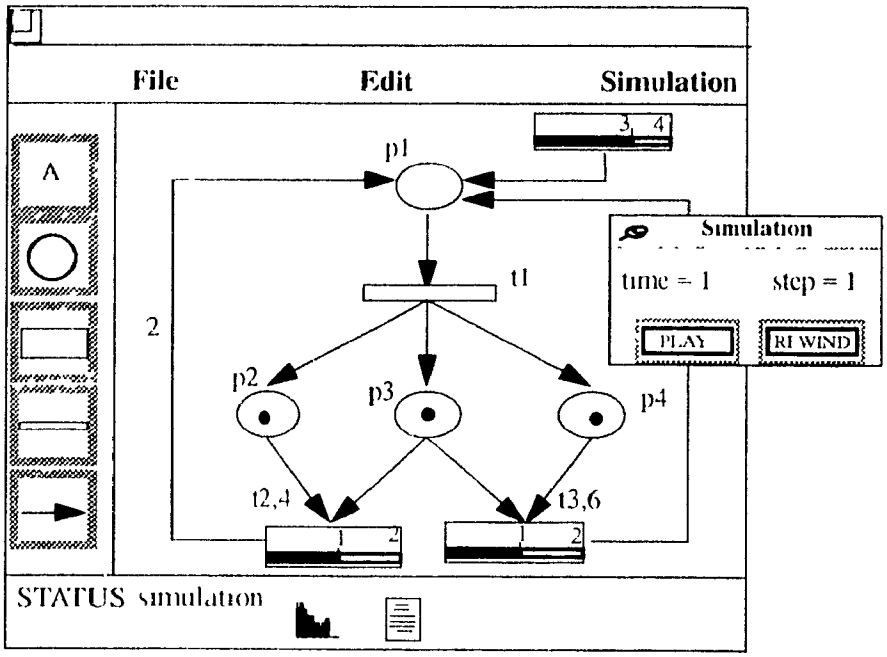

Figure 7b - Thermometer decrease

Average Throughput
t1 - 0.20
t2 - 0.15
t3 - 0.05
t4 - 0.025

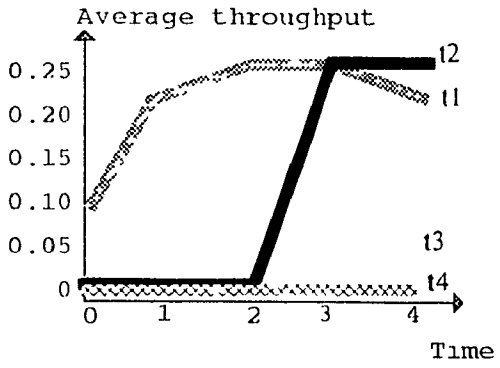

Figure 8 - Simulation results - average throughput of transitions

Finally, the user may split the drawing region into two sub-regions, each containing a model. Graphical simulation can be performed simultaneously, for example on alternative models of the same system. The user may control the simulation through the same dialogue box as the one shown in figures 4 to 7b. Comparative simulation can be described

with the same algorithm we saw at section 2.2.2 "Execution Model", control boxes being available for each model compared.

Chapter 3

Performance Analysis

3.1- Goals

The goal of performance analysis is to obtain quantitative results about a model's behaviour in order to understand and assess its dynamic characteristics. Examples of such results are the waiting time for transitions, the throughput of the model, the bottlenecks, and resource utilization. Performance analysis indicates potential problems in the model and favours its improvement, as it can be conducted on various scenarios. It helps users evaluate the impact of change on overall system performance.

This chapter undertakes the second portion of the dynamic behaviour of Petri nets by looking at performance analysis. In this chapter, we first describe the required inputs to performance analysis, review the resolution methods for Generalized Stochastic Petri Nets (GSPNs), and present the expected performance indices resulting from the analysis. The second part of the chapter addresses the front-end of performance analysis by proposing a generic graphical user interface specific to performance results definition and display.

3.2 - Performance Analysis Back-End

3.2.1 - Required Inputs

The inputs of the performance analysis engine are the following[1]:

- the places, transitions, arcs, and initial marking of the net
- the firing time of transitions

1 Readers who are unfamiliar with Petri net terminology may refer to Appendix A

- the weight of conflicting transitions

3.2.2 - Execution Model

Two types of performance analysis may be conducted. The first is conducted according to a simulation method, and the second according to an analytical solution method. The Monte Carlo simulation [14], for example, uses a pseudo-random number generator to implement the stochastic transition timing. A sequence of discrete events separated by time intervals is simulated and performance estimates can be obtained by gathering statistics from the execution of the model.

In this work, we focus our attention on analytical solution methods based on Generalized Stochastic Petri Nets (GSPNs). This particular class of Petri nets has been widely used in the construction of stochastic models of discrete event systems, due to the memoryless property of the exponential distribution of the firing times, making the GSPN isomorphic to continuous-time Markov chains [48]. Particularly, we are interested in GSPNs which have characteristics that reduce the number of reachable markings and thus the complexity of the solution such as inhibitor arcs and priorities [50].

The associated stochastic model can be derived from the *reachability set* of a GSPN in order to obtain performance indices. In the rest of this section, we formally define GSPNs and describe a firing rule used to generate the reachability set and graph (these are defined below). Solution methods for the derivation of the stochastic model and its interpretation may be found in [49].

A GSPN with priorities and inhibitor arcs is defined as follows [50]:

GSPN = (P, T, Π, I, O, H, W, M_0) where

- (P, T, I, O,M_0) is the underlying basic PN (cf. Chapter 1),

- H is a set of inhibitor arcs,

-Π is an assignment of priorities to transitions, which associates lowest priority (0) with timed transitions and higher priorities (>= 1) with immediate transitions,

- W = (w1, w2,...,wn) is an array whose *ith* entry is:

- the parameter of the negative exponential probability density function of the transition firing delay, if t_i is a timed transition (i.e. the rate of the timed transition)

- a weight used for the computation of firing probabilities of immediate transitions, if t_i is an immediate transition.

In a GSPN with a given initial marking M', the *reachability set* is defined as the set of all markings that can be "reached" from M' by means of a sequence of transition firings [49]. The *reachability graph* is a directed graph whose nodes represent the markings, and arcs are labelled with the transition whose firing produces the marking change. The reachability set and graph are constructed according to a firing policy, of which we describe an example below.

A *vanishing marking* is a marking that enables at least one immediate transition [48]. When such a marking is entered, the weights of the enabled immediate transitions are used to probabilistically select the transition to fire. A *tangible marking* is a marking that enables timed transitions only [48]. When such a marking is entered, the rates of the timed transitions are used to probabilistically select one timed transition to fire. Each timed transition is associated to a timer that is set to a sampled firing delay instance from its probability distribution, when the transition becomes enabled. All timers of enabled timed transitions are then decreased at the same speed until one of them becomes zero. At this

point, the transition whose timer equals zero fires. Every time a new marking is entered, timers are reset to new values sampled from their transition's probability distribution. It is, in effect, inessential to keep the values of timers due to the memoryless property of the exponential distribution [50].

The firing of a transition is considered an atomic operation where tokens are removed from the input places of the fired transition and added to its output places in one indivisible step.

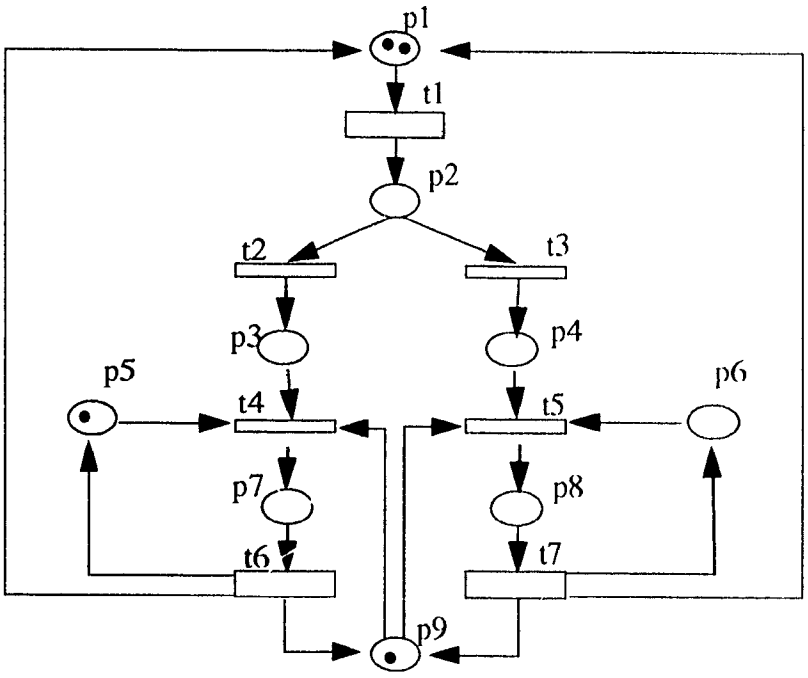

Figure 1 - GSPN model of a two-processor system [50]

An example PN, drawn from [50], and its corresponding reachability set and graph are illustrated in Figures 1 and 2. In this example, timed transitions are assumed to have be random variables with negative exponential distributions.

	p1	p2	p3	p4	p5	p6	p7	p8	p9
M0	2	0	0	0	1	1	0	0	1
M1	1	1	0	0	1	1	0	0	1
M2	1	0	0	1	1	1	0	0	1
M3	1	0	0	0	1	0	0	1	0
M4	1	0	1	0	1	1	0	0	1
M5	1	0	0	0	0	1	1	0	0
M6	0	1	0	0	1	0	0	1	0
M7	0	0	0	1	1	0	0	1	0
M8	0	0	1	0	1	0	0	1	0
M9	0	1	0	0	0	1	1	0	0
M10	0	0	0	1	0	1	1	0	0
M11	0	0	1	0	0	1	1	0	0

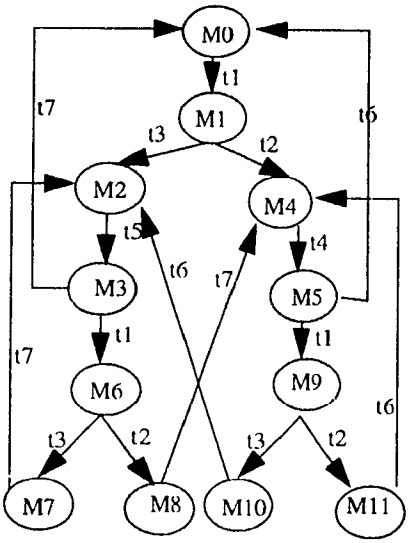

Figure 2 - Reachability set and graph of the GSPN in Figure 1 [50]

3.2.3 - Execution Options

The Markov chain may be solved at equilibrium conditions or at any arbitrary time instant t [48]. We refer to a resolution of the former type as a *steady-state solution* and to the latter as a *transient solution*. In the transient solution, performance indices are calculated at user-specified time points. Both solutions involve the resolution of a set of linear equations.

The steady-state solution may use a standard sparse matrix computation algorithm, adapted to the solution of a set of linearly dependent equations augmented with the proba-

bility normalization condition, as proposed in [14]. In case of matrices larger than 1023x1023, the Gauss-Seidel iterative method is used [29].

The transient solution method may be accomplished by a matrix exponentiation algorithm, where initial estimates of the optimal integration step are dynamically adjusted in order to keep it as large as possible while keeping control over round-off errors during vector additions. This resolution algorithm is proposed in [14].

3.2.4 - Results

The reachability graph permits the computation of the steady-state probabilities of the markings. For a live and bounded SPN [48], the steady-state probabilities of the reachable markings are used to obtain the steady-state probability of an action being enabled. From this result, the token probability density function of each place can be obtained. This result represents the steady-state probability of having a number of tokens in a place.

From the steady-state distribution over reachable markings, the following aggregate steady-state performance parameters are commonly and easily computed [47]

- The expected number of tokens in a place can be computed from the token probability density function.
- The average throughput rate of a transition. This result measures the flow of tokens through a transition and is obtained by the product of the transition's firing rate and its probability of being enabled.
- The token utilization or percentage of time a token is used in the model
- The average delay of a token in traversing a subnet can be derived using Little's for

mula [44]:

$$E(T) = E(N) / E(Y)$$

where E(T) is the average delay, E(N) is the average number of tokens in the process of traversing a subnet, and E(Y) is the average input (or output rate) into (or out of) the subnet.

- The probability that an event occurs. This measure is obtained by adding the probabilities of all markings in which the condition corresponding to the event holds TRUE. The event of interest must be specified by the designer.

- The throughput, which is the number of output results generated by the model during a specific time period.

- The response time or total time taken to get an output result.

Other performance indices which are more difficult to compute are the distribution of the delay incurred by a token in traversing a subnet or in completing a cycle through the net [47].

Despite the nature of the indices generated by analysis, they must be interpreted in order to grasp the dynamic behaviour and the impact of changes on the model. For example, a token utilization of close to 100% indicates the presence of a potential bottleneck. If the token represents a resource, the user may attempt to increase the number of resources or decrease the time delay of the activity responsible for treating this resource if its average throughput rate is considered low. Depending on the network's semantic, users may fine-tune the model until its behaviour meets the performance requirements. The users' ability to judge and interpret performance indices as well as their comprehension of the model is crucial to the efficient usage of performance analysis.

Note that the above performance indices may, in principle, be computed for transient solutions as well.

3.3 - Performance Analysis Front-End

The present section proposes a generic user interface allowing for performance results definition and display. Note that the user interface described here specifically addresses performance analysis component of a general modelling and analysis tool. This user interface is based on an original classification of performance results. The basic principles of the model editor, on which the visualization of results is based, can be found in section 2.3.1, "Network Edition" of the preceding chapter.

We describe the basic concepts related to the presentation of performance analysis results and illustrate the discussion with a suite of figures. We distinguish between three different types of performance results characterized by the network component to which they apply. These results are attributed to

- individual basic network components
- groups of similar basic components
- network states

3.3.1 - Results on Basic Network Components

Certain performance indices can be attributed to specific basic components of the network. For example, the expected number of tokens in a place is undoubtedly an attribute

of the object type Place (we use the words component and object interchangeably). In the same way, the average throughput rate is an attribute of the object type Transition.

We restrain the set of indices related to basic components to the following:

For each place, the indices are:
- the average number of tokens
- the probability that it contains at least one token

For each transition, the indices are:
- the probability of enablement
- the average throughput rate

In case of steady-state solutions, performance results are unidimensional, that is to say, to one object corresponds one real number. In the case of transient solutions, the results are two-dimensional as the time dimension comes into play. The best representation is therefore a graph such as a line graph, a histogram, or a table.

Figure 3 shows a network upon completion of performance analysis. The results are computed over the model's steady-state and appear as reals at default positions beside their owner object. Figure 3 also exhibits the performance analysis (*Perf-Analysis*) menu which enables access to the full functionality of performance analysis, including analysis options, performance results to be computed and displayed, and of course, the performance engine trigger.

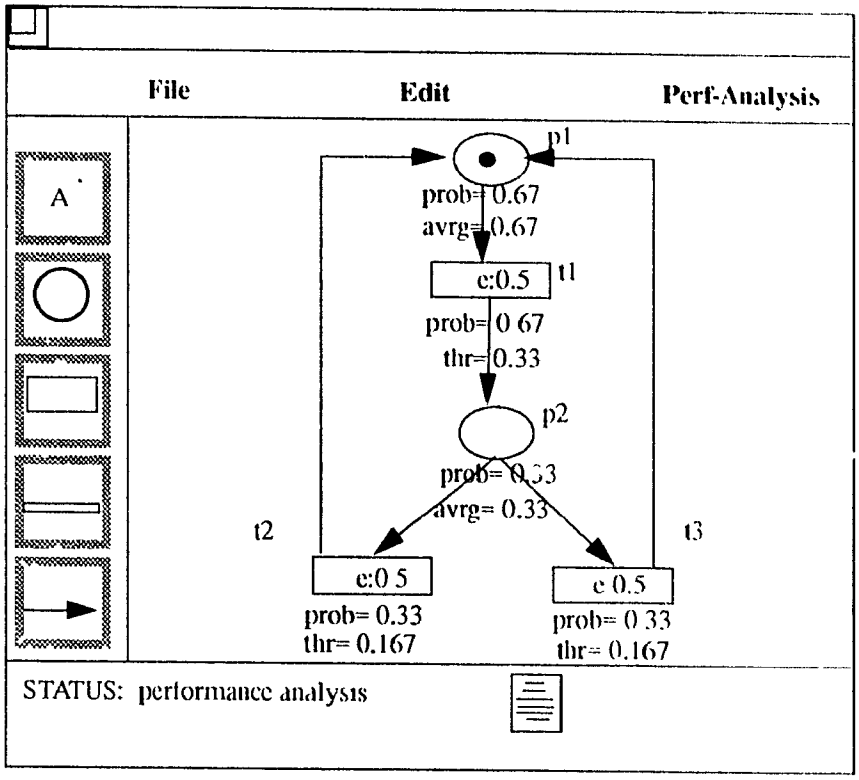

Figure 3 - Steady-state analysis

Transient results are positioned similarly to steady-state results, however, are initially represented as icons. Icons exhibit standard window system behaviour and can be expanded into resizable windows which, in turn, are re-iconizable. The window displays the actual graph computed.

Figure 4 shows the same model as the preceding figure upon completion of a transient performance analysis. The icon representing the average number of tokens of place p1 is shown in its expanded state.

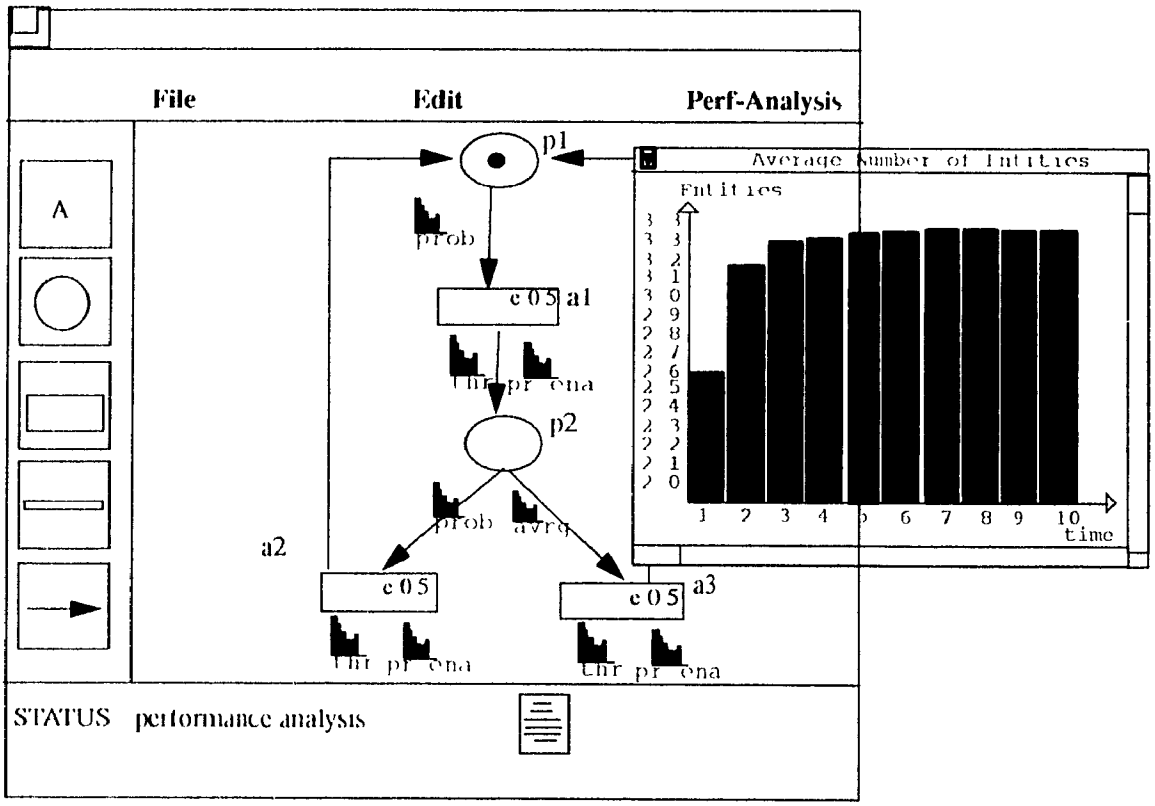

Figure 4 - Transient analysis

The graphical behaviour of performance indices related to basic network components is similar to that of objects' attributes in network edition. They may be dragged with the mouse and retain their relative position to their owner object if this one is dragged. They are, however, uneditable. Furthermore, graphical objects representing performance results react to the same principle of selective display as object attributes. That is, they can be shown and hidden at will, using a dialogue box provided for this purpose.

3.3.2 - Group Results

Group results are performance indices computed over a set of similar basic network components (i.e. a group of places or a group of actions) which are, ultimately, relevant given the semantics of the network. Groups consist of a set of places or a set of transitions over which results, similar to the results computed on individual components, are computed. For example, a network representing a company's overall activities can be partitioned by grouping places and transitions belonging to one particular branch of the company. Results computed over a group, in this example, a company's branch, are a higher-level indicator of overall performance.

For a group of places, as for individual places, we consider,

- the average number of tokens
- the probability that it contains at least one token

and for a group of transitions, as for individual transitions, we consider

- the probability of enablement
- the average throughput rate

Groups, being context-sensitive, necessitate user specification unlike basic component results which may be computed blindly for individual network components. A group of places, for instance, is defined by selecting the places belonging to the group and identifying the group with a unique label. Figure 5 shows a snapshot of the process, where

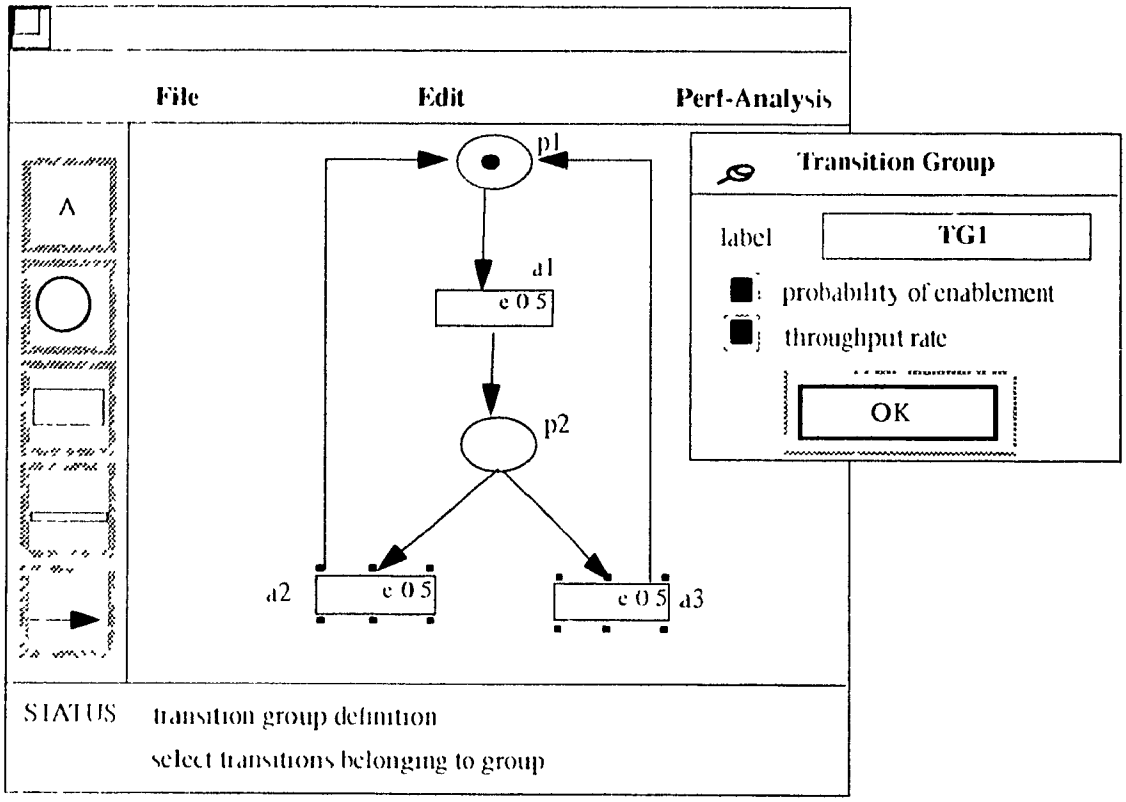

Figure 5 - Transition group definition

transitions *a2* and *a3* have been selected to form group *TG1*.The status area prompts users for the appropriate actions to take during *Place group definition*. The dialogue box permits group identification as well as the specification of the indices to be computed. The group is recorded upon a click on the *OK* button. It can be redisplayed and redefined, as the transitions belonging to a group are specially identified upon user request.

Upon completion of performance analysis, group results are displayable directly on the network or through a textual file. The first option, accessed through the *Perf-Analysis* menu, involves identifying the individual components of the group as well as displaying a box containing the group label and the computed indices. The second option is accessed through the file icon for *group* shown in the status region, as illustrated in Figures 3 and 4.

When expanded, the icon invokes a text editor which automatically loads a file of the following format:

GroupLabel$_1$
t_1
Tokens = i
% not empty = j

GroupLabel$_j$
t_j
Tokens = k
% not empty = l
.
.
.

where t_i is the set of transitions belonging to *GroupLabel*$_i$ and t_j is the set of transitions belonging to *GroupLabel*$_j$. *i*, *j*, *k* and *l* are real numbers.

Groups of places are defined and displayed in a similar manner than the one just described.

3.3.3 - State Results

Some performance results are related to *states* or markings reached by the network. The probability that a marking occurs, the return time to a marking, and the model's response time are examples of such results. Such a marking conveys particular meaning to users, given the semantics of the network. Results computed over states are context sensi-

tive, like results computed over groups.

The main challenge implied in the visualization of results defined over states, is not the display of the results but rather the specification of the state itself. We describe, hereafter, an original user interface approach which enables interactive specification of network states.

The specification of network states is initiated through a command of the *Perf-Analysis* menu. The main window enters the *state definition* mode, as shown in the status area of Figure 6. The network displayed does not contain the initial marking in order to enable the display of the user-defined state. The *State* dialogue box permits to enter the state label as well as the indices to be computed for that state. The *State* box remains throughout the state definition process, the final marking being recorded upon a click of the *OK* button. The state definition follows a behaviour identical to the initial making definition of mode *Edition* (cf. chapter 2). In effect, by double clicking on a place, a *Place* dialogue box, such as that shown in Figure 6 is displayed which enables the specification of the number of tokens in the related place. Like initial marking definition, advanced users may also define states by editing the default number of tokens, in this case the wild character '*'. Places which contain '*' number of tokens are not considered in the state definition. The tokens are displayed inside the place when the *OK* button of the *Place* dialogue box is clicked to permit the full visualization of the new marking. The process continues until the user presses *OK* in the *State* dialogue box.

The buffer-server example of Figure 7 illustrates the use of a simple interesting state definition. A buffer with maximum size 1000 blocks accumulates data produced by the *Input* component and consumed by the *Server* component. The designer's performance indice of interest is the probability of reaching the buffer's maximum capacity, given the rates of data production and consumption.

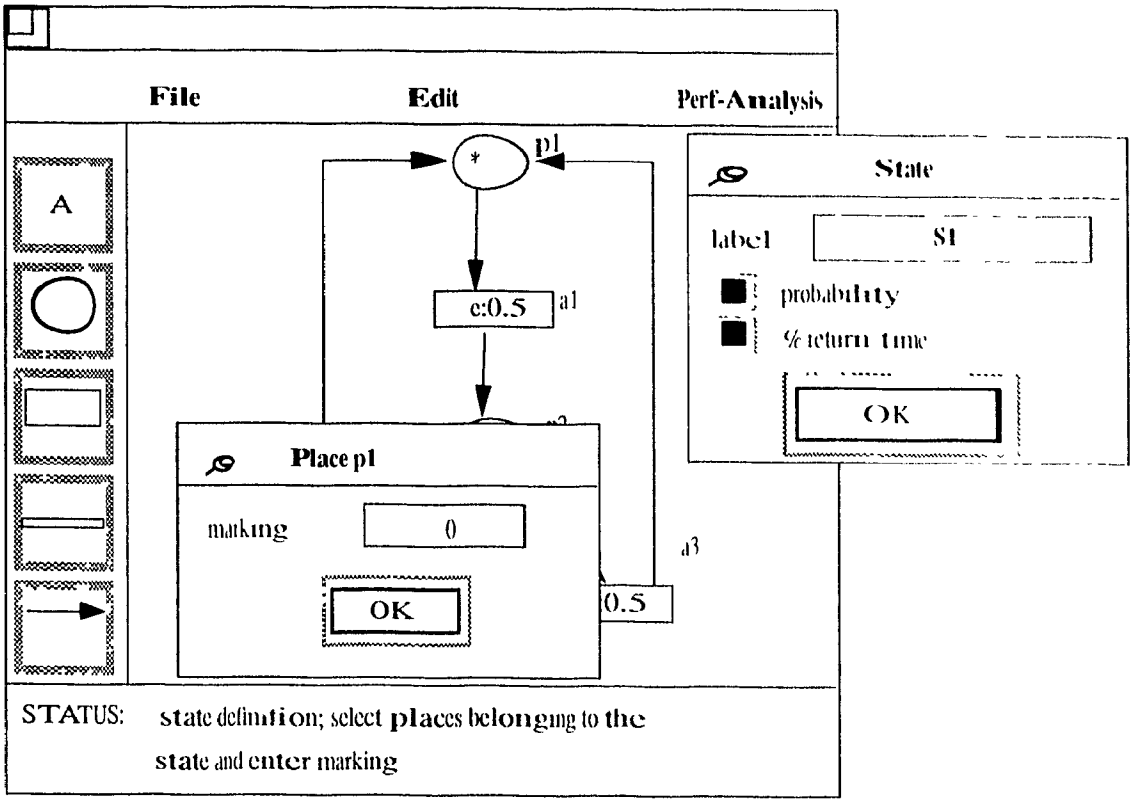

Figure 6 - State definition

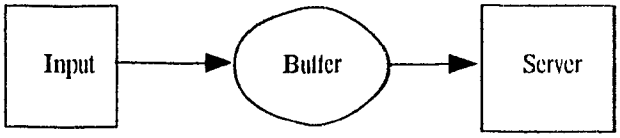

Figure 7 - Buffer-server example

The indices computed for user-specified states are displayed directly on the network or through a textual file. The first option is accessed through the *Perf-Analysis* menu and involves redrawing the tokens of the network to display the user-specified marking as well as displaying a box containing the computed indices. The second option enabling the display of state indices is accessed through a file icon shown in the status region upon completion of analysis, such as that illustrated in Figures 3 and 4. Double-clicking on the icon

invokes a text editor which automatically loads a file of the following format:

StateLabel$_i$
{(p_i, m_i), (p_{i+1}, m_{i+1}),..., (p_{i+n},m_{i+n})}
Probability = i
Return Time = j

StateLabel$_j$
{(p_j, m_j), (p_{j+1}, m_{j+1}),..., (p_{j+l},m_{j+l})}
Probability = y
Return Time = z

.
.
.

where i, j, y, and z are real numbers, m_i is the number of tokens of place p_i, and $\{(p_i, m_i)\}$ is the set of pairs (p_i, m_i) defined for marking $StateLabel_i$.

Versionning being an important task of the modelling activity, all performance results are retained in memory. Users may therefore modify object attributes or even subnets of equivalent purpose networks and compare performance results. Loading a network over which performance analysis has been previously run automatically enables the Show command for performance results and displays the result file icons in the status region of the main window.

Chapter 4

Integrating Graphical Simulation and Performance Analysis

The Macrotec Example

4.1 - About Macrotec

Performance analysis and graphical simulation are complementary strategies enabling network assessment and optimization. We envision a dynamic analysis tool which supports these strategies by offering an integrated environment based firstly on a formalism intelligible to both the performance analysis engine and the simulation engine, and secondly, on a user interface facilitating alternation between them.

We describe in this chapter Macrotec, a methodology and toolset targeted at business modelling [6] which realizes the integration of performance analysis and simulation. We begin with an introduction to the Macrotec methodology [4] and offer a functional overview of the Macrotec toolset. We pursue with our two engines of interest, that is, graphical simulation and performance analysis, and describe their design within Macrotec as well as their user interface. We conclude the chapter by illustrating the major Macrotec concepts in a typical scenario and by summarizing the main implementation issues.

4.2 - The Macrotec Methodology

Macrotec is a methodology and toolset for business modelling. It was developed in a joint project between CRIM and the DMR Group Inc. It combines several concepts originally developed in separate contexts, such as entity-relationship modelling of information, specialization and inheritance in the sense of object-oriented languages, event analysis, and analysis of data (product) flow as well as resource utilization. Theses concepts were integrated into a uniform modelling framework with precise semantics for the dynamic aspects, which have been defined through the formalism of Petri nets. We describe this formalism in what follows.

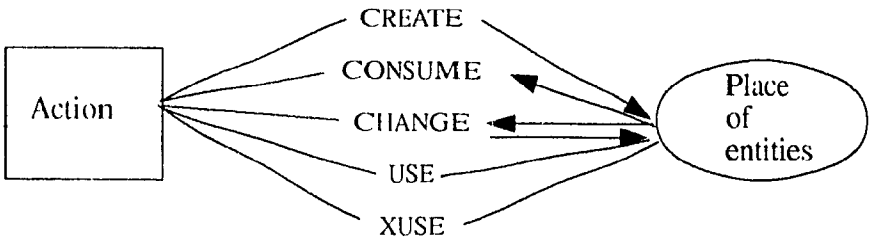

Figure 1 - Generic model for Macrotec

The generic model for the Macrotec formalism is shown in figure 1. Actions (represented as rectangles) act on places (represented as ellipses) of entities according to the type of relation between the place and action. The Petri net equivalent to the term "entity" is "token", whereas "actions" are equivalent to Petri net "transitions".

An *action* is a unit of work which produces a change in the system's state. Actions are characterized by a firing time which can be immediate, random exponentially distributed, or deterministic, as well as a weight used in situations of conflicts. *Entities* which reside

inside places may represent resources, for example, a human actor required in the accomplishment of an activity (action), or products created from an activity's execution. A *place* is a setting which contains a collection of entities belonging to the same type, a notion equivalent to colours in Coloured Petri Nets [34]. The initial marking defines the places' initial number of entities.

Relations connect places to actions and are specializations of Petri net "arcs". They affect the state change of the network when an action fires. Relations are characterized by a multiplicity and belong to one of 5 types - *Consume*, *Create*, *Change*, *Use*, and *XUse*. Upon action firing, entities are removed from the input place, added to the output places, changed, or used, possibly exclusively (XUse) by the action, according to the relation connecting the place to the action.

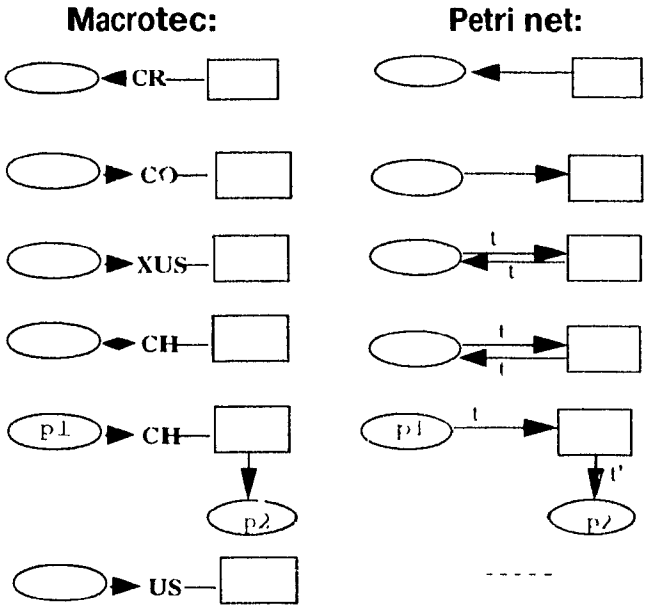

Figure 2 - Macrotec and Petri net equivalent

Figure 2 shows the correspondence between the Macrotec and the Petri net formalisms. The Create relation corresponds to an output Petri arc and the Consume relation to

an input Petri arc. The XUse relation corresponds to a input arc and an output arc connecting one place to a transition, the input tokens being the same as the output tokens. The Change relation involving two places corresponds to an input and an output Petri arc, where the input token value is different from the output token value. The Change relation involving one place only can be considered the special case of the Change relation involving two places, where the input and output place are the same. The Use relation does not have a Petri net equivalent.

4.3 - The Macrotec Functionality

The Macrotec methodology is supported by a toolset with the same name. The Macrotec toolset comprises the following tools and facilities:

- a tool for the graphical edition and validation of models, complemented with facilities for graphical layout,
- a dynamic analysis tool supporting both graphical simulation and performance analysis,
- a substitution tool for coping with the complexity of models based on a hierarchical approach and automatic abstraction of model parts into higher-level descriptions,
- facilities for data exchange and evolutionary design.

Figure 3 illustrates Macrotec's overall architecture, as implemented in version 1.0 [36]. Note that a unique interface provides external integration of the various tools and facilities. Users interact with the system via a single base window, giving them access to the full functionality of the system and allowing for easy switching between the different tools. Internally, the *core representation* manages all information manipulated and exchanged between the user interface and the various tools. Internal integration is furthered by the underlying storage facility, an object-oriented database allowing for storage

and retrieval of the core representation.

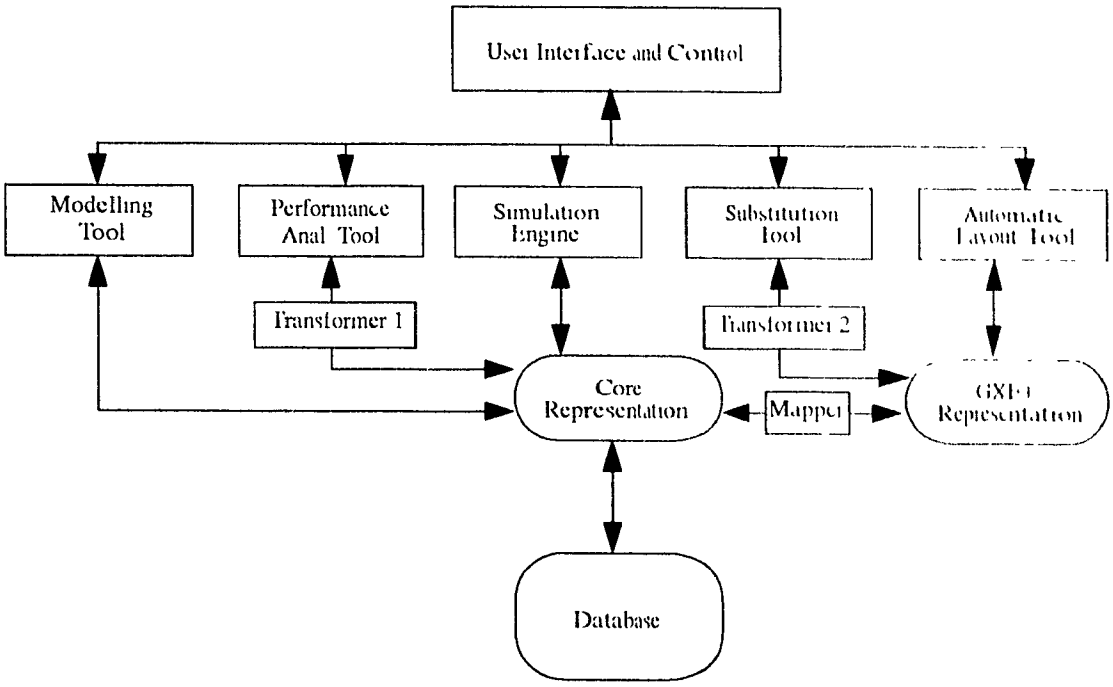

Figure 3 - Macrotec's overall architecture

Macrotec consists of two categories of tools. In the first category are the tools that manipulate graph layout data. Such tools store their data in the *GXF+* representation. GXF+ is our customized version of *GXF*, a standardized graph exchange format [52]. Supporting GXF+/GXF allows us to easily exchange data with other, special-purpose, GXF-based systems such as the automatic layout tools being developed at the University of Toronto. Non-GXF+-based systems require data transformation programs. For instance, integrating our substitution tool (implemented before adopting the GXF+ standard) required the development of the *Transformer2* program.

Tools belonging to the second category manipulate the model data that are not related

to the graph representation. In case these tools are part of the Macrotec process, e.g., the animation tool, they interact with the core directly. Otherwise, a data transfer program to and from the core is required. For instance, the performance analysis tool, running as a separate process, interacts with the main Macrotec process via *Transformer1*. Mapping of the core into the GXF+ representation and vice versa is carried out by the *Mapper* component.

The following sections discuss the internal and external integration of both simulation and performance analysis tools. Macrotec's user interface is an instantiation of the concepts discussed in the preceding two chapters. Developing this user interface allowed us to validate these concepts and support the functionality required in Macrotec version 1.0. The comprehension of the system's external integration necessitates a short introduction to its user's perspective which we discuss in the following section.

4.4 - The User's Perspective

The user interacts with Macrotec through one main window. Figure 4, for example, shows a main window containing the model of a simple delivery system. The main window consists of a palette on the left side containing the models' basic building blocks, where places are represented as ovals, actions as rectangles, and relations as directed arcs broken by the relation's acronym. The object attributes relevant for dynamic analysis are displayed inside their owner. Thus, a place's marking is shown inside the corresponding oval as an integer inside a small circle, and an action's firing time and weight are shown inside the corresponding rectangle. The multiplicity of a relation is represented as an integer beside the relation's acronym.

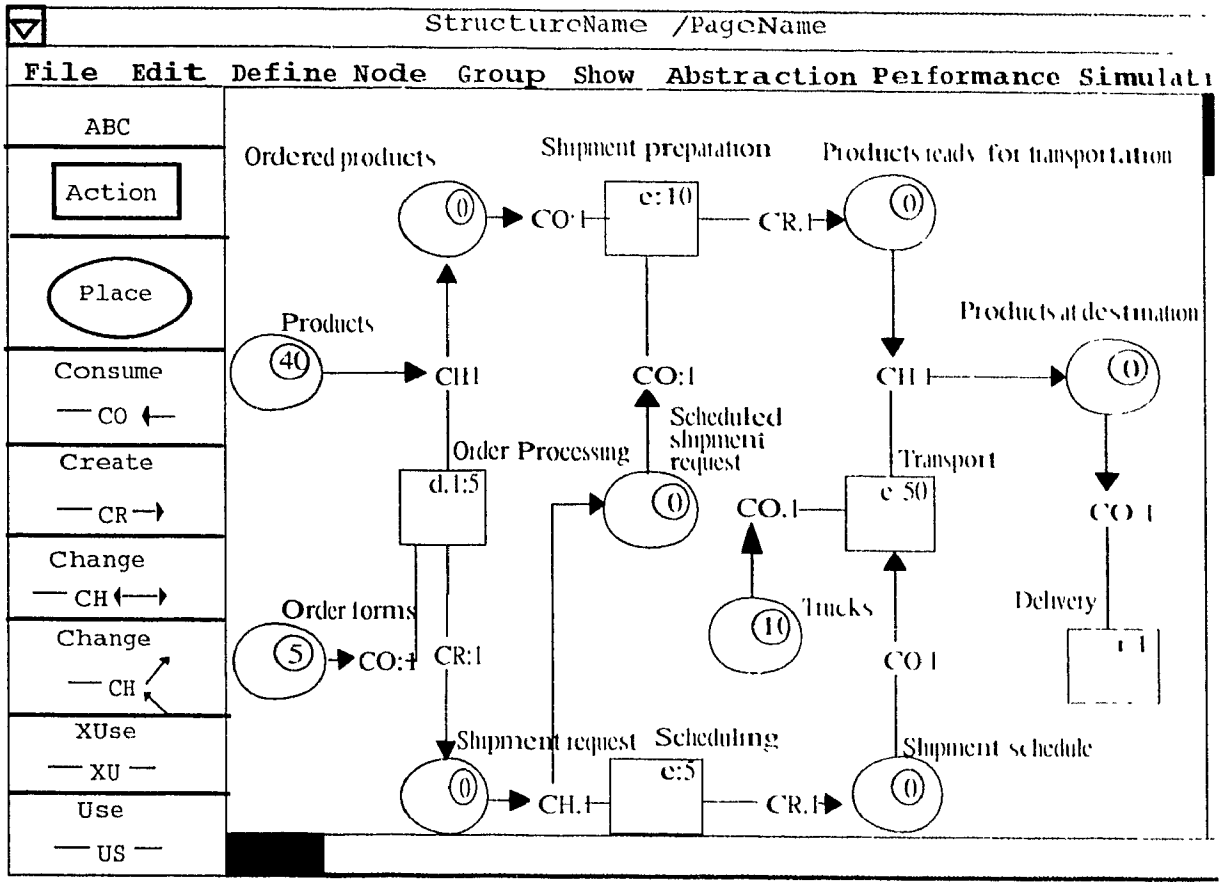

Figure 4 - Macrotec model of a delivery system

Graphical simulation and performance analysis can be directly launched from within the main window through menus *Simulation* and *Performance* respectively. Graphical simulation is carried out directly on the edited network, for instance, all graphical objects affected by a simulation step are refreshed to reflect their new state. Performance results are also displayed directly on the network.

The displayed model is in either of three states, animating, editing, or analysing performance. The control component at the user interface level (see figure 3) manages the access to these states and calls on the appropriate tool (either modelling, simulation, or performance analysis) to take over control when requested by the user.

4.5 - Graphical Simulation

The execution algorithm for simulation is identical to the one described in chapter 2, "Graphical Simulation". Thus, an action can be in one of four graphical states: inactive, enabled, firable and conflicting. Figure 5 illustrates the possible graphical states of an action, as simulation proceeds. The action shown fires in a deterministic time of 3 time units and its weight is 1, as noted by the inscription "d:3:1". An enabled action is distinguished by the display of a timer which is initialized to the action's firing time and decreased until the action becomes firable.

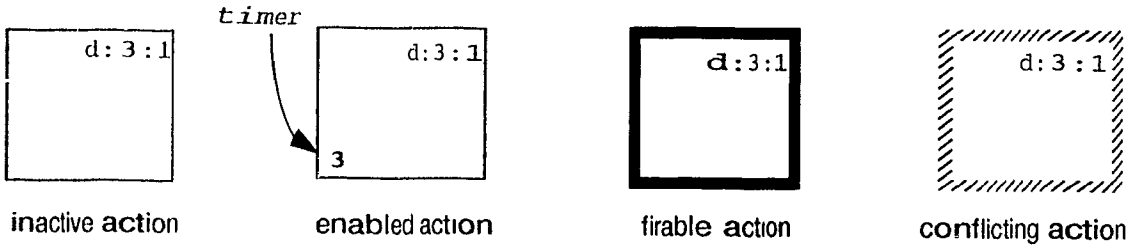

Figure 5 - Possible action graphical states during simulation

Figure 6 shows the dialogue box enabling control of step-by-step simulation. The box displays the global simulation time as well as the step number (c.f. chapter 2, "Graphical Simulation"). Buttons *Next Step* and *Cancel* permit to pursue one step of simulation or quit step-by-step simulation altogether.

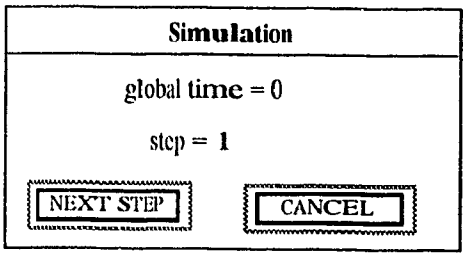

Figure 6 - Simulation Control

Figure 7 shows the portion of the system's architecture that interacts with the simulation engine. The core and the simulation engine keep different representations of the model. The core is the stored, central representation of the user interface from which other architectural components are initialized. The simulation engine's data is initialized to the core data (i.e. the initial marking), augmented with the simulation parameters specified by the user and is modified, during simulation, to keep track of new markings. The user interface stores the graphical information relative to the model, recording every change being made to the graphical objects.

From the selection of the simulation command to the graphical display of the first simulation step, we follow the control flow. First, the core is updated to the currently displayed model (graphical objects). The core and the user interface not being identical at all times, it is necessary to synchronize them prior to simulation. The control component launches the simulation engine, triggering its initialization procedures. The engine updates its representation, mapping it to the core's, and processes one simulation step. The Control component requests the new state of the model from the simulation engine and calls on graphical objects to update themselves accordingly. All above steps other than initialization procedures are repeated each time the user presses the *Next Step* button (see figure 6).

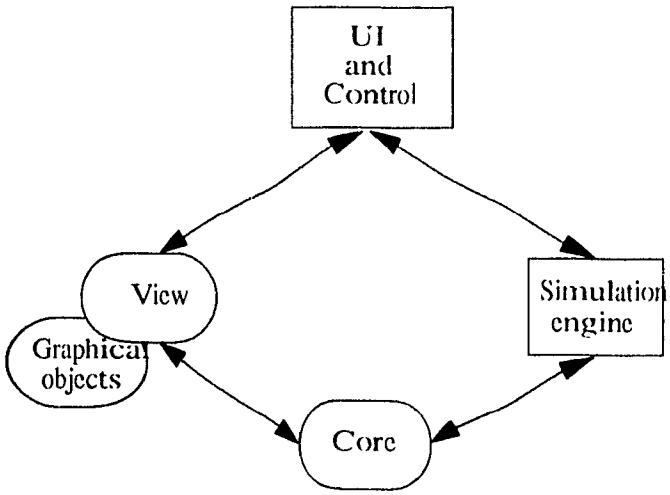

Figure 7 - Simulation Architecture

4.6 - Performance Analysis

The results of a preliminary study of various performance analysis tools conducted in the scope of the Macrotec project [35] suggested the use of an existing package to help achieve the dynamic analysis requirements of Macrotec [36]. The report of this study is presented in Appendix B. Five packages were evaluated, namely the Great Stochastic Petri Net package (GreatSPN) [15] from University of Turin, the Stochastic Petri Net Package (SPNP) [60] from Duke University, Design/CPN [54] from the Meta Software Corporation, Eval [73] from Verilog, and Voltaire [60] from McGill University. SPNP was selected for reuse in Macrotec due to the proximity of its underlying formalism to Macrotec's and its availability.

SPNP's underlying formalism is in fact an extension of the stochastic Petri net, namely the Stochastic Reward Net (SRN) [18] which is based on the Markov reward model paradigm [63]. SPNP provides a set of custom performance measures which can be computed either at steady-state or at specific time points. Furthermore, it allows for the specification of custom measures defined in terms of reward rates associated with markings or rewards associated to transitions.

A first prerequisite to the usage of the SPNP package in Macrotec concerns the ability of converting Macrotec models into SRN models used by SPNP. Secondly, in terms of performance, SPNP must be accessible not as a whole, but rather its relevant procedures yielding solution methods computable over SRN models must be accessible directly through Macrotec, and this, with full access to their available options. Finally, a last requirement to the use of SPNP is the availability of the package's analysis results in such a format that enables extraction and, after possible manipulations, integration into the Macrotec's data structures.

In order to bridge the gap between the Macrotec formalism and SRNs, a transformation unit is required that converts Macrotec models into SRNs according to the correspondence table of figure 2. The resulting model, however, somewhat loses descriptive power, a few "holes" in the SRN formalism necessitating approximate fillings. For example, the Use relation which has no Petri counterpart, is treated as a XUse relation, and deterministic firing times which are not supported in SRNs are approximated by random, exponentially distributed firing times. Based on our experience however, this loss of descriptiveness is not overly severe [76].

The SPNP package has a strictly textual user interface. SRN models are described using CSPL (C-based SPN language) [16] which is a library of C programming language constructs that facilitate easy description of SRN models. The SRN model, the analysis options as well as the performance measures are contained in SPNP's input C file which, after compilation, is run according to the specified solution method. Performance results are written into an output file with a well-defined format that allows convenient atomic retrieval.

Figure 8 shows the major architectural components involved in Macrotec's performance analysis functionality. Upon selection of the performance analysis command, the Control component updates the core representation of the model and launches the *Transformer*. The Transformer, containing knowledge about both formalisms, permits the translation from Macrotec into SRN models. The resulting SRN is written into a CSPL file following SPNP format, along with the proper analysis options as defined via the user interface. The Control component triggers the execution of SPNP's resolution methods on the input file and prompts the *Extractor* to scan the resulting file, extract pertinent results, and insert them into the core. Finally, Control calls on graphical objects at the user interface level affected by the analysis to update themselves according to the core, i.e., to display the relevant analysis results.

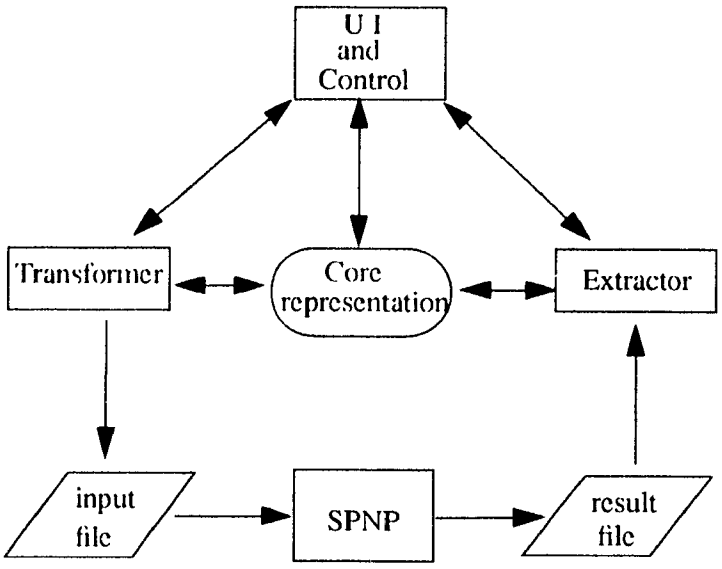

Figure 8 - Performance analysis architecture

Macrotec's current implementation supports performance results attributed to individual places and actions [39]. These are displayed by default beside their related object and can be hidden on request.

4.7 - Scenario

One typical application domain of Macrotec is the modelling of enterprise information systems for prescriptive usage. We provide a scenario of this kind of modelling. We modelled, as an example, a simple delivery system with products being ordered, transported, and finally delivered to destination. Note that we have successfully applied Macrotec to real business applications [76].

Figure 4 shows a network loaded by the user, possibly reusing a template or parts of an existing model. The user may edit and refine the network, adopting a top-down or bottom-up approach by respectively decomposing or abstracting parts of the network. The built-in validation component of the modelling tool makes sure that the different levels of the network are consistent.

The simulation engine is triggered through the base window. The graphical execution of the model may be performed at user-specified hierarchical levels, allowing for partial model assessments and permitting the user to define a configuration that corresponds to his or her mental model of the system.

Performance analysis generates quantitative results on model behaviour. For example, a comparatively low action throughput and average number of entities in an action's input place ("low" might have different significance depending on the network architecture) may confirm a bottleneck that has already become apparent during simulation. In light of these results, the user may increase the number of over-strained resources to smoothen execution (in our example, we would increase the number of "Trucks", if the action "Transport" had low throughput). Users may prototype the model by running, in an iterative way, simulation and performance analysis, until the appropriate attribute values and configuration for a desired model behaviour are found.

4.8 - Implementation

To conclude this chapter, we provide a few remarks about Macrotec's implementation. Macrotec is a Unix-based system developed mostly in C++ (minor parts were written in C), running under SunWindows, NeWS, and the X11 window system. Its user interface has been implemented with the *ET++* application framework [74], [75], as database we are using Gemstone[1] [13]. As hardware, we use Sun Sparc-2 workstations. We do not

1. Gemstone is a registered trademark of Servio Corporation

make use of colour in Macrotec, since this would make us dependable on the availability of colour monitors. Moreover, we contend that the visualization concepts developed in our project are expressive enough and do not necessitate colour. To date, we have invested more than four person years in its development.

Performance is a key factor for successful dynamic analysis. Our experience shows that our platform produces efficient graphical simulations. As for performance analysis, the efficiency obviously depends on the complexity of the model. With complex models, the bottleneck of performance analysis is mostly due to the back-end component, namely SPNP, as opposed to the front-end, the user interface, whose performance is only slightly degraded by the loading of results from the data base.

The data transformation programs required to integrate external tools into Macrotec are an indicator for the extensibility of the system. According to our experience, those programs dealing with the GXF+ representation tend to be considerably longer (4:1 ratio in lines of code) and more complicated than the ones interacting with the core representation. However, this will not be a severe limitation, since future Macrotec extensions will most likely require a transformation of type Transformer 1 (c.f. figure 3). We do not see the need for further graph manipulation tools, and hence transformations of type Transformer 2 will not be required.

We have adopted an object-oriented development approach that has provided significant productivity and quality gains. For instance, our user interface software is highly flexible and reusable, in part due to the use of ET++. ET++ is a powerful, object-oriented class library integrating user interface building blocks with high-level application framework components. The usually steep learning curve for such application frameworks has been alleviated by the use of some powerful C++ development tools, most notably, the *Sniff* tool [3]. Another benefit of the object-oriented approach in Macrotec is the use of Gemstone together with its C++ interface which has led to an efficient database interface. To prototype the user interface of Macrotec, we ran several user interface development cycles. The integration of several standalone tools with their own graphical user interface

was an incentive to meet (or surpass) the usability criteria of each.

Various existing dynamic analysis tools such as Design/CPN support graphical simulation and provide quantitative performance results. However, these tools do not support performance analysis in the full sense, as described in chapter 3. To enrich behavioural model comprehension in Macrotec, we chose to support both graphical simulation and performance analysis as we developed our own simulation engine and made practical reuse of the SPNP performance analysis package. To our knowledge, a single algorithm encompassing all facets of simulation and performance analysis is not, to this day, available. Moreover, it is questionable whether such an algorithm would perform reasonably.

As our system evolves, with new external tools requiring integration, we will be in a better position to determine how easily our system can be adapted and hence whether or not our design and guiding principles are indeed sound.

Chapter 5

Graphical Simulation as Algorithm Animation

5.1- Foreword

This chapter is concerned with the field of algorithm animation. We first introduce the domain and then correlate its techniques with that of graphical simulation, as studied in Chapter 2. We will show that graphical simulation can be interpreted as a special case of algorithm animation. In fact, in algorithm animation, the program's runtime operations and data are of interest and dictate how the animation appears. In a simulation instead, only the simulation data being produced by the executing program are considered relevant and displayed graphically, whereas the program itself and its internal data serve only as a means to produce and display the simulation data. We refer to the term "graphical simulation" to underline this distinction[1]. Based on this finding, we describe the design of graphical simulation as implemented in the Macrotec toolset in terms of algorithm animation systems such as Tango [65] and Balsa [10] and address its extensibility to support multiple-views and multiple-models, as general algorithm animation systems.

5.2- Fundamentals

Algorithm animation is a form of program visualization which complements traditional textual communication by using the "technology of interactive graphics and the crafts of graphics design, typography, animation, and cinematography to enhance the presentation and understanding of computer programs [2]". Specifically, animated visualizations help to evaluate existing programs by monitoring and comparing their performance

1. Note that what we call *graphical simulation* is sometimes referred to as *animation*.

with related programs. They support programmers at debugging programs and developing alternative solutions to problems by studying the dynamic behaviour of algorithms.

Special-purpose software environments for algorithm animation are often referred to as *algorithm animation frameworks* or *algorithm animation systems*. Such frameworks provide support for the activities of two types of users, namely *viewers* and *animators* [10]. Viewers directly interact with the dynamic graphical representations of the algorithm which are programmed by animators. We first reveal basic algorithm animation techniques from the viewer's point of view and pursue by describing the animator's tasks.

5.2.1- The Viewer's Perspective

In this section, we illustrate our discussion with figures from the Balsa algorithm animation system [10]. We provide examples of sorting algorithms since they are widely used as test-beds for such systems.

Viewers specify the algorithms and views of an animation and are responsible for managing the layout of views on the screen:

Algorithm. Numerous algorithms may be run simultaneously in order to compare solution approaches. Figure 1 shows the Balsa algorithm animation framework running on four sorting algorithms. The Shellsort, Heapsort, Mergesort, and Quicksort each run simultaneously in separate windows containing a "Dots view" of the algorithm. Each element of the array being sorted is represented as a square whose horizontal coordinate corresponds to its position in the array and whose vertical coordinate corresponds to its value.

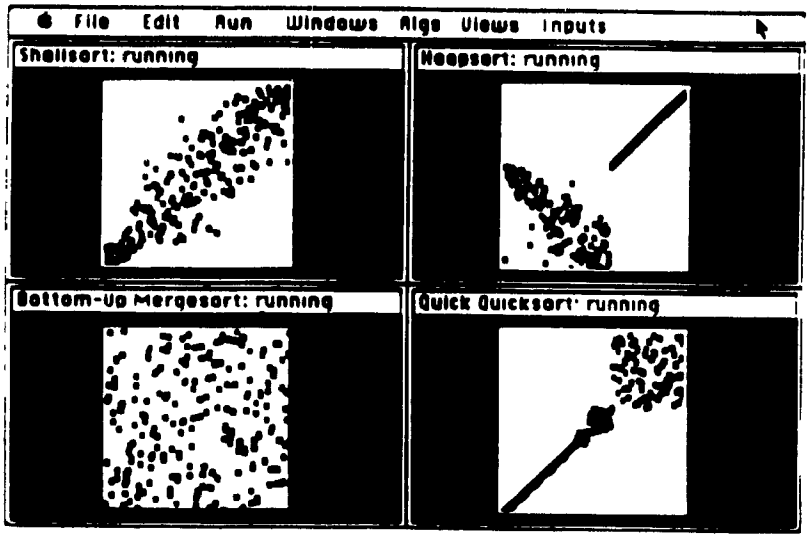

Figure 1 - Four algorithms running simultaneously [10]

Views. A view is a graphical display reflecting the algorithm's behaviour. Numerous views provided for an algorithm may be opened simultaneously, thus complementing each other's presented information. Views may be opened and closed during the algorithm's execution. Figure 2 shows multiple views of the Quicksort algorithm running under the Balsa system. The Dots view is shown on the left and the "Partition-Tree" view on the right. The latter displays each element of the array being sorted as a node in a tree. Its horizontal coordinate corresponds to its position in the array. Circular nodes are in their final position in the array, whereas contiguous square nodes represent sublimes waiting to be processed according to their depths in the tree. The example illustrates the use of state cues to show changes in the state of the algorithm's data structure, as the different nodes' graphical representations distinguish sorted and unsorted elements.

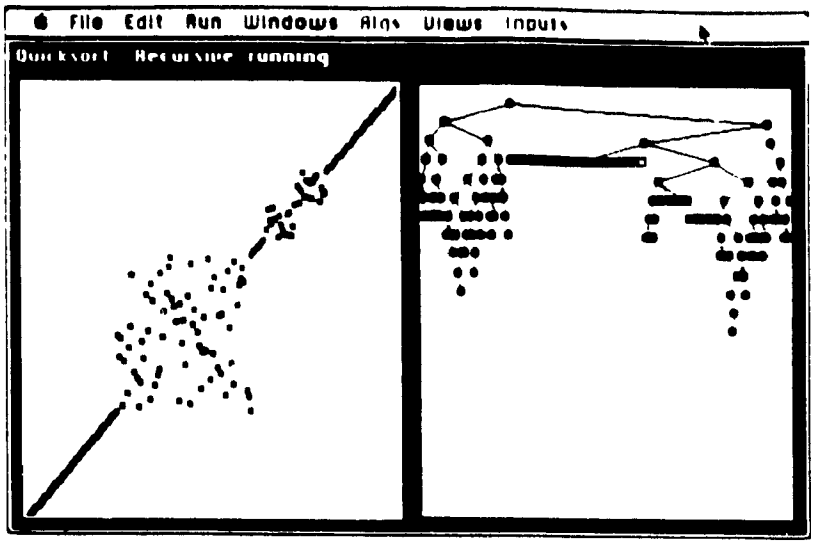

Figure 2 - Multiple views of an animated algorithm [10]

Communication between the view and the algorithm is bidirectional. Views update their display according to the algorithm's state. On the other hand, views may transfer user input to the algorithm. Such input is based on the current runtime state of the algorithm. Figure 3 shows a tree-traversal algorithm which has run through its initialization phase and now pauses, awaiting for the user to select a vertex from where the traversal will begin. The window's left side displays the undirected graph view of the algorithm. The right side shows the adjacency matrix view, where a marker at position (x,y) indicates the presence of a directed edge between the *xth* and the *yth* vertices.

A further task of animation viewers is the control over algorithm execution. Animation may be carried out at various speeds, the views may be refreshed at specific points in the algorithm, and the animation can be reset. As an example, Balsa distinguishes between "step-points" and "stop-points" at which to update and refresh the views. Step-points are analogous to a conventional debugger's notion of "single stepping" through the algorithm, that is advancing to the next line. The animation may also pause when a certain stop-point in the algorithm has been reached a specified number of times. The determination of algo-

rithm instructions available as refresh points is accomplished by the animator, as we will see in the following section.

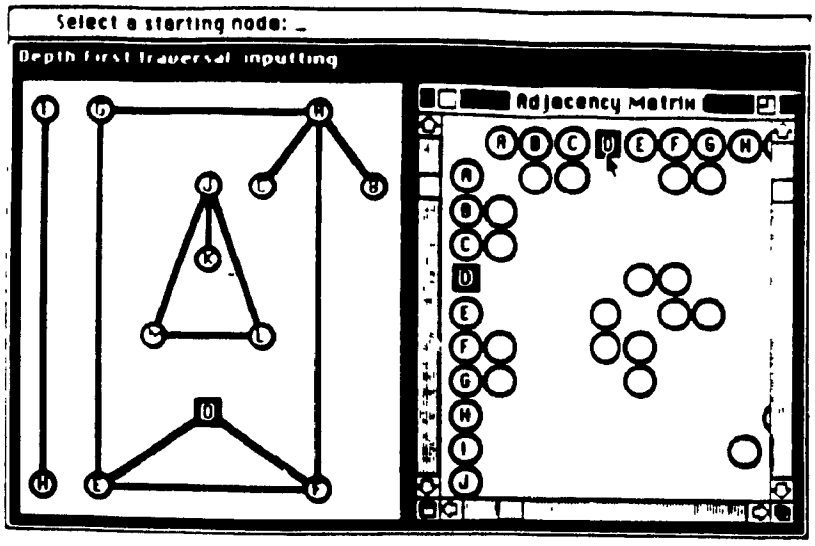

Figure 3 - An animated algorithm awaiting for user input through a view [10]

5.2.2- The Animator's Perspective

Animators are responsible for the two following main tasks:

Annotating the algorithm. Annotating the algorithm to be animated consists of inserting into it animation-specific operations which update the algorithm view display. These operations are sometimes refered to as *interesting events* since they are the algorithm's fundamental operations selected for describing the algorithm's dynamic behaviour through its views. As a rule of thumb, interesting events are inserted wherever a *print* or a *read* statement might have been placed for debugging, tracing, or generating output of the algorithm in a non-graphical environment. For example, the Selection Sort algorithm finds

the smallest item of an array and swaps it with the item in the first position. It then finds the second smallest item and swaps it with the item in the second position, continuing in this way until the sort is completed. Two events of interest which may be reflected in the views are the Compare and Exchange operations.

Designing views. This task consists of implementing the data structures which make up the view as well as their behaviour. Graphical shapes (for example, rectangles or circles) as well as the operations which reflect a change in their state (for example, move or highlight) are therefore implemented at this stage. Views, which are updated each time an interesting event is reached during the algorithm's execution, require a mapping mechanism from the screen coordinates into the object displayed, in order to allow users to dynamically modify the algorithm's data structures. Common logical structures of views such as graphs and trees, as well as common behaviour such as window-management aspects, may be stored into libraries; thus the animator needs only code particulars of the view which itself may be shared by many algorithms.

Designing views is an art which, above all, necessitates a thorough understanding of the algorithm to be animated. Brown presents in [9] a three-dimensional framework for characterizing algorithm animation views. Along the first axis of this framework, views are characterized according to their content. Animators may choose to design views whose displays are isomorphic to the algorithm's data structures, so that at any instant, one can be reconstructed from the other. On the other hand, animators may design views which display synthetic information of the algorithm, and therefore have no direct mapping to the algorithm. The second axis of Brown's framework presents a characterization of views according to the type of transition occurring between the states of objects in the view. Transitions may be incremental and therefore smooth, or they may be discrete, the latter resulting in abrupt visual changes. Finally, the third axis characterizing views is the persistence of the view. Views may display the current status of the algorithm or may show a history of the each change in information. Animators will design views along this framework depending on the most convenient representation for end-users.

Once the annotation of the algorithm and the design of views are completed, animators must coordinate the algorithms and their related views. This task is highly dependent on the algorithm animation system's design. In the following section, we discuss the main design concepts of four existing algorithm animation systems which, on the basis of the diversity of their design, we selected to present.

5.3- Algorithm Animation Systems

Algorithm animation systems provide support for watching, hearing, and interacting with dynamically changing graphical representations of an executing program. It also provides tools to construct algorithm views and map them to the program. A wide variety of designs were developed to implement animation algorithm systems that underlie diverse coupling between their components, as we see below. In this section, we describe the design of four systems that we feel representative of the diversity in algorithm animation system design.

5.3.1- MVC-Based Systems

Some algorithm animation systems have been developed using object-oriented methods. Object-oriented programming techniques are claimed to lend themselves naturally to animation [46], a data structure being an "object", a program running on the data structure being a "method" executed within or upon that object, and the graphical representation of

the object being a "view". One such system [46] functions within Smalltalk's model view-controller paradigm [43]. In this paradigm, the *model* is the program's data structures and behaviour. The *view* displays the program's state and handles graphical tasks. The *controller* contains the interface between its associated model and view and input devices, and schedules interactions with other view-controller pairs. Figure 4 shows the typical model-view-controller structure.

General purpose animation algorithm systems based on the MVC paradigm tend to lack performance in realizing real-time dynamics of programs [46]. This limit led to the development of application-oriented systems. We describe three such systems in the following sections.

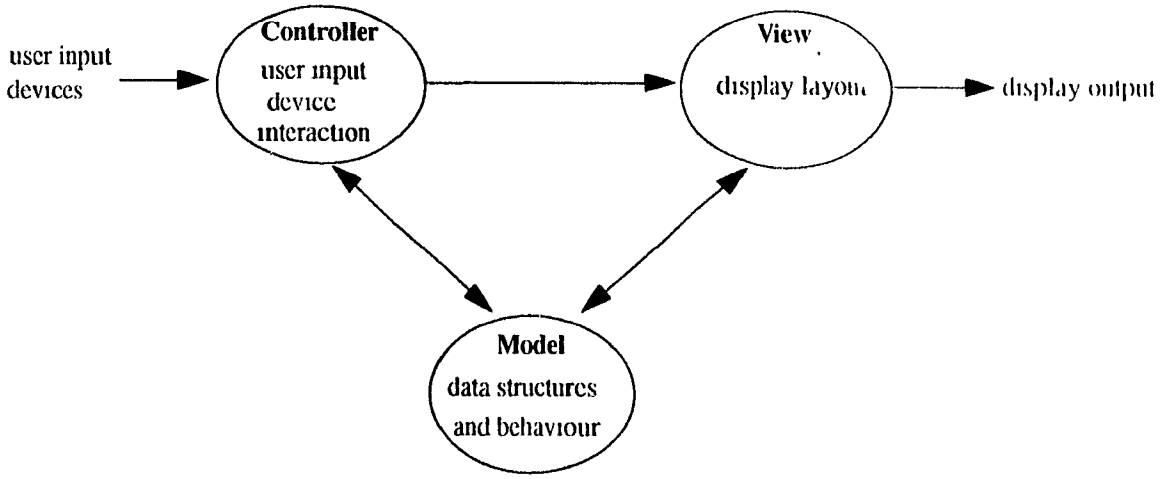

Figure 4 - Model-View-Controller structure

5.3.2 - Animus

Animus [22] is based on Smalltalk, however it uses temporal constraints as defined in

ThingLab [6] to describe the appearance and structure of displayed objects and their evolution in time, resulting in increased performance. In Animus, objects created have their behaviour described in terms of timed or static constraints. When the model executes, time advances through a global clock triggering time-dependent events that alter the state of objects. These events may, in turn, trigger other events that are dependent on the state of these objects. The treatment of the event queue generated in this way thus maintains and propagates constraints describing objects states.

5.3.3 - Balsa

In the Brown University ALgorithm and Animation (Balsa) system [10] a *modeller* maintains an abstract representation of the information that a *renderer* draws on the screen. A single model may be shared by multiple renderers that simultaneously display different views of the model. Figure 5 shows Balsa's basic architecture. Rectangles represent components implemented by the animator, whereas ovals represent part of the animation system that route information among components. Here, two views noted by Renderer-1 and Renderer-2 share the same model of the algorithm. An *Adapter* is necessary to transform data from the stream of algorithm output events to the information needed by the modeller and renderers.

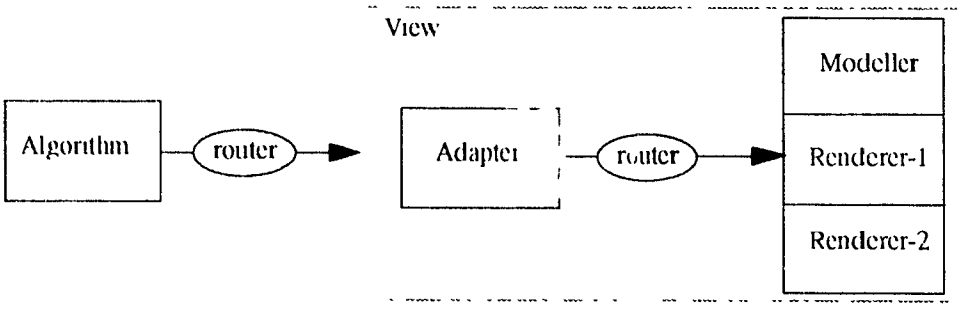

Figure 5 - The Balsa architecture

5.3.4 - Tango

Stasko's Tango [65], [69] is a more recent framework for algorithm animation. Figure 6 shows Tango's basic architecture. Key operations which are important to the algorithm's semantics are specified following two equivalent methods. The first one uses a text editor to add explicit procedure calls directly into the algorithm. The second one uses a tool, Field's annotation editor [65], to scan the code and define "algorithm operations" interactively through a dialogue window. This method has the advantage of leaving the original algorithm intact since no extra code is embedded into it. Algorithm operations are forwarded to the mapping component when they are reached during program execution.

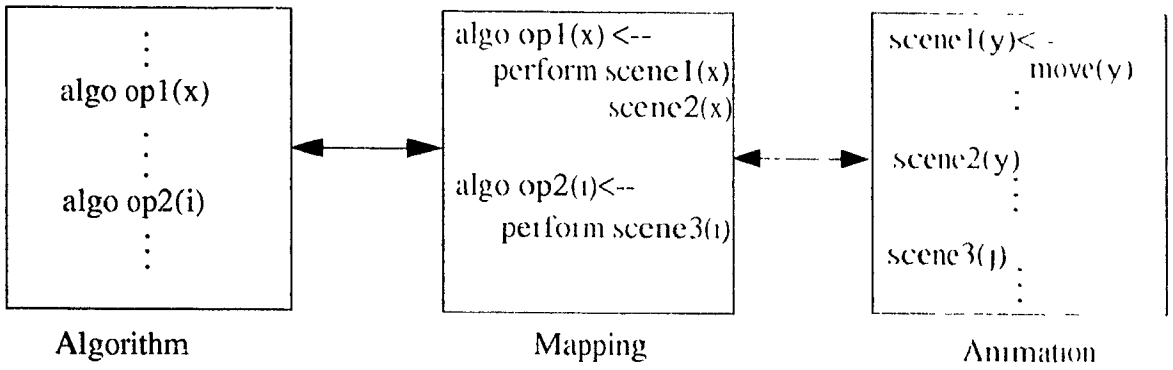

Figure 6 - Tango's architecture

The animation component describes graphical objects and the operations on them. It may reference a library written in C which makes use of four abstract data types that define a formal methodology, namely the path-transition paradigm [66]. The *image* type represents the graphical object. The *location* type is the object's position in (x,y) coordinates. The *path* is the change in image attributes from one update of the display to the next. Finally, the *transition* is an animation action which uses a path to modify an object's position or appearance. With the animation's library, users may create animation scenes which are, in fact, procedures inside the animation component and which are composed of predefined Tango instructions from the animation library that alter the state of objects

using the four data types and their methods. Algorithm operations inserted into the original code are mapped to animation scenes which are performed when they are reached during execution.

The mapping component is a control file containing the algorithm operations and the animation scenes that must be played when an algorithm operation is reached during execution. Algorithm operations may contain parameters (program variables) that are sent to and used by the animation scenes. Users need only modify this control file to associate different scenes to an algorithm operation. Many scenes may be associated to an algorithm animation, a feature lacking in Balsa which allows only one-to-one mappings between operations and scenes.

Animation within Tango runs via two UNIX processes, namely the animation process and the algorithm process. The animation process first reads the control file of the mapping component and dynamically loads the animation-specific operations. It then waits for the algorithm operations from the animated algorithm to be dispatched. The algorithm process is executed and algorithm operations are distributed to the animation process as they are reached. Interprocess communication is achieved via the Field environment.

5.4- Macrotec's Graphical Simulation as Algorithm Animation

In studying different algorithm animation systems, we found ET++ an appropriate environment for developing graphical simulation in a business-oriented application. In many ways, ET++'s internal schemes for managing graphical applications are similar to that of Smalltalk's MVC paradigm. We therefore chose to construct Macrotec's graphical simulation component directly from these available schemes which furthermore lend themselves well to extensibility, as discussed in the next section. Moreover, little effort was foreseen to implement the graphical behaviour of objects as ET++ contains an exten-

sive library of graphics behavioural primitives.

Figure 7 reproduces a detailed picture of Macrotec's simulation architecture as discussed in chapter 4. The UI and Control component refers to MVC's controller. It picks up user input and triggers other architectural components. The Simulation Engine refers to MVC's model. It contains the data structures representing the model as well as the behaviour of the model. Lastly, the view is responsible for the graphical layout of objects representing the model.

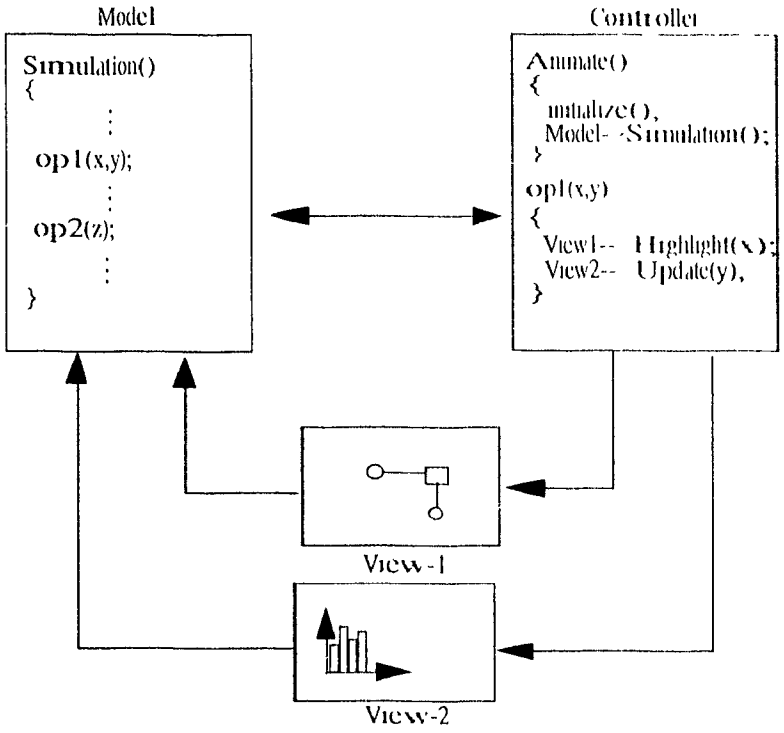

Figure 7 - Macrotec's detailed simulation architecture

5.5- Extending Graphical Simulation in Macrotec

5.5.1- Extensibility of the Design

We have seen in chapter 4 that Macrotec implements only a subset of the visualization concepts discussed in chapter 2. We demonstrate in this section the extensibility of Macrotec's graphical simulation design to support future modifications encompassing the features in chapter 2. These features may be categorized into three groups according to the Macrotec architectural component they affect. We identify each group below and discuss the impact of their integration into Macrotec on the tool's architectural components.

The first group comprises features that affect the number of views involved in the simulation. Numerous views may be added into Macrotec and managed automatically using ET++'s internal schemes. The extensibility of the number of views is a crucial factor to the expressiveness of graphical simulation. In particular, it enables the representation of the quantitative behaviour of the network. For example, figure 7 contains two different views. View 1 is the qualitative view of simulation where the logical behaviour of the network is described through firable action highlights, action timer display and other graphical concepts described in chapter 4. View-2 is an additional view representing one aspect of the quantitative behaviour of the network. In this figure, a histogram is displayed that may represent such indices as the throughput of an action versus time.

The second group of features which may be added to Macrotec affects the Model component of the tool's architecture. The simulation algorithm, as implemented in Macrotec's current version, does not contain interesting events in the sense of algorithm animation. This is due to the fact that the Model implements a single step of simulation and that the current view is altered only once execution has reached the end of this step. The need

for adding interesting events becomes obvious in attempting to extend the graphical simulation component to support smooth graphical variations as proposed in chapter 2. In fact, smooth token movement and smooth thermometer decrease that show the time elapsed since the beginning of an activity require the insertion into the Model of operations affecting the state of views. A simulation step, which is strictly associated to an increase of the global simulation time in Macrotec's current version, may be split into substeps so as to permit the graphical representation of intermediary logical operations such as the execution of an activity and token movement, and this in order to increase the network's comprehension.

The last group of simulation features which may be added to Macrotec's functionality affect both the Model and the View For example, the addition of interesting events into the Model and the addition of views may be proved essential in simulating hierarchical Petri nets. Hierarchical nets are specified on numerous edition pages, representing different levels of abstraction. Each page is made to correspond internally to a specific view Interesting events positioned at specific points in the Model where simulation dynamically affects a different page can be used to automatically display the corresponding view Finally, comparative animation as described in chapter 2, implies the use of at least two views. Interesting events may be essential to display the state of two networks under comparison, independently from the actual point of execution reached in both Models For example, both network states may be displayed as soon as one Model reaches the end of a simulation step.

Macrotec's Controller architectural component is evidently affected by these three groups of features, as it maps the interesting events to the views. A typical scenario where all graphical simulation components play a role is described hereafter and refers to figure 7.

The Controller picks up a request from the user to trigger simulation and surrenders

control to the Model. When execution reaches interesting event *op1(x,y)* which is simply a procedure call with parameters x and *y*, the control is retrieved by the Controller and execution of the corresponding procedure takes place. This procedure contains operations that affect one or many views. In our example, View-1 is affected and may call on its graphical objects to update themselves according to parameter x.

5.5.2- Extending Macrotec's Development Environment

We discuss in this section a potential path along which to augment Macrotec's support for graphical simulation. Macrotec's development environment, ET++, provides little explicit support, as is, for animating displays in the pure graphical sense. Flexible abstractions in Artkit [31] which support powerful animation techniques can be added to ET++ in order to ease the programming of altering graphics states. We describe in the following some of the animation techniques described in [31] which would help implement the features discussed in the previous section and thus refine Macrotec's graphical simulation.

Timing refers to the properties of moving objects. A Petri-Net token, for example, can be shown as migrating from the input to the output place of a firable action. Controlled timing is provided to allow objects to migrate in a specific time with as smooth a transition as the OS and window system performance and timing allow.

The ***slow-in / slow-out transitions*** technique provides for non-uniform pacing of movements. Tokens in Macrotec's simulation may begin their trajectory slowly, move quickly along the input and output arcs, and end slowly. This way, the accent in the movement is put on the input place from which the token originates and the output place to which it is destined.

Arcs describe trajectories for movement as a parametric curve of one variable. This provides a uniform framework for describing token movement along the network arc during graphical simulation.

Artkit is a C++ library built on powerful animation abstractions that support sophisticated animation techniques such as these ones. The integration of Artkit into ETPN may be an approach to enrich Macrotec with direct support for animated presentations.

Chapter 6

Ongoing and Future Work

6.1 - Tailored Modelling and Dynamic Analysis Tools

Formalisms such as Petri Nets have demystified dynamic analysis and made of performance evaluation a field no more restricted to performance specialists. Results generated from performance analysis and simulation can be pertinently extracted and presented intelligibly to diverse users through Petri nets and an attractive, easy-to-use interface.

General graphical dynamic analysis tools may be tailored to application domains of interest. Tools intended for business domains, such as Macrotec, may present results and enable result definition in terms of business indices such as raw and net profits, depreciation, etc. In this way, the interpretation of results is furthered and little analysis is left on the part of users. End-users needs, in particular their terminology, concepts, and model evaluation criterias, must be identified in order to adapt on one hand the performance and simulation engines to support relevant indice computation, and on the other hand, to adapt the user interface to present these indices textually following the domain's terminology and graphically, following the domain's concepts.

Improvements along this path have been anticipated in designing the Macrotec tool. An optional view, for example, displays network components as icons representing business concepts such as reports, human resources, machine resources, etc. The detailed design of such a view, currently under implementation, can be found in [41].

6.2 - Animation Extensions to User Interface Frameworks

Currently available user interface frameworks offer little support for animated displays. Flexible animation abstractions that could be integrated into a user interface framework could be explored using the experience gained in the object-oriented ET++. These abstractions would support animation scene descriptions, using a library such as Artkit.

6.3 - A Comparative Study of Tools Based on the Petri Net Formalism

Partly inspired by the MACA project, we are currently undertaking a study at CRIM to evaluate and compare tools which are based on the Petri-Net formalism. Petri Nets being widely used in software engineering for modelling and analysing systems, the evaluation of available CASE tools based on this formalism takes on increasing importance. The major achievement sought in this process is to develop a methodology for evaluating Petri net-based tools, apply it, and, in the long term, transfer our experience. The activities involved in the project are described hereafter.

Pertinent properties of projects benefiting from the Petri Net approach are identified, and based on these, evaluation criterias are elaborated. The realization of this task is enriched through the experience gained from MACA and another finished project at CRIM, both involving modelling and formal analysis. The projects' different application domains, on one hand real-time constraints and on the other, organisational flow, conveys generality to the identified properties and wideness their spectrum.

Next, a literary review concerning similar studies and Petri-Net based tools in general

serves as a basis to establish a list of such tools, may they be commercial or research. Those tools that are submitted to our evaluation are then selected, principally on the basis of their availability.

Prototype examples that exhibit the pertinent properties are elaborated, inspired from the two projects discussed above. Those examples are implemented with the tools selected for studying and evaluated according to the identified criterias. Obviously, this is a good opportunity for evaluating Macrotec which uses the SPNP package. Finally, results are synthesised into a comparative report of Petri-Net tools, whereas the methodology and experience developed from the exercise is detailed to serve potential users.

The status of this study is currently at the implementation stage. Interested readers may find related documentation as internal CRIM reports in [19], [20], [21], [27], [28].

Conclusion

The skills required in the modelling and dynamic analysis of systems through specialized textual languages such as GPSS is mastered by few, leaving this domain unexploited due to its evident restrictive usage. Concerns of this nature guided the elaboration of high level formalisms that became the basis of graphical dynamic analysis tools which take advantage of their visual potential. Petri Nets are widely used in modelling and analysis activities due on one hand to their descriptive power of phenomena such as concurrency and synchronization, and on the other, to their applicability to formal performance analysis methods. In particular, timed Petri Nets enable direct representation of concepts like activities, their inputs, outputs, and duration, making them a favoured approach to business modelling.

Major facets of the dynamic analysis of a model are simulation and performance analysis. Both serve to assess, evaluate and find alternatives to a model. Whereas simulation is mostly a guide towards the model assessment, performance analysis mostly provides for model evaluation as it generates quantitative measures of the model at its equilibrium state through formal methods.

Packages exist that offer integrated support for graphical model edition, graphical simulation, and performance analysis. Whereas graphical edition is a well developed discipline, graphical dynamic analysis generally suffers from poor user interaction and graphics techniques. We have attempted in this work to rationalize and structure basic visualization concepts gained from the study of various existing packages and recommended certain extensions that we feel augment the transparence of the dynamic behaviour of models. Furthermore, we have shown that techniques from the algorithm animation field can be applied to improve dynamic analysis visualization, as both are concerned with state modifications through time.

Visualization concepts promote user accessibility to both inputs and outputs of simulation and performance analysis engines. Inputs include the network itself and the options available for the execution of the engines. Inputs are defined through a state-of-the art graphical editor and dialogue boxes. Outputs are the results generated from the engines. In the case of the simulation engine, principal results are displayed on the network itself. As for the performance analysis engine, results are displayed textually or as graphs, and are distinguished according to the network components they refer to, may they be individual components, groups of components, or states of the network. Performance analysis may be tailored by defining a variety of custom indices relevant within the network's semantics, possibly in a graphical way.

Further directions in the field of dynamic behaviour visualization include tailoring the user-interface to specific application domains. We see in this concept a potential to support users from interpreting analysis results, that is, from the need of computing by hand measures of interest using basic indices automatically generated. Improvements in this direction would provide users with high-level performance indices that, ultimately, directly match their measures of interest.

Acknowledgments

I thank Ruedi Keller for his sincere encouragements and his diplomatic way of getting me back on the right track when I was digressing, and most of all for his most valuable resource, his time, of which he was never scarse. Our countless discussions resulted not only in improvements related to my work but also in a personal working methodology I have adopted and will carry on in the future.

Working with Macrotec[1] programmers and Tigers made the most wicked bug seem inoffensive as the talent and dynamism of our team was almost palpable. Thanks to all of you for easing debugging sessions, and releasing the tension during intensive coding sessions with your optimism and laughter.

Finally, I cannot pass this opportunity of thanking members of the McGill secretarial office without whose patience and understanding, I would never have accomplished this

1. Macrotec was developed as part of a joint CRIM/DMR Group Inc. project, which is part of the IT MACROSCOPE project.

Bibliography

[1] R. Y. Al-Jaar, A. A. Desrochers.Performance evaluation of automated manufacturing systems. In *Transactions on Robotics and Automation*, pages 621--639, 6(6), December 1990.

[2] Ronald M. Baecker. An application overview of program visualization. *Computer Graphics*, July 1986.

[3] Walter R. Bischofberger. Sniff - a pragmatic approach to a C++ programming environment. In *Usenix C++ Conference*, Portland, OR, August 1992.

[4] Gregor v. Bockmann. Decoupage et simulation d'architectures distribuees. Technical Report CRIM-91/09/20, Centre de Recherche Informatique de Montréal (CRIM), Montreal, September 1990.

[5] G. v. Bochmann, A. Debaque, R. Dssouli, A. Jaoua, R. Keller, N. Rico, and F. Saba. A method for architectural modelling and dynamic analysis of information systems and business processes. Technical report CRIM-92/12/10, Centre de Recherche Informatique de Montreal (CRIM), Montreal, December 1992.

[6] A. H. Borning. Thinglab - a constraint-oriented simulation laboratory. PhD thesis, Stanford University, March 1979.

[7] Mark H. Brown and Robert Sedgewick. Techniques for algorithm animation. *IEEE Software*, pages 124-135, January 1985.

[8] Gretchen P. Brown, Richard T. Carling, Christopher F. Herot, David A. Kramlich, and Paul Souza. Program visualization: graphical support for software development. *IEEE Computer*, pages 27-35, August 1985.

[9] Mark H. Brown. Perspectives on algorithm animation. In *Proceedings of Human Factors in Computing Systems*, pages 33-38, 1988.

[10] Mark H. Brown. Exploring algorithms using Balsa-II. *IEEE Computer*, pages 136-157, May 1988.

[11] Mark H. Brown. Zeus: A System for algorithm animation and multi-view. Technical report, Digital, Palo Alto, February 1992.

[12] Mark H. Brown and John Hershberger. Color and sound in algorithm animation. *IEEE Computer*, pages 52-63, December 1992.

[13] Paul Butterworth, Allen Otis, and Jacob Stein. The Gemstone object database management system. Communications of the ACM, 34(10), pages 64-77, October 1991.

[14] G. Chiola. A Graphical Petri net tool for performance analysis. In Modelling Techniques and Performance Evaluation, pages 323, 1989-333, North Holland, 1987.

[15] G. Chiola. GreatSPN users' manual, version 1.3. Technical report, University of Turin, September, 1987.

[16] G. Ciardo. Manual for the SPNP Package, version 3.1. Technical report, Duke University, March 1992.

[17] G. Ciardo, J. K. Muppula, and K. S. Trivedi. Analysing concurrent and fault-tolerant software using stochastic reward nets. To appear in *Journal of Parallel and Distributed Computing*

[18] G. Ciardo, J. K Muppula, and K. S. Trivedi. On the solution of GSPN reward models. In *Performance Evaluation*, pages 237-254, 12(4), July 1991.

[19] Anne-Claire Débaque and Fayez Saba. Exemple de modélisation d'un processus de demande de prêts. Technical report CRIM, Centre de Recherche Informatique de Montréal (CRIM), Montreal, May 1993.

[20] Anne-Claire Débaque, Paul Freedman, Jean-Michel Goutal, Rudolf Keller, Michel Levy, Marianne Ozkan, and Fayez Saba. Critères d'evaluation d'outils basés sur les réseaux de Pétri. Technical report CRIM, Centre de Recherche Informatique de Montréal (CRIM), Montreal, May 1993.

[21] Anne-Claire Débaque, Paul Freedman, Jean-Michel Goutal, Rudolf Keller, Michel Levy, Marianne Ozkan, and Fayez Saba. Liste d'outils de simulation basés sur les réseaux de Pétri. Technical report CRIM, Centre de Recherche Informatique de Montréal (CRIM), Montreal, May 1993.

[22] Robert A. Duisberg. Animation using temporal constraints: an overview of the Animus system. *Human-Computer Interactions*, vol. 3, pages 275-307, 1987-88.

[23] Thomas Eggenschwiler and Erich Gamma. ET++ Swaps Manager: Using Object Technol-

ogy in the Financial Engineering Domain. In *OOPSLA'92*, pages 166-177. Vancouver, B.C., October 1992.

[24] F. Feldbrugge. Petri net tool overview 1989. In *Advances in Petri Nets*, pages 152-178, 1989.

[25] F. Feldbrugge. Petri net tool overview 1992. Private Communication, 1993.

[26] Ulrich Frank. Designing procedures within an object-oriented enterprise model. In *Proceedings of the Third International Working Conference on Dynamic Modelling of Information Systems*, pages 365-385, Noordwijkerhout, The Netherlands, June 1992.

[27] Paul Freedman and Rudolf Keller. ECORP: Etude comparative des outils Réseau de Petri. Project description, Centre de Recherche Informatique de Montréal (CRIM), Montreal, December 1993.

[28] Paul Freedman and Michel Levy. Exemple de modélisation d'un processus d'ouverture et de fermeture automatique. Technical report CRIM, Centre de Recherche Informatique de Montréal (CRIM), Montreal, May 1993.

[29] W. H. Harrod and R. J. Plemmons. Comparaison of some direct methods for computing stationary distributions of Markov chains. In *SIAM journal Sci. Stat. Comput.*, June 1984.

[30] Doug Hayes. The XTANGO environment and differences from Tango. Technical report, Georgia Institute of Technology,Georgia, 1990.

[31] Scott E. Hudson and John T. Stasko. Animation support in a user interface toolkit: flexible, robust and reusable abstractions. Technical report, Georgia Institute of Technology,Georgia, 1992.

[32] Daniel H. H. Ingalls. The Smalltalk graphics kernel. *Byte*, pages 170-194, August 1981.

[33] A. Jaoua, J.M. Beaulieu, N. Belkiter, A.C. Débaque, J. Debarnais, R. Lelouche, T. Monkam,and R. Reging. Rectangular decomposition of object-oriented software architectures. Technical report, Laval University, Quebec, June 1992.

[34] K. Jensen Coloured Petri Nets: A High level language for system design and analysis. Aarhus University, Denmark, November 1990.

[35] Rudolf K. Keller, Marianne Ozkan, and Natalie Rico. Comparaison fonctionelle des outils de simulation. Technical report, Centre de Recherche Informatique de Montréal (CRIM), Montreal, September 1992

[36] Rudolf K. Keller, Richard Lajoie, Marianne Ozkan, Fayez Saba, Xijin Shen, Tao Tao, and G. v. Bochmann. The Macrotec toolset for CASE-based business modelling. *Proceedings on Computer-Aided Software Engineering*, Singapore, July, 1993. To appear.

[37] Rudolf K. Keller, Richard Lajoie, Marianne Ozkan , Fayez Saba, Xijin Shen, Tao Tao, and G. v. Bochmann. User interface aspects in the Macrotec toolset for business modelling and simulation. HCI poster, Orlando, Florida, August 1993. To appear.

[38] Rudolf K. Keller, Richard Lajoie, Marianne Ozkan ,Fayez Saba, Xijin Shen, Tao Tao, and G. v. Bochmann. User interface aspects in the Macrotec toolset for business modelling and simulation Unpublished.

[39] Rudolf K. Keller, Richard Lajoie, Marianne Ozkan , Fayez Saba, Xijin Shen, and Tao Tao. Macrotec version 1.0 user interface specification. Technical report, Centre de Recherche Informatique de Montréal (CRIM), Montreal, May 1993.

[40] Rudolf K. Keller, Richard Lajoie, Marianne Ozkan , Fayez Saba, Xijin Shen, and Tao Tao. Macrotec version 1.0: design and implementation. Technical report, Centre de Recherche Informatique de Montréal (CRIM), Montreal, May 1993.

[41] Rudolf K. Keller and Xijin Shen. Macrotec version 1.0 : icone view specification. Technical report CRIM, Centre de Recherche Informatique de Montréal (CRIM), Montreal, May 1993.

[42] Thomas Kofler. Robust Iterators in ET++. *StructProg*, 14(2), pages 62-84, April/June 1993.

[43] Glenn E. Krasner and Stephen T. Pope. A Cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, pages 26-49, August 1988.

[44] J. D. C. Little. A Proof for the queuing formula L=lambdaW. *Operations Research*, vol. 9, pages 383-387, 1961.

[45] J. C. Lloret, J. L. Roux, B. Algayres, and M. Chamontin. Modelling and evaluation of a satellite system using Eval, a Petri net based industrial tool. In *Application and Theory of Petri*

Nets, ed. K. Jensen, 1992.

[46] Ralph L. London and Robert A. Duisberg. Animating programs using Smalltalk. *IEEE Computer*, pages 61-71, Augut 1985.

[47] M. A. Marsan, A. Bobbio, G. Conte,and A. Cumani. Performance analysis of degradable multiprocessor systems using Generalized Stochastic Petri Nets. *Distributed Processing T-C Newsletter*, no. 6, pages 47-54, 1984.

[48] M. A. Marsan. Stochastic Petri nets: An elementary introduction. Tutorial notes, *11th International Symposium on Protocol Specification, Testing, and Verification*, Stockholm, 1991.

[49] M. A. Marsan and G. Conte. A Class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(2), pages 93-122, May 1984.

[50] M. A. Marsan, G. Balbo, G. Chiola, G. Conte, S. Donatelli,and G. Franceschini. An Introduction to generalized stochastic Petri nets. *Microelectron. Reliab.*, 31(4), pages 699-725, 1991.

[51] B. Melamed and R. J. T. Morris. Visual simulation: The Performance Analysis Workstation. *IEEE Computer*, pages 87-94, August 1985.

[52] Alberto O. Mendelzon. Declarative database visualization: recent papers from Hy+/ GraphLog project. Technical report CSRI-285, Computer Systems Research Institute, University of Toronto, June 1993.

[53] A. Merlin and D. J. Farber. Recoverability in communications protocols - implications of a theoretical study. *IEEE Trans. Commun.* vol. 9, September 1976.

[54] Meta Software Corporation. Design/CPN user manual version 2.0. 1992.

[55] M. K. Molloy. Performance analysis using stochastic Petri nets. *IEEE Trans. Comput.*, vol. 9, pages 913-917, September 1982.

[56] M. K. Molloy. On the Integration of delay and throughput measures in distributed processing models. Ph.D. dissertation, University of California, Los Angeles, 1981.

[57] Darren New and Paul D. Amer. Adding graphics and animation to Estelle Technical

report, University of Delaware, Newark.

[58] Darren New and Paul D. Amer. Protocal visualization of Estelle specifications. Technical report, University of Delaware, Newark.

[59] J. D. Noe and G. J. Nutt. Macro E-nets representation of parallel systems. *IEEE Trans. Comput*, vol.8, pages 118-127, August 1973

.

[60] Pierre Parent and O. Tamin. Voltaire. a discrete event simulator, *IEEE PNPM-91 Conference Proceedings*, Melbourne, Australia, December 1991.

[61] G. J. Ramackers and A. A. Verrijn-Stuart. First and second order dynamics in information systems. In *Dynamic Modelling of Information Systems*, pages 237-256, North-Holland, 1991.

[62] Yen-Ping Shan. MoDE. A UIMS for Smalltalk. Proceedings of ECOOP/OOPSLA, pages 258-268, October 1990.

[63] R. M. Smith, K. S. Trivedi, and A. V. Ramesh. Performability analysis: measures, an algorithm, and a case study. *IEEE Trans. Comput.*, pages 406-417, April 1988.

[64] John T. Stasko and Joseph F. Wehrli. Three-dimensional algorithm animation. Private communication.

[65] John T. Stasko. Tango: a framework and system for algorithm animation. IEEE Computer, pages 27-39, September, 1990.

[66] John T. Stasko. The Path-Transition Paradigm: A Practical methodology for adding animation to program interfaces. Technical report, Georgia Institute of Technology, Georgia, June 1991.

[67] John T. Stasko and Doug Hayes. XTANGO algorithm animation designer's package. Technical report, Georgia Institute of Technology, Georgia, October 1992.

[68] John T. Stasko and Eileen Kraemer. A Methodology for building application-specific visualizations of parallel programs. Technical report, Georgia Institute of Technology, Georgia, June 1992.

[69] John T. Stasko. Simplifying algorithm animation with TANGO. Private communication.

[70] K. S. Trivedi, J. K. Muppula, S. P. Woolet, B.R. Haverkort. Composite Performance and

Dependability Analysis. *Performance*, 14(3-1), pages 197-215, 1992.

[71] Robert Valette. Les Réseaux de Pétri. L.A.A.S./C.N.R.S., Toulouse, France, May 1990.

[72] A. Verbraeck and F. W. Wierda. Interactive modelling for Information Systems design: The MOSAIC tool. In *Dynamic Modelling of Information Systems*, ed H. G. Sol and K. van Hee, pages 189-225, Amsterdam, 1991.

[73] Verilog. Stochastic Petri net solver, release 1.0. 1990.

[74] Andre Weinand, Erich Gamma,and Rudolph Marty. ET++ - An object-oriented application framework in C++. *Object Oriented Programming Systems, Languages, and Applications '88 Conference Proceedings*, September 1988.

[75] Andre Weinand, Erich Gamma,and Rudolph Marty. Design and implementation of ET++, a seamless object-oriented application framework, *Structured Programming*, 10(2), pages 63-87, 1989.

[76] Ghassan Youssef and Fayez Saba. Décomposition et simulation d'architectures distribuées. exemple d'architecture.Technical report CRIM, Centre de Recherche Informatique de Montréal (CRIM), Montreal, July 1993.

Appendix A

Glossary of Basic Petri Net Terminology

This appendix defines the basic Petri net concepts without, however, getting into an exhaustive study of Petri net theory. These notions are essential to the comprehension of this research. Readers unfamiliar with Petri nets are encouraged to read the definitions in order to subsequently use them as a reference guide.

A ***token*** is a marker. It can be an object, a state, or anything we care to associate with it. Tokens may take on values belonging to the network's datatype or *colorset*

A ***place*** is a location that may contain zero or more tokens. A place may be *typed*, meaning that all its tokens belong to the same colorset. A *marking* is a state of the net. It can be described by the number of tokens residing in each place. The *initial marking* is the state of the net preceding any action firing.

A ***transition*** is an action whose occurrence can change the number and values of tokens in one or more places. A transition is *enabled* if the necessary conditions permit it to start its execution. A transition becomes *firable* when it has finished executing. A transition is *inactive* if it is not executing and the conditions required to start execution are FALSE.

The ***firing time*** of a transition is the time lapse between the transition's enablement and its actual firing. Transitions may be *timed* or *immediate*. Timed transitions may have in the case of stochastic nets non-deterministic firing times, typically random, exponentially distributed firing time or they may have a fixed deterministic firing time. Immediate transitions fire in zero time. *Firing rates* are associated to exponentially distributed timed transitions and may depend on the marking of the network. Figure 1 shows the different states of a transition as time increases. Figure 2 shows the distinction between the different firing times of transitions.

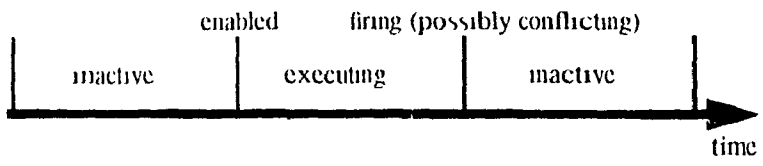

Figure 1 - Transition states as a function of time

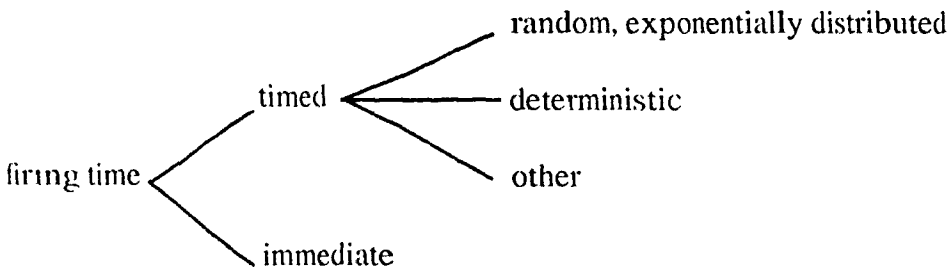

Figure 2 - Possible firing times of a transition

A ***guard*** is a boolean expression associated to a transition. The guard must be evaluated to TRUE for the transition to become enabled.

An *arc* is a connection between a transition and a place. A transition may require more than one token from an input place in order to execute. Modelling one-to-one relations, in terms of tokens, between input and output places is cumbersome and may reduce overall readability. A notation to represent many-to-many relations can be used to reduce diagram size. Such a notation may simply take on the form of an integer associated to each arc specifying the number of tokens that will be removed from the input place of the transition or added to the output place of a transition when it fires. We use the term *multiplicity* to describe the concept of having a certain number of tokens migrating through the arcs.

Arcs may have associated inscriptions that define the set of tokens involved in the firing of the connected transition. Furthermore, they may inhibit a transition when the number of tokens in its input place is greater or equal to x. We call this type of arc, an *inhibitor* arc.

The set of places, transitions and arcs as well as the initial marking are sometimes referred to as the *structural component* of the net as opposed to the *dynamic component* which refers to the transitions' firing times.

When many transitions are simultaneously firable and share an input place that contains an insufficient number of tokens to provide all these transitions, then the transitions are said to be in *conflict* with one another. To control conflicts, *weights* may be associated to transitions.

In our work, Petri net elements are graphically represented as follows

- Arcs are represented as unidirectional arrows,
- Places are represented as ovals,
- Immediate transitions are represented as "flat" rectangles,
- Timed transitions are represented as rectangles,
- Tokens are represented as small filled circles.

Figure 3 shows a network containing four places (p1, p2, p3, p4), two immediate transitions (t2 and t3) and two timed transitions (t1 and t4). Transition t1 has a random, exponential firing rate e1. t4 has a deterministic rate d4. Transitions t2 and t3 are conflicting, they are always enabled simultaneously. Weights w2 and w3 are therefore assigned to t2

and t3 respectively. Two arcs have a multiplicity of 2. The initial marking consists of a single token inside p1.

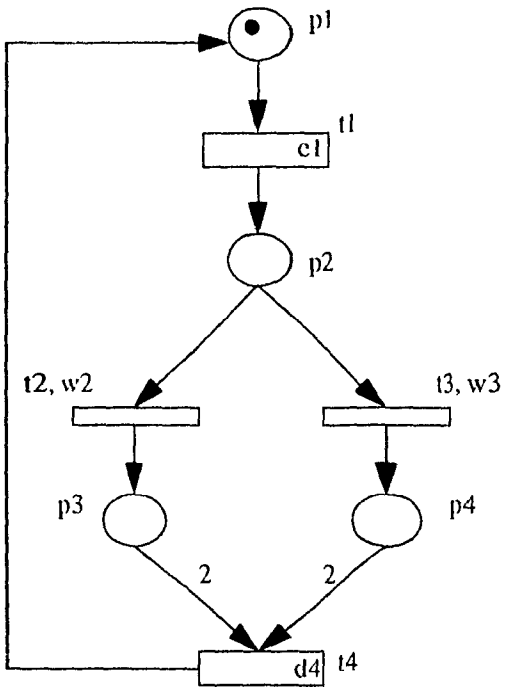

Figure 3 - An example Petri net

Translation Table

Alternating between languages often distorts and perverts semantics. In an attempt to clarify Petri Net terminology and ease communication, we provide the French translations we adopted to terms that, according to our experience, have caused some controversy

when improperly translated.

transition firing time	durée d'execution
enabled transition	transition activee ou transition executable
fired transition	transition franchie
firing	franchissement

Table 1 - English-French translation of common Petri net terminology

Appendix B

Functional Comparison of Various Tools Based on the Petri Net Formalism

Comparaison fonctionnelle de quelques outils informatiques basés sur le formalisme Réseaux de Pétri: GSPN, SPNP, Design/CPN et Voltaire

Document de travail

Marianne Ozkan
Natalie Rico
Rudolf Keller

Juillet 1992

Le projet MACA (MACroscope Architecture) inclue l'implantation d'un outil de modélisation et de simulation de systèmes d'information ("Macrotec toolset"). Cet outil est basé sur la méthodologie développée dans le cadre de ce projet [3]. Nous avons évalué la fonctionalité de plusieurs outils de simulation existants dans le but de les intégrer à Macrotec. Ce rapport résume cette étude.

Les outils évalués sont les suivants:

1- GSPN [1]
2- SPNP [2]
3- Design/CPN [7]
4- Eval [6]
5- Voltaire [4]

Ce rapport compare ces outils selon une liste de critères prédéterminés tirés de l'analyse de besoins du volet simulation de l'outil Macrotec. Le lecteur pourra se référer à la section suivante pour une définition et discussion de ces critères. Le rapport conclue avec une recommandation quant à l'outil (ou les outils) apte à s'intégrer dans MACA.

Notez que nous supposons le lecteur familier avec la théorie des réseaux de Petri. Dans le cas contraire, le lecteur est prié de se référer à [5].

Comparaison fonctionnelle des outils

Critère	GSPN	SPNP	Design/CPN	Eval	Voltaire
I- Editeur graphique	x	-	x	?	?
II- Modélisation					
Type de réseau:					
- élémentaire	x	x	x	x	x
- hiérarchique	x	-	x	?	x
- coloré ou prédicat-transition	-	-	x	-	x
- place/transition	x	x	x	?	x
Temporisation des transitions:					
- déterministe	x	-	x	-	x
- exponentielle	x	x	x	x	x
- immédiate	x	x	x	-	x
- autre	-	-	x	-	-
- variable	x	x	x	?	-
Résolution de conflits entre transitions	x	x	x	?	x
Arcs inhibiteurs	x	x	x	?	-
"Guards" associés aux transitions	-	x	x	?	x

Critère	GSPN	SPNP	Design/CPN	Eval	Voltaire
III-Analyse					
Analyse quantitative:					
- analytique	x	x	-	?	-
- simulation	x	x	x	?	x
Etat du réseau:					
- transient	x	x	x	?	x
- à l'équilibre	x	x	-	?	-
Analyse qualitative	x	x	-	?	-
Animation temporelle	x	-	x	?	-
Génération automatique de résultats	x	x	-	?	x
IV-Accessibilité de la représentation interne	x	x	-	?	x

Terminologie

Modélisation

Un *reseau de Pétri hiérarchique* peut être découpé en plusieurs niveaux représentant différents degrés d'abstraction.

Un *réseau de Pétri coloré* ou *prédicat/transition* est un réseau dont les jetons représente différentes entités, à l'instar des variables.

Un *réseau place/transition* permet un nombre quelconque de jetons dans chacune des places.

La *temporisation* d'une transition peut être déterministe, immédiate, exponentielle ou autre. Une transition déterministe a une durée d'exécution fixe, soit x unités de temps. Une transition immédiate s'exécute instantanément, donc en un temps 0. Une transition dont la durée est exponentielle possède un temps d'exécution f(x) où f est une fonction exponentielle. Une transition peut avoir une durée d'exécution suivant une distribution quelconque. Une transition variable signifie que sa durée d'exécution peut varier selon les marquages.

Un *conflit* se produit lorsque deux ou plusieurs transitions se partagent les mêmes places d'entrée et qu'un nombre insuffisant de jetons se trouvent dans ces places pour que toutes les transitions se déclenchent à la fois. La résolution de conflits peut s'effectuer en associant aux transitions, des probabilités de déclenchement en cas de conflit.

Un arc *inhibiteur* permet d'inhiber une transition lorsque le nombre de jetons dans la place entrée est supérieure ou égal à x.

Un "*guard*" est une expression booléene associée à une transition. Cette expression doit être évaluée à vrai pour que la transition soit déclenchable.

Analyse

L' *analyse quantitative* permet d'analyser la performance dynamique du réseau en procurant des résultats tels que le débit moyen d'une action, le temps d'attente d'une transition et l'utilisation des ressources. La résolution du réseau peut se faire *analytiquement* ou par *simulation* (exécution du réseau).

De plus l'analyse quantitative peut être effectuée à l'*état d'équilibre* ou à partir d'un temps donné x (analyse *transiente*).

L'*analyse qualitative* permet de vérifier des propriétés générales de tous les comportements possibles du réseau (par exemple, l'absence de blocage).

L'animation temporelle est l'exécution graphique du réseau où les durées des transitions sont prises en considération.

Les résultats des analyses qualitatives et quantitatives doivent être générés automatiquement sans aucune programmation de la part de l'usager.

Représentation interne

La *représentation interne* d'un outil de simulation est la forme sous laquelle l'outil lit et sauvegarde la description du réseau. L'accessibilité de cette représentation permet l'échange de données entre différents outils.

Recommandations

GSPN possède toutes les caractéristiques désirées mais présente des problèmes d'installation.

Nous recommendons donc SPNP, qui se rapproche le plus des fonctionalités désirées et qui est disponible sur notre plate-forme. Il semble être fonctionnel et bien documenté.

Références

[]1 G.Chiola, "G*reat*SPN's Users' Manual, Version 1.3, Università di Torino, Septembre 1987.

[2] G.Ciardo, J.K.Muppala, "Manual for the SPNP Package Version 3.1, Duke University, Mars 1992.

[3] MACA, Rapport de la phase II, 1992.

[4] P.Parent, Voltaire Users'Manual, McGill University, 1990.

[5] R.Valette, "Les Réseaux de Pétri", L.A.A.S./C.N.R.S., Toulouse, Mai 1990.

[6] Stochastic Petri Net Solver Release 1.0, Verilog, 1990.

[7] DesignCPN, User Manual Version 2.0, Metasoft, 1992.