Structured Stochastic Modeling of Fault-Tolerant Systems*

Cristiano Bertolini   Leonardo Brenner   Paulo Fernandes†   Afonso Sales
Avelino F. Zorzo‡

Pontifícia Universidade Católica do Rio Grande do Sul
Av. Ipiranga, 6681 - 90619-900 - Porto Alegre, Brazil
{cbertolini, lbrenner, paulof, asales, zorzo}@inf.pucrs.br

Abstract

*Fault-tolerant mechanisms have been increasingly used to develop safety-critical systems in the past years. Therefore the accurate description of these mechanisms is crucial if we want that their use do not bring any kind of unexpected result due to the misinterpretation of their features. This paper presents a new way of precisely describing fault-tolerant mechanisms using a formalism that have a Markovian behavior. More specifically, we describe how to apply Stochastic Automata Networks (*SAN*) to describe a Dependable Multiparty Interaction (*DMI*) mechanism.*

1. Introduction

Parallel programs are usually composed of diverse concurrent activities, and communication and synchronization patterns between these activities are complex and not easily predictable. Thus, parallel programming is widely regarded as difficult. For instance, Foster [15] says that parallel programming is "more difficult than sequential programming and perhaps more difficult than it needs to be". In addition to the normal programming concerns, the programmer has to deal with the added complexity brought about by multiple threads of control: managing their creation and destruction, and controlling their interactions via synchronization and communication.

Furthermore, with the proliferation of distributed systems, computer communication activities are becoming more and more distributed. Such distribution can include processing, control, data, network management, and security [21]. Although distribution can improve the reliability of a system by replicating components, sometimes an increase in distribution can introduce some undesirable faults. It is important that this distribution is implemented in an organized way in order to reduce the risks of introducing faults, as well as the risks of coping with residual faults.

As in sequential programming, complexity in distributed, in particular parallel, program development can be managed *by providing appropriate programming language constructs*. Language constructs can help both by supporting encapsulation so as to prevent unwanted interactions between program components and by providing higher-level abstractions that reduce programmer effort by allowing compilers to handle mundane, error-prone aspects of parallel program implementation [15].

A language construct that encloses multiple processes executing a set of activities together is called a *multiparty interaction* [18, 13]. In a multiparty interaction, several executing processes somehow "come together" to produce an intermediate and temporary combined state, use this state to execute some joint activity, and then leave the interaction and continue their normal execution.

This paper uses a mechanism that integrates concurrent exception handling to the multiparty interaction concept. The extended multiparty interaction concept is called *dependable multiparty interaction* (DMI) [26] and it is able to cope with several concurrent exceptions being raised during the multiparty interaction. The DMI mechanism has been used to implement control software for several safety-critical systems [25, 27], but most of these control software were developed in an *ad hoc* manner.

Formal analysis of a system using DMI has also been developed [25], based on a Temporal Logic description of the mechanism [28]. Although this description gives a good insight of the DMI properties, it does not provide the system developer with a tool to analyse the system probabilities to reach faulty states.

Moreover, it is crucial that formal description of a system is used when designing complex safety-critical applications. This description should not only guarantee the pre-

* This work was developed in collaboration with HP Brazil R&D.
† Partially funded by CNPq/Brazil. Corresponding author. The order of authors is merely alphabetical.
‡ Partially funded by CNPq/Brazil.

Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04)
1526-7539/04 $20.00 © 2004 IEEE

Authorized licensed use limited to: Pontificia Universidade Catolica do Rio Grande do Sul (PUC/RS). Downloaded on June 10,2022 at 17:11:15 UTC from IEEE Xplore. Restrictions apply.

cise understanding of the mechanisms that are being applied in the design, but should also make it possible to get performance and reliability measures.

A stochastic modeling formalism seems the natural choice to formal description. A myriad of formalisms could be employed, but the nature of DMI suggests some desirable properties. The straightforward Markov Chain formalism [24] lacks from structure, nevertheless it offers efficient numerical tools. Stochastic Petri Nets [1] provides structure and clear synchronization primitives, however there is not a real modular concept and even partition techniques do not offer a sufficient modular approach to Stochastic Petri Nets [8, 12, 11]. We suggest the use of Stochastic Automata Networks (SAN) [23] which is being used to develop performance models to parallel and distributed computer systems [19, 5, 4]. The SAN formalism is usually quite attractive when modeling systems with several parallel cooperative activities. Another important advantages of the SAN formalism is the efficient numeric algorithms to compute stationary and transient measures [14, 2]. Those algorithms take advantage of the structured and modular definitions, allowing the treatment of considerably large models[1]. For all those reasons, we believe SAN is quite adequate to describe cooperative activities designed using DMI. However, other formalisms could also be employed with similar advantages, *e.g.*, PEPANETS [16].

This paper describes how to use SAN to model a system designed using DMI. Section 2 presents a quick introduction to the dependable multiparty interaction concept. In Section 3, we briefly describe the Stochastic Automata Networks formalism. Section 4 presents the description of DMI using SAN. In Section 5, we show an example of a SAN model for an application designed using DMI. Finally, in Section 6, we present the numerical results of the proposed model.

2. Dependable Multiparty Interactions

Several multiparty interaction mechanisms do not provide features for dealing with possible faults that may happen during the execution of the interactions. In some faults, the underlying system that is executing those multiparty interactions will simply stop the system in response to a fault. In DisCo [17], for instance, if an assertion inside an action is false, then the run-time system is assumed to stop the whole application. This situation is unacceptable in many situations.

1 The state of art in SAN lumpability [2] could not be employed to the proposed model in this paper due to different synchronizing events among similar automata. Actually, the algorithm proposed in [2] can only be applied to replicated automata.

A mechanism that brings together a way to handle exceptions during a multiparty interaction is the *Dependable Multiparty Interactions* (DMI) mechanism [26]. Specifically, a DMI is a multiparty interaction mechanism that provides facilities for *handling concurrent exceptions* and *assuring consistency upon exit*. Fig. 1 shows how exception handling is organized in a DMI . More details on how to use DMI can be found in [26, 27, 25].

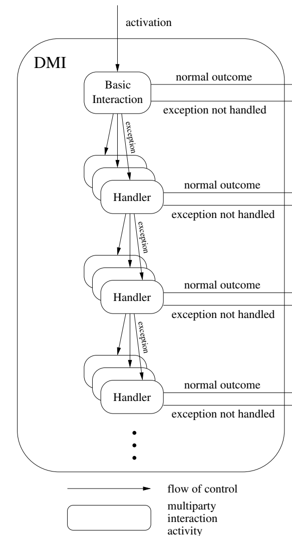

Figure 1. Dependable Multiparty Interactions

A DMI is represented by a set of *roles* which are executed by *players*. A player can activate a role in a DMI in order to execute the commands inside a role. A DMI only starts, when all its roles have been activated, and the guard (boolean expression) at the beginning of the DMI is *true*. A DMI only finishes, when all players have finished to execute their roles, and the assertion at the end of the DMI (boolean expression) is *true* (if no exceptions have been raised). Roles can only access data that are sent to them when they are activated, or data sent by other roles, belonging to the same DMI. Exceptions can be raised during the execution of a DMI. If exceptions are raised, then all roles that have not raised an exception are interrupted, and an exception resolution algorithm [9] is executed when all roles have either raised an exception or have been interrupted. If there is a handler to deal with the exception that was decided upon by the exception resolution algorithm, then this handler is activated by all roles. If there is no handler to deal with the exception that was decided upon by the exception resolution algorithm, then the exception is raised in the callers of all roles. Handlers have the same number of roles as the DMI to which they are connected.

Authorized licensed use limited to: Pontificia Universidade Catolica do Rio Grande do Sul (PUC/RS). Downloaded on June 10,2022 at 17:11:15 UTC from IEEE Xplore. Restrictions apply.

3. Stochastic Automata Networks

The SAN formalism was proposed by Plateau [22] and its basic idea is to represent a whole system by a collection of subsystems with an independent behavior (*local transitions*) and occasional interdependencies (*functional rates* and *synchronizing events*). The framework proposed by Plateau defines a modular way to describe continuous and discrete-time Markovian models [23]. However, only continuous-time SAN will be considered in this paper, although discrete-time SAN can also be employed without any loss of generality.

The SAN formalism describes a complete system as a collection of subsystems that interact with each other. Each subsystem is described as a stochastic automaton, *i.e.*, an automaton in which the transitions are labeled with probabilistic and timing information. Hence, one can build a continuous-time stochastic process related to SAN, *i.e.*, the SAN formalism has exactly the same application scope as Markov Chain (MC) formalism [24, 7]. The state of a SAN model, called *global state*, it is defined by the cartesian product of the *local states* of all automata.

There are two types of events that change the global state of a model: *local events* and *synchronizing events*. Local events change the SAN global state passing from a global state to another that differs only by one local state. On the other hand, synchronizing events can change simultaneously more than one local state, *i.e.*, two or more automata can change their local states simultaneously. In other words, the occurrence of a synchronizing event forces all concerned automata to fire a transition corresponding to this event. In fact, local events can be viewed as a particular case of synchronizing events that concerns only one automaton.

Each event is represented by an *identifier* and a *rate* of occurrence, which describes how often a given event will occur. Each transition may be fired as result of the occurrence of any number of events. In general, non-determinism among possible different events is dealt according to Markovian behavior, *i.e.*, any of the events may occur and their occurrence rates define how often each one of them will occur. However, from a given local state, if the occurrence of a given event can lead to more than one state, then an additional *routing probability* must be informed. The absence of routing probability is tolerated if only one transition can be fired by an event from a given local state.

The other possibility of interaction among automata is the use of functional rates. Any event occurrence rate may be expressed by a constant value (a positive real number) or a function of the state of other automata. In opposition to synchronizing events, functional rates are one-way interaction among automata, since it affects only the automaton where it appears.

Figure 2 presents a SAN model with two automata, four local events, one synchronizing event, and one functional rate. In the context of this paper, we will use roman letters to identify the name of events and functions, and greek letters to describe constant values of rates and probabilities.

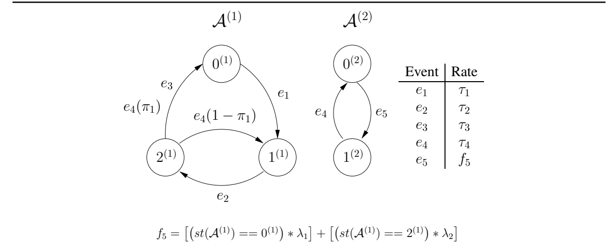

Figure 2. Example of a SAN model

In the model of Fig. 2, the occurrence of the event e_4 changes the automaton $\mathcal{A}^{(2)}$ from $1^{(2)}$ to $0^{(2)}$ state at the same time that the automaton $\mathcal{A}^{(1)}$ changes from $2^{(1)}$ to $0^{(1)}$ state with probability equal to π_1, or from $2^{(1)}$ to $1^{(1)}$ state with probability equal to $1 - \pi_1$. Notice that the rate of the event e_5 is not a constant rate, but a function rate called f_5. Due to event e_5, the firing of the transition from $0^{(2)}$ to $1^{(2)}$ state occurs with rate λ_1 (if automaton $\mathcal{A}^{(1)}$ is in $0^{(1)}$ state) or λ_2 (if automaton $\mathcal{A}^{(1)}$ is in $2^{(1)}$ state). If automaton $\mathcal{A}^{(1)}$ is in $1^{(1)}$ state, the transition from $0^{(2)}$ to $1^{(2)}$ state does not occur (event e_5 rate becomes equal to 0). Function f_5 in the SAN formalism is described through the notation[2] employed by the PEPS2003 tools [3]. Fig. 3 shows the equivalent Markov chain to Fig. 2.

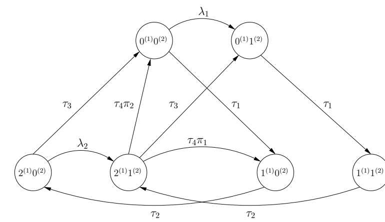

Figure 3. Equivalent Markov Chain to Fig. 2

The use of functional expressions is not limited to event rates. In fact, routing probabilities also may be expressed as functions. The use of functions is a powerful primitive of

2 The interpretation of a function can be viewed as the evaluation of an expression of non-typed programming languages, *e.g.*, C language. Each comparison is evaluated to value 1 (for *true*) and to value 0 (for *false*).

Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04)
1526-7539/04 $20.00 © 2004 IEEE

Authorized licensed use limited to: Pontificia Universidade Catolica do Rio Grande do Sul (PUC/RS). Downloaded on June 10,2022 at 17:11:15 UTC from IEEE Xplore. Restrictions apply.

SAN, since it allows to describe very complex behaviors in a very compact format. The computational costs to handle functional rates has decreased significantly with the developments of numerical solutions for the SAN models, *e.g.*, the algorithms for generalized tensor products [3].

4. SAN model for a DMI mechanism

This section describes a general manner to translate a DMI definition to a SAN model. The structure of a DMI definition composed of D DMIs, R roles, and P players (see Section 2) can be described by three subsets of generic automata called $DMI^{(k)}$ ($k = 1 \dots D$), $Role^{(j)}$ ($j = 1 \dots R$), and $Player^{(i)}$ ($i = 1 \dots P$).

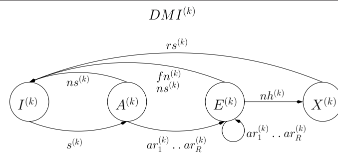

Figure 4. Generic DMI automaton

The $DMI^{(k)}$ automaton (Fig. 4) represents the kth DMI and it is composed of states: $I^{(k)}$ (*idle*) representing that the kth DMI is not needed at the moment; $A^{(k)}$ (*active*) indicating that the kth DMI has its players activated (synchronized), and they are waiting roles to be assigned; $E^{(k)}$ (*executing*) representing that the players are executing their roles; and $X^{(k)}$ (*abort exception*) representing that the kth DMI is waiting for external intervention to deal with a non-handable (global) exception.

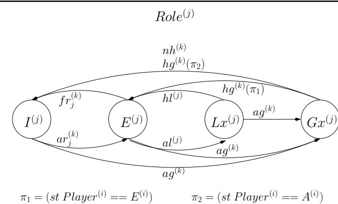

Figure 5. Generic $Role$ automaton

The $Role^{(j)}$ automaton (Fig. 5) represents the jth role and it is composed of states: $I^{(j)}$ (*idle*) indicating that the jth role has not been assigned to a player; $E^{(j)}$ (*executing*) representing that the jth role has been assigned to a player, which is performing some computation; $Lx^{(j)}$ (*local exception*) representing that the jth role is handling an exception locally, *i.e.*, without involving the other DMI's roles; and $Gx^{(j)}$ (*global exception*) indicating that the jth role is handling the exception cooperatively (with other DMI's roles).

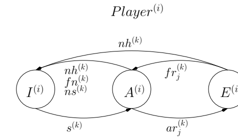

Figure 6. Generic $Player$ automaton

The $Player^{(i)}$ automaton (Fig. 6) represents the ith player and it is composed of states: $I^{(i)}$ (*idle*) representing that the ith player is not ready to work[3], *i.e.*, it is waiting for a DMI activation (synchronization); $A^{(i)}$ (*active*) indicating that the ith player is ready to work, but a role has to yet been assigned to it; and $E^{(i)}$ (*execution*) representing that the ith player is performing a role (including the handling of an exception).

A DMI state can be modified by the occurrence of events that affect the $DMI^{(k)}$ automaton, its role automata ($Role^{(j)}$), and all automata representing the players that can be assigned to those roles. Note that to each DMI, there are exclusive roles, but the players can be assigned to roles belonging to as many DMIs as defined by the system description.

The events in a subset coordinated by the $DMI^{(k)}$ are:

- $s^{(k)}$ - the DMI activation; this single event starts the DMI by synchronizing the passage from $I^{(k)}$ to $A^{(k)}$ in $DMI^{(k)}$ automaton with the passage from $I^{(i)}$ to $A^{(i)}$ in all automata representing players coordinated by $DMI^{(k)}$;
- $ns^{(k)}$ - synchronizes the passage from $A^{(k)}$ to $I^{(k)}$ or $E^{(k)}$ to $I^{(k)}$ when the guard or the assertion of the $DMI^{(k)}$ is not satisfied;
- $fn^{(k)}$ - finalizes the execution of the $DMI^{(k)}$ and all their players;
- $ar_j^{(k)}$ - the assignment of the $Role^{(j)}$ to one of the players; there are as many ar_j events as there are roles to be assigned in the $DMI^{(k)}$, and each of them synchronizes the beginning of execution in the automata representing the player and the role;
- $fr_j^{(k)}$ - finalizes the execution of the $Role^{(j)}$ and synchronizes the passage from $E^{(k)}$ to $A^{(k)}$ of the player;
- $al^{(j)}$ - a local exception is raised in the $Role^{(j)}$;
- $hl^{(j)}$ - a local exception in the $Role^{(j)}$ is handled;

3 This could also indicate that the player is executing some computation that does not involve cooperative work.

1526-7539/04 $20.00 © 2004 IEEE

Authorized licensed use limited to: Pontificia Universidade Catolica do Rio Grande do Sul (PUC/RS). Downloaded on June 10,2022 at 17:11:15 UTC from IEEE Xplore. Restrictions apply.

- $ag^{(k)}$ - sinalizes a global exception to all concerned roles in the $DMI^{(k)}$;
- $hg^{(k)}$ - handles a global exception in the $DMI^{(k)}$ and return the concerned roles to their previous states;
- $nh^{(k)}$ - passes the $DMI^{(k)}$ from $E^{(k)}$ to $X^{(k)}$ (non-handled exception) and set all their players and roles as inactive;
- $rs^{(k)}$ - reset the execution of the $DMI^{(k)}$ after a non-handled global exception was raised.

5. Case study: Fault-Tolerant Production Cell

In this section, we used a safety-critical system to show how to model a system designed with DMI using the SAN formalism. The system we have used is a Fault-Tolerant Production Cell [25] which consists of six devices: two conveyor belts (a feed belt and a deposit belt), an elevating rotary table, two presses, and a rotary robot that has two orthogonal extendible arms equipped with electromagnets (see Fig. 7). These devices are associated with a set of sensors that provide useful information to a controller and a set of actuators via which the controller can exercise control over the whole system. The task of the cell is to get a metal blank from its "environment" via the feed belt, transform it into a forged plate by using a press, and then return it to the environment via the deposit belt. More precisely, the production cycle for each blank is:

1. If the traffic light for insertion shows green, a blank may be added, *e.g.*, by the blank supplier, to the feed belt from the environment;
2. The feed belt conveys the blank to the table;
3. The table rotates and rises to the position where the magnets of the robot are able to grip the blank;
4. Arm 1 of the robot picks the blank up and places it into an unoccupied press, either press 1 or press 2;
5. The chosen press forges the blank;
6. Arm 2 of the robot removes the forged plate from the press and places it on the deposit belt; and
7. If the traffic light for deposit is green, the plate may be forwarded further and carried to the environment where a container may be used, *e.g.*, by the blank consumer, to store the forged pieces.

Normally, both presses are used and a certain amount of interleaving of two such production cycles, one for each press, is possible. A correct control program must satisfy certain requirements specified by the Fault-Tolerant Production Cell model, *e.g.*, safety, liveness, and failure detection and continuous service. Other requirements, such as flexibility and efficiency, may be taken into account, but must not conflict with the previous requirements.

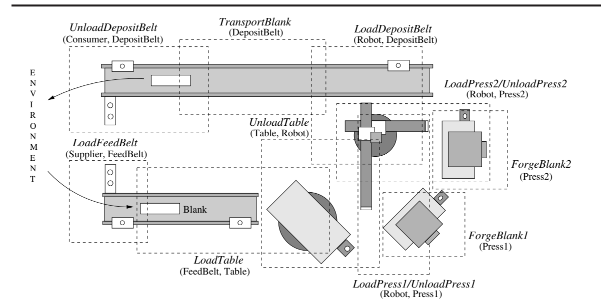

Figure 7. Fault-Tolerant Production Cell

5.1. Proposed model

To validate the ideas proposed in Section 4, we modeled a control software for this production cell using SAN. Although the whole design of a control software for the Production Cell is composed of twelve DMIs, the SAN model for this case study represents only the DMIs that are critical in the system, *i.e.*, the DMIs that involve the robot and the forging of blanks. The proposed SAN model contains seven DMIs (*UnloadTable*, *LoadPress1*, *ForgeBlank1*, *UnloadPress1*, *LoadPress2*, *ForgeBlank2*, and *UnloadPress2*) and four *players* (*Table*, *Robot*, *Press1*, and *Press2*). Then the problem size (product state space) becomes considerably smaller (2.23×10^{13}) than the original model with twelve DMIs (4.84×10^{23}). Starting from this reduced model, which is still very large, we can use algebraic aggregation [3] reducing the problem to a reasonable size (2.05×10^{6}).

Although a DMI may contain several participants, in the case study used in this paper, the DMIs will contain only one or two participants. As can be seen in Fig. 7, a DMI encloses the control of a sequence of operations between devices (players). Each DMI encloses a set of devices that must interact in a coordinated fashion to satisfy the safety and fault tolerance requirements of the case study. If two DMIs overlap, they cannot be performed in parallel because they both involve the same device. For instance, *UnloadTable* cannot be executed in parallel with *LoadPress1* because both DMIs involve the robot, and the robot can participate in only one of them at a time.

The obvious interaction among DMIs is represented by possible assignment of roles from different DMIs to a same player. Such interaction exclude all control of the parallelisation possibilities to a DMI description. Actually, the absence of such interaction implies a fully concurrent execution of all DMIs, *i.e.*, tasks can be performed in any order. Many real applications require synchronization of tasks both inside a DMI and among different DMIs.

The synchronization among roles the same DMI is represented by *event superposition*, *i.e.*, the replacement of two events that must happen in a specific order by a single event.

Authorized licensed use limited to: Pontificia Universidade Catolica do Rio Grande do Sul (PUC/RS). Downloaded on June 10,2022 at 17:11:15 UTC from IEEE Xplore. Restrictions apply.

If the jth role must be performed just before the execution of the lth role, such synchronization can be represented by the replacement of the finishing event of $Role^{(j)}$ ($fr_j^{(k)}$) and the assignment of $Role^{(l)}$ ($ar_l^{(k)}$) by a single event called $sr_{jl}^{(k)}$. The use of event superposition assures not only the precedence of the jth role over the lth role, but it also forces the succession of these roles. Note that such synchronization has been applied to this case study, since its DMIS have a specific order to be executed, *e.g.*, in the *Unload-Table* DMI, the table has to be elevated and rotated before the robot can grab the metal plate from the table.

The synchronization among DMIS have the same nature as the synchronization among roles. Nevertheless, the representation in the SAN model need to be a little bit more sophisticated. Synchronization among different DMIS not only affect all DMIS automata, but also the concerned players automata. Regarding DMIS automata, we also use an event superposition technique, that works in a similar manner as for the synchronization among roles. For instance, if the kth DMI must precede the lth DMI, the ending event of the $DMI^{(k)}$ ($fn^{(k)}$) and the starting event of the $DMI^{(l)}$ ($s^{(l)}$) must be replaced by a single event called $sd^{(kl)}$. Notice that the events $fn^{(k)}$ and $s^{(l)}$ also synchronize the players used by the DMI tasks. Therefore, the event superposition must take that into account, specially, when a same player is used by both synchronized DMIS. In that particular case, the player must stay in the active state ($A^{(i)}$) after the end of the kth DMI, in order to be ready to the execution of the lth DMI. The automaton of the player ($Player^{(i)}$) concerned by the sequence of DMIS must include an additional loop transition in the active state ($A^{(i)}$). This loop transition allows that a player executes roles in different DMIS in an ordered sequence.

Another possibility of interaction among DMIS is the message exchange, which can be easily described in a SAN model through the use of functional rates. Such functional rates depend on the state of other DMIS in order to allow, or not, the occurrence of events. The adequation of the SAN formalism is once again clear, since the inclusion of functional rates keeps the same automata structure into the model.

6. Reliability and performance indices

In Section 5, SAN was used to model a control system using DMIS. This model was executed in the PEPS2003 tools [3] to analyse the behavior of the system. This section presents some of the indices obtained from the execution of the PEPS2003 tool.

The system was modeled with three basic configurations. First the production cell was executed with only one functioning press and the other press was not used in the production cell (*one-press*). The second configuration considered the usage of a primary press and a backup press (*backup-press*). The backup press is only activated when the primary press is broken. And finally, the third configuration sets the system to use both presses equally, thus both presses can be used in parallel (*two-presses*).

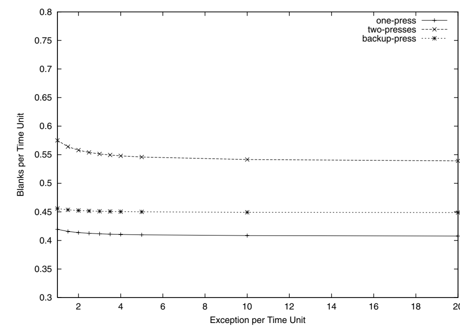

Figure 8. Throughput *vs.* Exception Rate

The first case study considers the SAN model regarding a steady state system, *i.e.*, when an abort exception occurs, the system reboots and the service restarts. An example of such system recovery may be the replacement of a faulty press by a new press by the operator.

Considering this situation, Fig. 8 shows the system throughput (blanks processed per minute) upon several exceptions being raised (exceptions per time unit). The exceptions can be from one of three kinds:

- exceptions that can be dealt locally ($Role^{(j)}$ is at state $Lx^{(j)}$);
- exceptions that can be dealt globally ($Role^{(j)}$ is at state $Gx^{(j)}$); and
- exceptions that can not be dealt and therefore it takes $DMI^{(k)}$ to state $X^{(k)}$ (abort exception).

Notice that the system throughput with two presses working at the same time is not twice the throughput with just one press, because the robot is a bottleneck to the system. The increase in the number of exceptions does not affect the system throughput considerably. While the number of exceptions per time unit increases ten times, the loss in the system throughput is only about 10% for all configurations, *i.e.*, when there are two active presses. For example, with two-presses configuration this happens because the robot is frequently involved in the handling of global exceptions in a DMI, and therefore the enrolment of the robot player in another DMI may be delayed.

Authorized licensed use limited to: Pontificia Universidade Catolica do Rio Grande do Sul (PUC/RS). Downloaded on June 10,2022 at 17:11:15 UTC from IEEE Xplore. Restrictions apply.

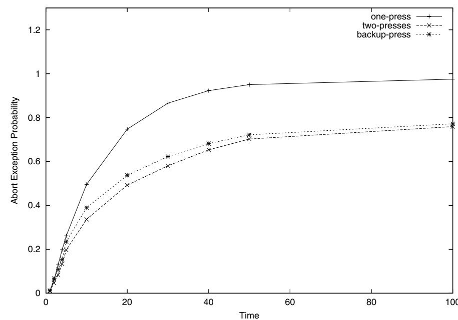

Figure 9. Abort Exception Probability *vs.* Elapsed Time

Fig. 9 depicts the results of the transient analysis of the system, *i.e.*, it represents the situation in which a DMI entering state $X^{(k)}$, stays in state $X^{(k)}$. The results presented in the figure consider rare exceptions (one exception being raised per time unit). Obviously, the longer the system stays executing, the bigger the probability to get an abort exception will be.

It is interesting to notice that the one-press configuration rapidly reaches a high abort exception probability. The behavior of the other two configurations is almost exactly the same, since the full system stop (abort exception in both presses) remains equal. Such result is comprehensible, since in the backup-press configuration, the overall press use (sum of the use of the primary and the backup presses) is the same as in the two-presses configuration. In fact, the probability of abort exception in the primary press will be considerably higher than the probability of the backup press. Nevertheless, the average for those two probabilities will be quite similar to the probability of abort exception in both presses for the two-presses configuration.

As general conclusion, we may argue that the two-presses configuration is a better choice. In a first approach, such conclusion seems obvious. Nevertheless, the numerical results for a specific, and well parameterized, case may furnish a deeper analysis, *e.g.*, the additional cost of a second press (and/or its maintenance) may be justified, or not, by the throughput increase.

7. Conclusion

The main contribution of this paper is to show a feasible way to conjugate the specification power of a DMI description and the performance evaluation capacities of a SAN model. Starting from a DMI specification, it is possible to obtain a large, but tractable, equivalent SAN model. From this model we could compute stationary and transient measures that may furnish useful performance information about the fault-tolerant system.

Note that the results shown in the previous section do not add a particular understanding about the modeled system (Fig. 7). It seems quite evident to any experienced fault-tolerant specialist that the two-presses option (active redundancy) has better performance (higher throughput in Fig. 8) than the other options, *i.e.*, one-press (no redundancy) and backup-press (passive redundancy). However, it was a little bit more surprising to see an almost equal behavior for the abort exception probability (Fig. 9). In fact, we must insist that the purpose of our paper is not to analyse a particular fault-tolerant example, but to show the benefits from the conjugation of the DMI description and the SAN performance evaluation.

One of the main concerns in the model development phase is the (usually huge) size of the SAN model. A DMI description with D DMIS, R roles, and P players represents a SAN model with $D+R+P$ automata, up to $7D+4R$ events, and a product state space of $4^D \times 4^R \times 3^P$ states. Although the particular synchronization of roles could reduce considerably the number of events, the SAN model remains with a quite large product state space. Fortunately, the natural restrictions of any DMI description reduce this product state space to a much smaller reachable state space. Consequently, it is vital to consider an efficient way to deal with very sparse representations.

The use of Generalized Tensor Algebra (GTA) [6] is quite efficient to the representation and storage of models, but other techniques, such as Matrix Diagrams [20], can be also employed to an even more efficient generation of the state space. The modular nature of the DMI description and the number of distinct events seem determinant to justify the choice of the SAN formalism. For example, a Stochastic Petri Nets (SPN) representation of an equivalent SAN model must represent a near intractable number of transitions in a single net. However, it is important to notice that SPN models have a powerful software tool, called SMART [10], which allows to obtain performance indices even for such impressively large product state spaces.

The natural future work for is a careful study about the information that can be extracted from a SAN model constructed from a DMI specification. The modeling experience presented in this paper was just an initial attempt, and we believe that the current SAN modeling technique has much more information to offer. A comparison with actual reality measures may also help the development of a further comprehension of SAN models benefits. Another interesting future work can include some extensions to SAN numerical tools in order to facilitate the solution of models with very large product state space, and relatively small reachable state space. Such numerical improvements will allow the solution of a wider class of DMI descriptions.

1526-7539/04 $20.00 © 2004 **IEEE**

Authorized licensed use limited to: Pontificia Universidade Catolica do Rio Grande do Sul (PUC/RS). Downloaded on June 10,2022 at 17:11:15 UTC from IEEE Xplore. Restrictions apply.

References

[1] M. Ajmone-Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley and Sons, 1995.

[2] A. Benoit, L. Brenner, P. Fernandes, and B. Plateau. Aggregation of Stochastic Automata Networks with replicas. *Linear Algebra and its Applications*, 386:111–136, July 2004.

[3] A. Benoit, L. Brenner, P. Fernandes, B. Plateau, and W. J. Stewart. The PEPS Software Tool. In P. Kemper and W. H. Sanders, editors, *Computer Performance Evaluation / TOOLS 2003*, volume 2794 of *Lecture Notes in Computer Science*, pages 98–115, Urbana, IL, USA, 2003. Springer-Verlag Heidelberg.

[4] C. Bertolini, A. G. Farina, P. Fernandes, and F. M. Oliveira. Test Case Generation using Stochastic Automata Networks: Quantitative Analysis. In *Proceedings of the* 2nd *IEEE International Conference on Software Engineering and Formal Methods*, Beijing, China, September 2004.

[5] L. Brenner, L. G. Fernandes, P. Fernandes, and A. Sales. Performance Analysis Issues for Parallel Implementations of Propagation Algorithm. In *Proceedings of the* 15th *Symposium on Computer Architecture and High Performance Computing*, pages 183–190, São Paulo, November 2003.

[6] L. Brenner, P. Fernandes, and A. Sales. Why you should care about *Generalized Tensor Algebra*. Technical Report TR 037, PUCRS, Porto Alegre, 2003. http://www.inf.pucrs.br/tr/tr037.pdf.

[7] L. Brenner, P. Fernandes, and A. Sales. The Need for and the Advantages of Generalized Tensor Algebra for Kronecker Structured Representations. In 20th *Annual UK Performance Engineering Workshop*, pages 48–60, Bradford, UK, July 2004.

[8] P. Buchholz. An Adaptive Decomposition Approach for the Analysis of Stochastic Petri Nets. In *International Conference on Dependable Systems and Networks (DSN'02)*, pages 647–656, Washington, DC, USA, June 2002. IEEE CS-Press.

[9] R. H. Campbell and B. Randell. Error recovery in asynchronous systems. *IEEE Transactions on Software Engineering*, 12(8):811–826, 1986.

[10] G. Ciardo, R. L. Jones, A. S. Miner, and R. Siminiceanu. Logical and stochastic modeling with SMART. In P. Kemper and W. H. Sanders, editors, *Computer Performance Evaluation / TOOLS 2003*, volume 2794 of *Lecture Notes in Computer Science*, pages 78–97, Urbana, IL, USA, 2003. Springer-Verlag Heidelberg.

[11] G. Ciardo and K. S. Trivedi. A Decomposition Approach for Stochastic Petri Nets Models. In *Proceedings of the* 4th *International Workshop Petri Nets and Performance Models*, pages 74–83, Melbourne, Australia, December 1991. IEEE Computer Society.

[12] S. Donatelli. Superposed generalized stochastic Petri nets: definition and efficient solution. In *Proceedings of the* 15th *International Conference on Applications and Theory of Petri Nets*, pages 258–277, Springer-Verlag Heidelberg, 1994. R. Valette.

[13] M. Evangelist, N. Francez, and S. Katz. Multiparty interactions for interprocess communication and synchronization. *IEEE Transactions on Software Engineering*, 15(11):1417–1426, 1989.

[14] P. Fernandes, B. Plateau, and W. J. Stewart. Efficient descriptor - Vector multiplication in Stochastic Automata Networks. *Journal of the ACM*, 45(3):381–414, 1998.

[15] I. Foster. Compositional parallel programming languages. *ACM Transactions on Programming Languages and Systems*, 18(4):454–476, 1996.

[16] S. Gilmore, J. Hillston, L. Kloul, and M. Ribaudo. PEPA nets: a structured performance modelling formalism. *Performance Evaluation*, 54(2):79–104, 2003.

[17] H.-M. Jarvinen and R. Kurki-Suonio. DisCo specification language: Marriage of actions and objects. In *11th International Conference on Distributed Computing Systems*, pages 142–151. IEEE CS Press, 1991.

[18] Y.-J. Joung and S. A. Smolka. A comprehensive study of the complexity of multiparty interaction. *Journal of ACM*, 43(1):75–115, 1996.

[19] R. Marculescu and A. Nandi. Probabilistic Application Modeling for System-Level Performance Analysis. In *Design Automation & Test in Europe (DATE)*, pages 572–579, Munich, Germany, March 2001.

[20] A. S. Miner and G. Ciardo. A data structure for the efficient Kronecker solution of GSPNs. In *Proceedings of the* 8th *International Workshop on Petri Nets and Performance Models*, pages 22–31, Zaragoza, Spain, September 1999.

[21] P. G. Neumann. Distributed systems have distributed risks. *Communications of the ACM*, 39(11):130, 1996.

[22] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. In *Proceedings of the 1985 ACM SIGMETRICS conference on Measurements and Modeling of Computer Systems*, pages 147–154, Austin, Texas, USA, 1985. ACM Press.

[23] B. Plateau and K. Atif. Stochastic Automata Networks for modelling parallel systems. *IEEE Transactions on Software Engineering*, 17(10):1093–1108, 1991.

[24] W. J. Stewart. *Introduction to the numerical solution of Markov chains*. Princeton University Press, 1994.

[25] J. Xu, B. Randell, A. Romanovsky, R. J. Stroud, A. F. Zorzo, E. Canver, and F. von Henke. Rigorous Development of an Embedded Fault-Tolerant System Based on Coordinated Atomic Actions. *IEEE Transactions on Computers*, 51(2):164–179, 2002.

[26] A. F. Zorzo. *Multiparty Interactions in Dependable Distributed Systems*. PhD thesis, University of Newcastle upon Tyne, Newcastle upon Tyne, UK, 1999.

[27] A. F. Zorzo, A. Romanovsky, B. R. J. Xu, R. J. Stroud, and I. S. Welch. Using Coordinated Atomic actions to design safety-critical systems: A production cell case study. *Software: Practice and Experience*, 29(8):677–697, 1999.

[28] A. F. Zorzo, A. Romanovsky, and B. Randell. *TLA Specification of a Mechanism for Concurrent Exception Handling*, pages 41–59. Kluwer Publ., Holland, 2002.

1526-7539/04 $20.00 © 2004 **IEEE**

Authorized licensed use limited to: Pontificia Universidade Catolica do Rio Grande do Sul (PUC/RS). Downloaded on June 10,2022 at 17:11:15 UTC from IEEE Xplore. Restrictions apply.