FACULDADE DE INFORMÁTICA
PUCRS – Brazil
http://www.inf.pucrs.br

A Comparison between Usage Modeling Formalisms for Statistical Software Testing

André Gobbi Farina and Flávio Moreira de Oliveira

TECHNICAL REPORT SERIES

Number 023
March, 2002

Contact:

farina@cpts.pucrs.br

http://www.inf.pucrs.br/~farina

flavio@cpts.pucrs.br

http://www.inf.pucrs.br/~flavio

André Gobbi Farina is a post-graduate student of PPGCC at PUCRS/Brazil. He is a member of the CPTS project since 2000. He develops research in statistical software testing.

Flávio Moreira de Oliveira works at PUCRS/Brazil since 1992. He is professor and coordinator of the CPTS project (technical cooperation agreement PUCRS-HP). He develops research in software testing and multiagent systems.

Published by PPGCC – FACIN – PUCRS
Av. Ipiranga, 6681
90619-900 Porto Alegre – RS – Brazil

LIST OF ABREVIATIONS

CPTS – Centro de Pesquisa em Teste de Software

CTMC – *Continuous Time Markov Chain*

DTMC – *Discrete Time Markov Chain*

HP – *Hewlett-Packard*

MTBF – *Mean Time Between Failures*

MTTF – *Mean Time to Failure*

SAN – *Stochastic Automata Network*

1. INTRODUCTION

In the software testing area, aiming to provide information about the system's reliability, the statistical testing of software sprung up. This technique combines elements of statistics to software test techniques, allowing the collection of measures about the system.

The Software Statistical Testing makes use, most of the times, of usage models, mappings of the typical ways the user can employ the system, as a mark in the process. In this work we focus on the statistical test using usage models. These models use as typical formalism the Markov chains.

The Markov chains bring several limitations, both in modeling and computing terms. On the other hand, a recent formalism called Stochastic Automata Networks shows several characteristics that may compensate the limitations show by the preceding formalism**.**

In this context, this work aims to make a study in terms of usage models, making use of a documents editor, the DocsEditor that will be introduced later, as target tool. From this tool, usage models were built with the use of the two formalisms above, aiming to establish a comparative analysis in terms of modeling, both in terms of the process and in the potential of the model itself.

In the first chapters we will discuss the basic concepts in terms of modeling formalisms and statistical testing of software. The fourth chapter shows the target tool, in terms of its characteristics and functions. In the fifth chapter we will show a report of the experimentation process, describing the usage model process using each one of the formalisms as well as the diagrams and other elements of the generated models themselves. Afterwards, we will show the conclusions resulted from the comparative analysis.

2. STOCHASTIC MODELS

It is possible for us to represent the behavior of a system describing all the different states it may show and indicating the possible transitions from one state to another during its execution. This system can be represented as a markovian process when the time spent in each state appears exponentially distributed. A collection (set) of states is associated to this markovian process. This system can assume only one state at any time. Several concepts of this chapter were extracted from [Stewart 94].

In this context, this chapter shows two system modeling formalisms: the Markov Chains and the Stochastic Automata Networks, discussing some of their main characteristics and usages. The formalism of the Markov Chains was chosen due to the fact that it constitutes a reference in the researched papers [Whittaker 94] [Sayre 99a] [Fosse 95] [Marre 95] [Sayre 99b]. On the other hand, the Stochastic Automata Network formalism is our study target, aiming to establish a sort of comparison to the other formalism.

2.1 Markov Chains

A stochastic process is defined as a family of random variables defined in a space of probabilities and indexed by a parameter. This parameter, in general, refers to an index set of the process time, or time gap. If we have discrete time (ex: $T = \{1,2,3...\}$) we have a discrete stochastic process, while if we have continuous time (ex: $T = 0 < t < \infty$) we have a continuous stochastic process. A markovian process is a stochastic process whose probabilities distribution function executes the fundamental property of the markovian process, in other words, the evolution of the process depends exclusively on the current state that it shows, not depending on the states previously shown.

The evolution of the system is represented by transitions of the process from one state to another. These transitions are assumed to happen in an instantaneous way (without time consuming).

When the space of states of a markovian process is discrete, this process is called a Markov Chain. These chains divide, according to the time scale, in Discrete Time Markov Chain (DTMC) and Continuous Time Markov Chain (CTMC).

In the discrete time chains (DTMC), we have conditional probabilities of transitions from one state to another. These probabilities of transition of the Markov Chain can be represented as *Pij* (probability of transition from the *i* state to the *j* state) in homogenous chains – whose transitions do not depend on time -, or as *Pij(n)* (probability of transition from the *i* state to the *j* state in the *n* time) in the non-homogenous chains – whose transitions depend on time. These probabilities are represented by a real number between 0 and 1, and the sum of all the probabilities of transition from one state to each one of the other states must be 1.

The probabilities of transition of the Markov Chain are represented through matrices with dimensions *n x n*, with *n* being the number of states of this chain. In the discrete time chains, this matrix is called matrix of transition probabilities, and in the continuous time chains it is called matrix of transition rates.

Through the system model under the shape of a Markov Chain, one can know the probabilities of being in a certain state – or set of these – in a certain moment posterior to the beginning of the process, estimate how much time is needed to reach a certain state for the first time, and several other measures.

The analysis of the stochastic process is said to be stationary when it analyzes the characteristic statistics of the model in a way not depending on the time *t* in which its observation is started, i.e., when the process doesn't shift, being considered a time close to infinite. On the other hand, the transitory analysis does the analysis of a state considering another previous state, i.e., the probabilistic analysis of a course in the space of states.

2.2 Stochastic Automata Networks

A Stochastic Automata Network (SAN) [Fernandes 98] consists of a number of individual stochastic automata that operate in a somewhat independent way. Each automaton is represented by a certain number of states, along with rules or

probability functions that rule the movements from one state of the automata to the other. The state of an automaton in a certain time *t* is the state that it occupies in the time *t*. The state of the automata network is determined by the state that each one of the constituent automata occupies in this same time *t*. [Stewart 94].

Stochastic automata networks show an important methodology of system modeling that allows the combination of several model techniques, such as Petri Nets, Queuing Networks, and others. [Stewart 94]

The main advantage of using Stochastic Automata Networks has to do with the fact that this formalism automatically generates its transition matrix in an implicit way through tensorial algebra [Stewart 95]. The non-explicit generation of the transition matrix constitutes a significant gain in accordance with the use of memory.

As seen in [Plateau 96], the increase of the systems complexity has made the space of states of the models (especially the Markov chains) so large, that their analysis becomes impracticable. The fact of parallel and distributed systems being generally seen as collections of components that operate with certain independence ends up approximating the problem to the formalism of the SAN. The problem of the explosion of the space of states is minimized in models using Stochastic Automata Networks, for instead of the creation of a transition matrix, smaller matrices are created (one for each automata), searching the relevant information from these matrices.

The transition rates from one state to another, in the Stochastic Automata Networks are represented in the model's arcs themselves, being exponentially (in the case of continuous time) or geometrically (in the case of discrete time) distributed. There are two kinds of transition rates:

- Fixed rate: as in the Markov chains
- Functional or Net-depending rate: the rate is a function of the states from other automata.

The transitions represented by the model's arcs are caused by events. These events, whose rates can be either fixed or functional, can be divided in two kinds:

- Local Event: affects only the automaton to whom it belongs;

- Synchronizing Event: transitions that occur simultaneously in the other automata, triggered by the event.

The stochastic automata networks can have a certain degree of interaction among their automata, these interactions being represented by the functional rates and by the synchronizing events. They can show the two kinds of transition. In the case of functional rates, they must be evaluated (calculated) at each use, through their definition formula. On the other hand, the synchronizing events affect the whole network, causing an event in an automaton to simultaneously trigger an event in other automata, synchronizing the interactivity between them.

3. USAGE MODELS

In the software testing area, there are several techniques of test case generation that aim to find, in the program, fail that lead to correction of errors in its codification. The main techniques are the functional testing, that aims to test the system behavior based on its specification, and the structural testing, that aims to test the application based in its source-code. [Myers 79] These techniques, through the analysis of the program entries, the traveled paths and exits, aim to analyze the maximum of possible fail situations. However, such exhaustive situation is not always possible, due to the high cost and even to the infinite amount of possibilities that it shows.

3.1 Statistical Software Testing

In face of this situation, when it comes to selecting the priorities and even "what will be tested", we don't find a trustable support in these techniques. In this line of thought, it's not possible to infer about the reliability degree of the tests. Therefore, the software statistical testing associates the statistical science to the solution of these problems [Sayre 99a]. Its main advantage is allowing the use of statistical inference techniques to compute probabilistic aspects of the testing process, like its reliability, mean time to failure (MTTF) mean time between failures (MTBF) etc [Whittaker 94].

Besides this, the statistical testing maps all the possible uses of the software, through the creation of usage models [Sayre 99a], which will be discussed in the next section.

In [Sayre 99], we can see the typical process of statistical testing. This process is divided in:

- **Get Specification**: the usage model must be developed from the specification of the correct behavior of the system. It can be defined through formal specification, documentation of the requirements, user's manual, prototype etc
- **Develop Model Structure**: identification of the states and transition arcs between them, through a manual process.

- **Assign Probabilities:** the probabilities of transition between the states of the model are manually attributed or automatically calculated from the usage analysis.
- **Verify and Analyse Model:** this is where we make calculations aiming to support the planning of the tests, to validate the model and so on.
- **Run Non-random Tests**: generation of test cases, to cover all the arcs of the model, following the order of the occurrence.
- **Run Random Tests**: generation of random tests, from the model. These tests can be manually or automatically executed.
- **Estimate Reliability:** analysis of the random testing log, aiming to estimate the reliability from the logged failures and the states in which they occurred. An example of this log is the testing chain in [Whittaker 94].
- **Decide whether to Stop Testing:** consists of the evaluation of the testing log, deciding when the test should stop or proceed.
- **Stop and Report Results:** after the tests are finished, their results can be used to decide about the releasing of a product, to evaluate the control degree of the developing process, to evaluate the performance of new elements, integrated to the system, and several other uses.

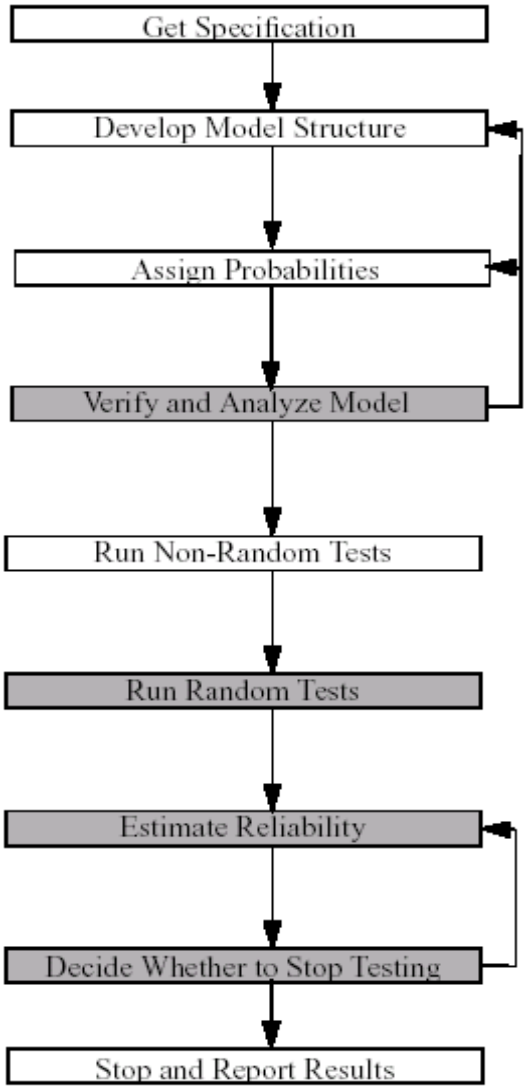


Figure 3.1 – Statistical Testing Process

3.2 Usage Models

A usage model of software characterizes the operational use of a software system. The software is used by a user in a certain environment [Trammell 95]. The user can be a person, a hardware device, or another system, the users being stratified if necessary. Software usage could be a working session, a transaction, or any unit of service limited by a start/end event, which ends up defining a usage instance. A usage environment can be characterized by platforms, mono-user/multi-user, concurring applications etc.

The structure of a usage model is formed by a group of states and by the transitions between them, which form a graph. The nodes of this graph represent the states of the model, its arcs representing the transitions between the states. This structure represents the possible uses of the software. A distribution of probability is applied to this structure, so that it represents the expected use of the software.

To represent the usage model, we make use of formalism – usually the Markov Chains, discussed in chapter 2 – of systems modeling. In this formalism, the states of use of the software are represented by states of the chain, the user actions being mapped in the transitions between the states of the chain. There are still two special states, the *Invoke* – which represents the beginning of the execution – and the *Terminate* – which represents the end of the execution, these two being the only initial and final states of the chain, respectively. The transition probabilities represent the probability of action by the user, configuring the typical uses of the software. [Sayre 99b]

These formal models allow the test engineer – responsible for the creation and orientation of the tests – to become aware of the critical ways, i.e., more likely to fail, concentrating the efforts of the test in this niche. This analysis is made from the occurrence probabilities, combined with each use of the software.

The analysis of the usage model allows the extraction of several information about the model, such as: [Trammel 95]

- Amount of statistically typical usage paths of the software;
- Long-term occupation, such as the time utilization percentages in each state;
- Average amount of events in a test case;
- Average amount of events between two states;
- Others.

So, the usage model can be used in different stages of the software's life cycle, such as refine the specification, evaluate the complexity, conduct the verification efforts, identify the frequency of determined events, plan the testing schedule, and infer about the software's reliability.

4. GENERIC FORMS BASED DOCUMENTS EDITOR

The Generic Forms Based Documents Editor is a tool that allows the edition of structured, object-oriented, forms based documents. The DocsEditor, as it is called, was developed in the context of the CPTS project, a scientific cooperation agreement established between Hewlett-Packard Brasil and PUCRS in the software testing area. In this case, a document is composed of a set of form objects. Each form leads the user to keep the text and the information in a pre-defined style and format. The application's interface is shown in pictures 4.1a (structure) and 4.1b (screen).

Some of the advantages of using this tool are:

- Easiness in keeping a team using the same format and the same style in documents editing;
- Easiness in identifying information fragments in the generated document;
- Easiness in reusing document parts, once the document is stored as an aggregation of document parts, instead of being stored as a whole.

A document, to the Docs Editor, is an instance of a document type. A document type describes the types of forms that a document, in a certain class, must contain, and the hierarchical relations between these forms (tree structure). The structure of a document is always hierarchical, allowing subsections arranged in sections, in as many levels as is it is necessary.

A document instance is made of a tree structure instance (which includes the list of all used forms) and the content of the forms. When the document is loaded, the editor is able to load all used forms, so to allow its edition.

<table>
<tr><td colspan="2">Title/document id area</td></tr>
<tr><td colspan="2">Menu area</td></tr>
<tr><td>Tree area</td><td>Forms area</td></tr>
</table>

Figure 4.1a –DocsEditor Interface Structure

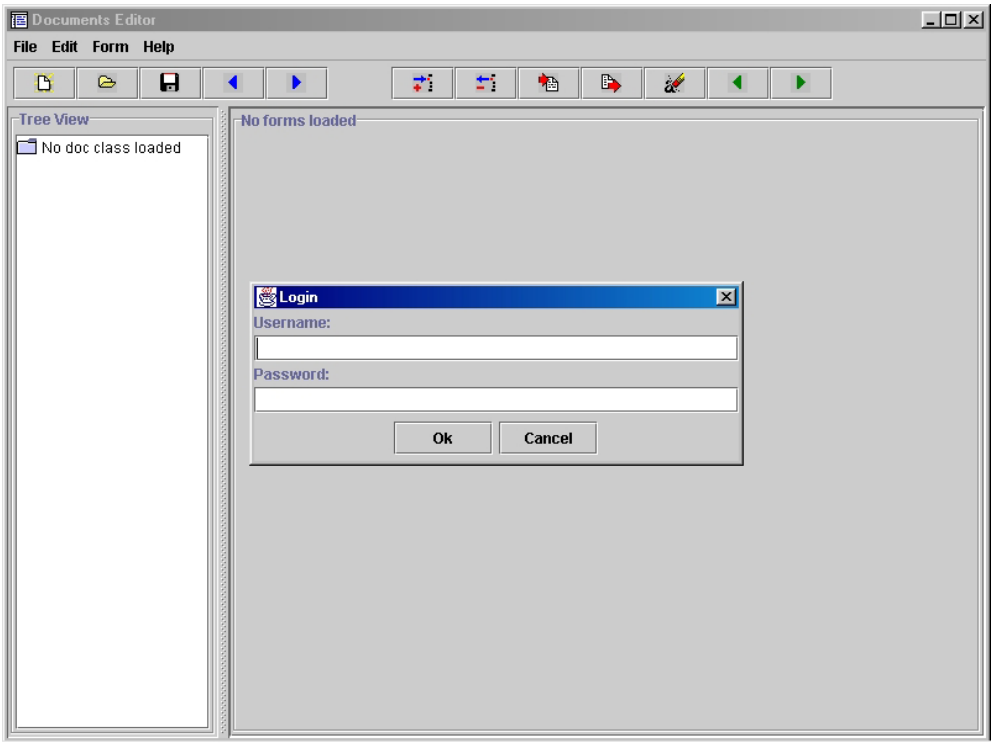


Figure 4.1b – DocsEditor Interface

The system administrator must register all its users. These users are arranged in groups and may configure the sharing permissions of their documents in the same bases of the permission system from UNIX operational system.

In its current version (2.05), the DocsEditor allows working with just one form and one document at a time. So, when a new document is loaded, any kind of previous document is discharged. If there's an active document when a new document is created, the editor allows saving the document.

The DocsEditor is a client-server application made up of two modules: the server module and the editor module. The server module is responsible for saving, restoring and controlling the properties of the edited documents by the instances of the editor module. The editor module allows the user to choose the kind of document he wants to work with, to edit a document and to send or restore documents from an active instance of the server module. Figure 4.2 shows the application's architecture.

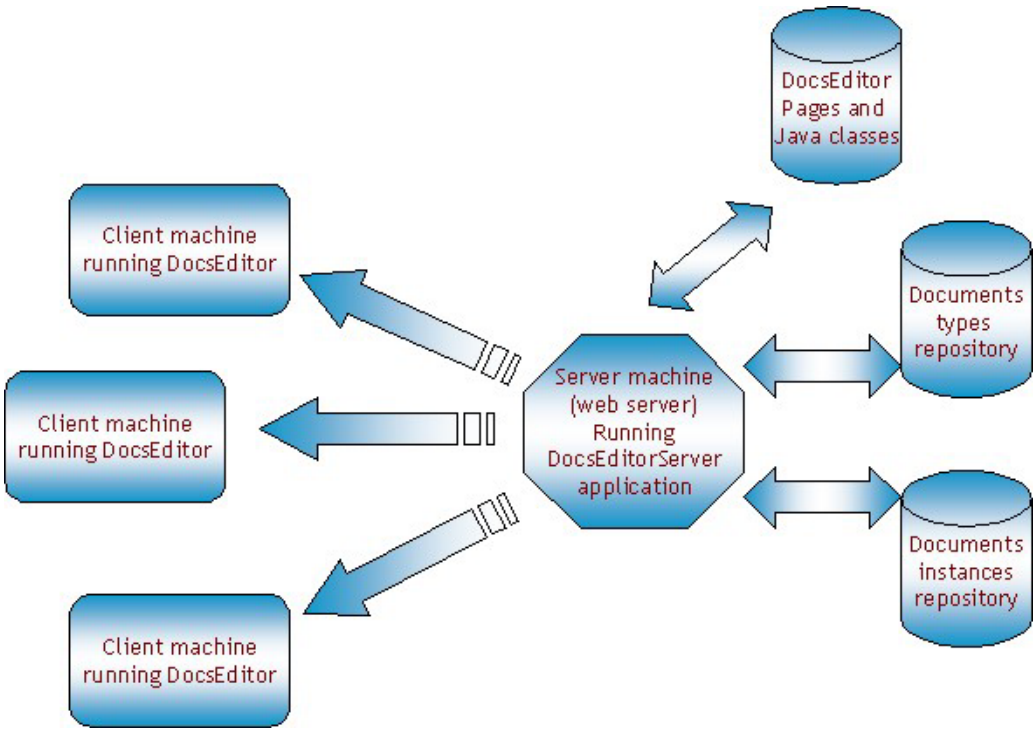


Figure 4.2 – Application Architecture

5. EXPERIMENT'S REPORT

The experiment made on this paper made use of the application discussed in the previous chapter, the DocsEditor, as a case study to a comparative analysis of the two formalisms discussed in chapter 2. The experience consisted of the creation of a usage model to the application using each one of the formalisms and, subsequently, of a comparison between these models, taking into account the constructing process and the information provided by each model.

The goal of this case study is evaluating the advantages and indications of each one of the formalisms in the construction of usage models to software statistical testing.

Once the target-application of the case study is chosen, we proceeded to its adjustment in order to provide information relating to its usage. We implemented an event log, which registered the initial state and the user's action. The event log is created locally, generating a log file to each user of the application.

Another relevant detail is the model delimitation. Usage models built in this paper don't take into consideration the items of the object tree, and consequently, the instanced forms. When building the models, we take into account the dialogue and selection screens and the state of the document (no document, titled document and untitled document) as states, as well as the application commands as transitions between these states.

In the next section, we describe the generated models, making use of the two different formalisms, describing their building process and its diagram.

5.1 Model 1 – Markov Chain

The structure of the model using the Markov Chain was extracted from the application description through a finite state machine, whose information was extracted from the system's specification. This state machine describes the system's evolution, from an initial state, the application loading, to a single final state, which is

its finalization. All the possibilities of user/application interactions are described, calculating all the possible paths of the system.

From this structure, and making use of the information contained in each user's event log, we proceeded to the verification of the transition rates of the Markov Chain, then constituted. Through traversing the chain, lead by the information of the event log, we counted the amount of times each path – transition between states – was followed.

Based on these values, and taking into account the amount of times the application was executed (28 executions) in the log sample, we established the transition rates, as number of transitions per execution. When the transitions had a null number of passages, they were given a low transition rate, indicating that they could occur, and taking into account the fact that the experiment used just a sample of the typical use of the software.

Figure 5.1a shows the usage model using the Markov Chain, showing distinct state classes. First, we can cite the states that reflect the current type of document, showing no document (No Doc state), a not yet titled document (Untitled Doc state) or even an already titled document (Titled Doc state, whose transitions appear expanded in figure 5.1b).

Besides this states class, the model still shows "dialogue" states. These states represent the message and/or selection window that are shown to the user during the application execution. Each dialogue (dialog 1, dialog 2 etc) represents a different kind of window, the letter at the end of the state name (a, b, c and so on) representing different situations in which this window is shown.

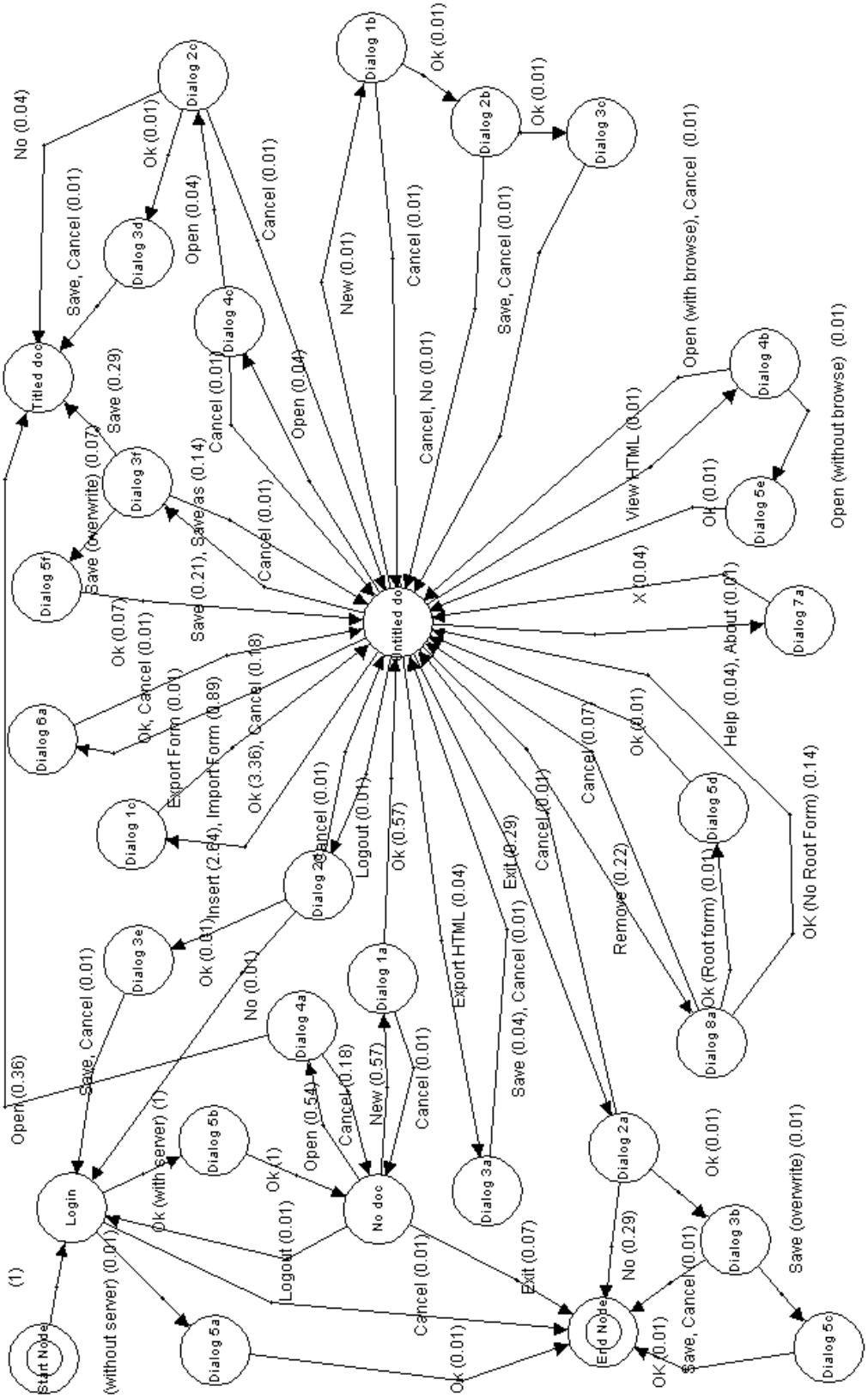


Figure 5.1a –DocsEditor Usage Model using Markov Chain

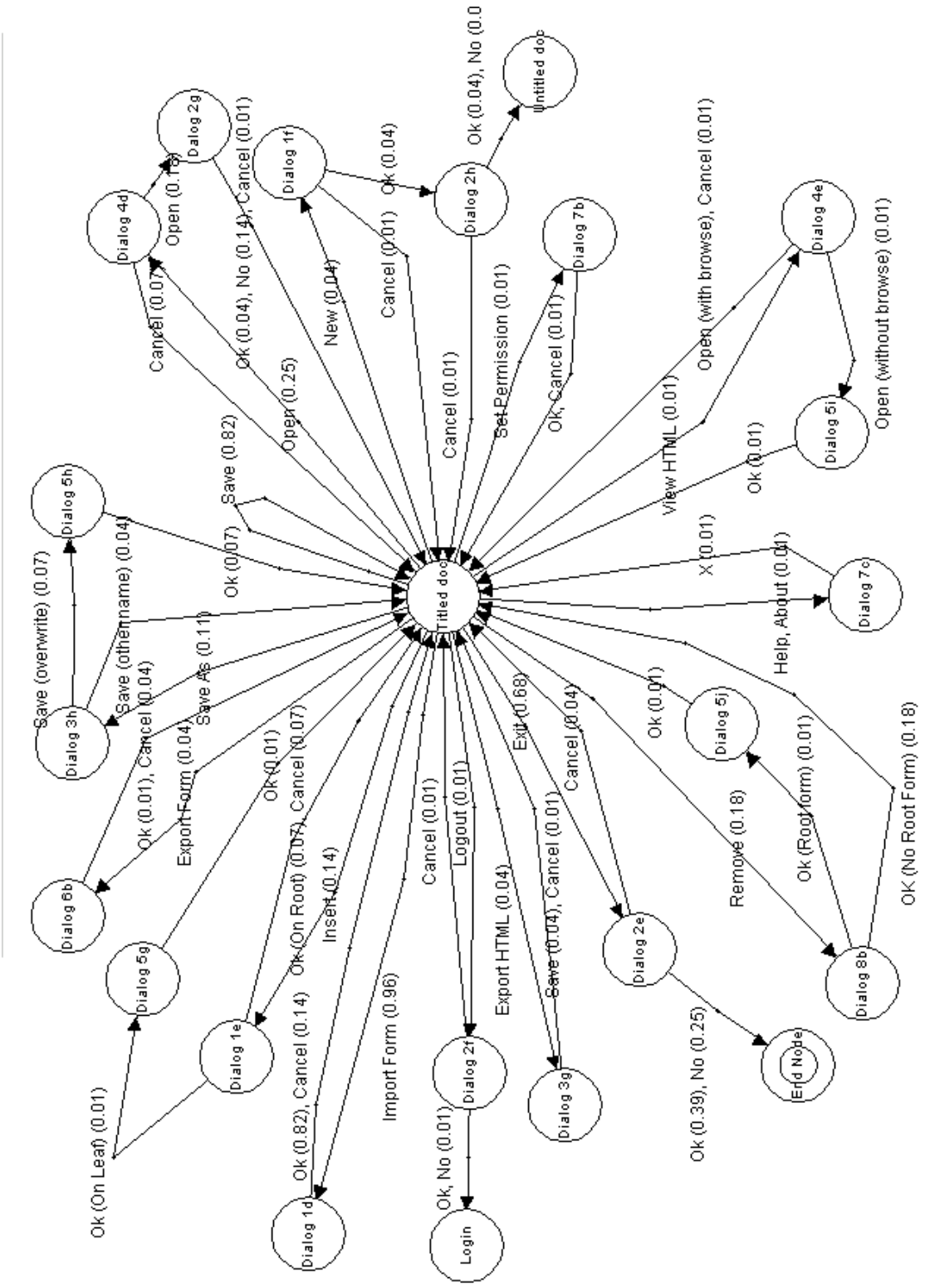


Figure 5.1b – Expansion of the distincted node

The initial and final states complete the model, and represent the load and finalization of the application, respectively, as well as the login state, which refers to the window in which the user enters his user code and password, so that he/she is allowed, or not, to use the tool. Some transitions take into consideration the type of node selected in the application's object tree (root node, leaf node or intermediate node), as well as the existence or not of an Internet browser or an active application server.

5.2 Model 2 – Stochastic Automata Network

The usage model built by using the Stochastic Automata Network had its conception associated with the observation of the state classes found in model 1. Therefore, we defined two SAN main automata, which are the "Type of Document" automaton (figure 5.2) and the "Dialogues" automaton (figure 5.3).

The first automaton's states represent the same types of documents of the previous model, besides having its arches representing the transitions between these states. The transitions in parentheses constitute synchronizing events, establishing this relation with the "Dialogues" automaton.

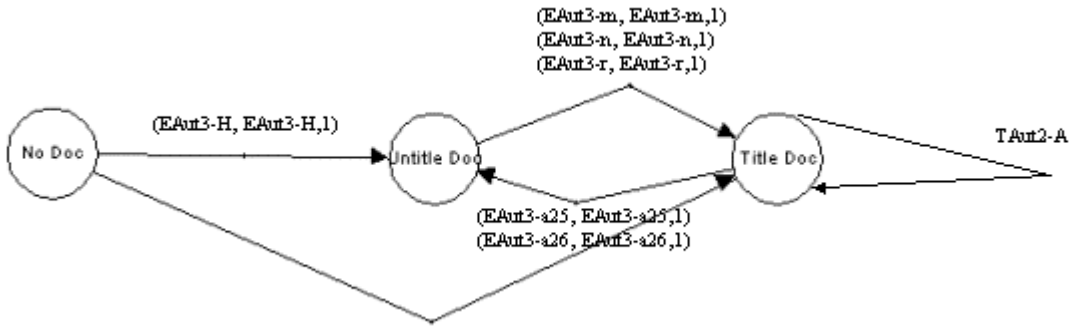
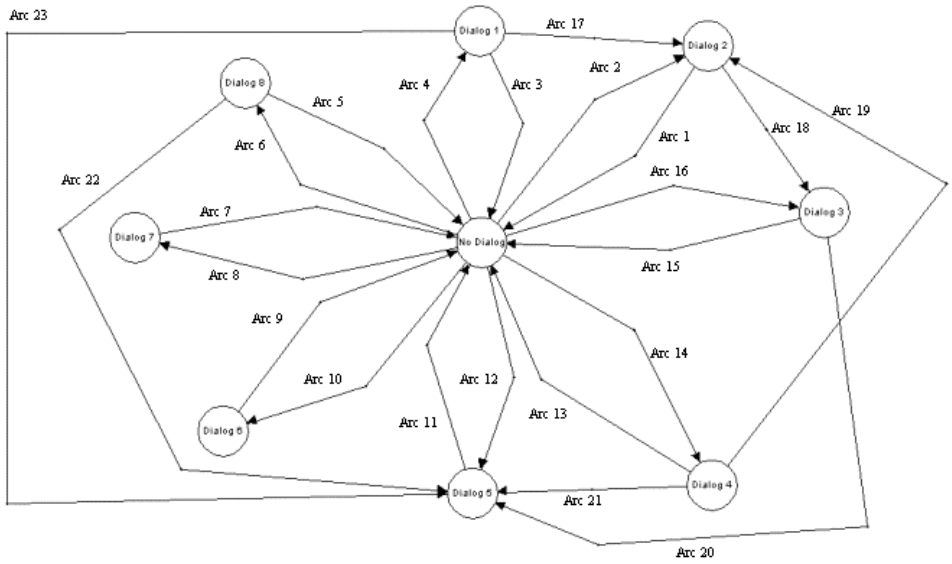


Figure 5.2 – "Document Type" automaton of the SAN

The second automaton's states represent the application's dialogue boxes, with the addition of a ninth state that means the non-exhibition of dialogues. The arches have different local transitions – beginning with a 'T' – and synchronizing events with the previous automaton. The identifiers of each arch's transition are grouped below the automaton's diagram, to allow a better visualization.



ARC 1	ARC 2	ARC 3	ARC 4	ARC 5	ARC 6	ARC 7	ARC 8	ARC 9	ARC 10
TAut3-M	TAut3-L	TAut3-G	TAut3-F	TAut3-O	TAut3-N	TAut3-U	TAut3-S	TAut3-v	TAut3-u
TAut3-e	TAut3-a1	(EAut3-H,1,1)	TAut3-b	TAut3-P	TAut3-a8	TAut3-a13	TAut3-T	TAut3-a39	TAut3-a38
TAut3-k	TAut3-a6	TAut3-c	TAut3-w	TAut3-a9		TAut3-a20	TAut3-a12	TAut3-a40	
(EAut3-m,1,1)	TAut3-a52	TAut3-y	TAut3-x				TAut3-a19		
TAut3-a2		TAut3-z	TAut3-a21						
TAut3-a4		TAut3-a22	TAut3-a46						
TAut3-a7		TAut3-a42							
TAut3-a24		TAut3-a43							
(EAut3-a25,1,1)		TAut3-a47							
(EAut3-a26,1,1)		TAut3-a48							
TAut3-a31									
TAut3-a32									
TAut3-a33									
TAut3-a53									
TAut3-a54									

ARC 11	ARC 12	ARC 13	ARC 14	ARC 15	ARC 16	ARC 17	ARC 18
TAut3-C	TAut3-A	TAut3-E	TAut3-D	TAut3-J	TAut3-I	TAut3-d	TAut3-f
TAut3-R	TAut3-B	TAut3-W	TAut3-V	TAut3-K	TAut3-o	TAut3-a23	TAut3-l
TAut3-Z		TAut3-X	TAut3-h	TAut3-g	TAut3-p		TAut3-a3
TAut3-t		TAut3-i	TAut3-a14	(EAut3-n,1,1)	TAut3-a34		
TAut3-a11		TAut3-a15	TAut3-a27	TAut3-q	TAut3-a49		
TAut3-a18		TAut3-a16		(EAut3-r,1,1)			
TAut3-a37		TAut3-a28		TAut3-a5			
TAut3-a45				TAut3-a34			
				TAut3-a35			
				TAut3-a50			
				TAut3-a51			

ARC 19	ARC 20	ARC 21	ARC 22	ARC 23
TAut3-j	TAut3-a	TAut3-Y	TAut3-Q	TAut3-a44
TAut3-a29	TAut3-s	TAut3-a17	TAut3-a10	
	TAut3-a36			

Figure 5.3 – “Dialogs” automaton of the SAN

The SAN is composed of three other automata, which represented transition conditions in the other model. The “Tree” automaton (figure 5.4a) represents the type of node selected in the application objects’ navigation tree, having the “Root” (root node), “Leaf” (leaf node) and Other (intermediate nodes) states. The “Server” automaton indicates the presence or absence of an active server model in the application, and the “Browser” automaton (figure 5.4c) indicates the presence or absence of an Internet browser.

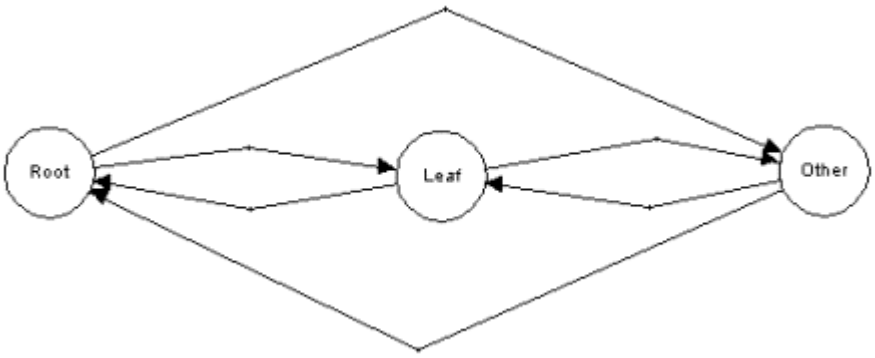


Figure 5.4a – “Tree” automaton of the SAN

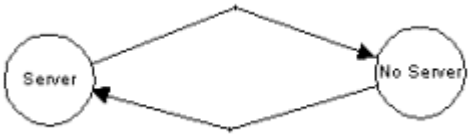


Figure 5.4b – “Server” automaton of the SAN

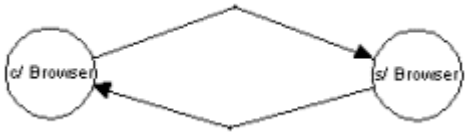


Figure 5.4c – “Browser” automaton of the SAN

So, after the definition of the Stochastic Automata Network constituent automata, we proceeded to the definition of local transitions and synchronizing events that would constitute the automata’s arches. Contrary to the process made in model 1, where the analysis of the specification and of an event log provided the transitions and

their rates, we used, in model 2, a kind of translation from model 1 to the structure of model 2.

So, conceiving each state of model 1 as an aggregate of the state of each one of the automata on model 2, we proceeded to the mapping of the transitions between the models. The transitions that caused a state alteration in just one of the automata constituted local transitions, while those that involved alterations in more than one automaton constituted synchronizing events.

The rates used in each transition are the same as in model 1, being expressed in the SAN's syntax and listed in the table below (figure 5.5). The transition rate in each arch of a SAN is resulted from the sum of all the local transitions contained in this arch, plus the synchronizing event rates. Each local transition rate, as seen below, has several conditions that, resulting in a true logical value, make it a not-null rate, effectuating the transition.

```
TAut3-A := (Aut2 == No Doc)&&(Aut4 == No Server)*0.01; (OK)
TAut3-B := (Aut2 == No Doc)&&(Aut4 == Server)*1.00; (OK)
TAut3-C := (Aut2 == No Doc)&&(Aut4 == Server)*1.00; (OK)
TAut3-D := (Aut2 == No Doc)&&(Aut4 == Server)*0.54; (Open)
TAut3-E := (Aut2 == No Doc)&&(Aut4 == Server)*0.18; (Cancel)
TAut3-F := (Aut2 == No Doc)&&(Aut4 == Server)*0.57; (New)
TAut3-G := (Aut2 == No Doc)&&(Aut4 == Server)*0.01; (Cancel)
EAut3-H := (Aut4 == Server)*0.57; (OK)
TAut3-I := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.04; (Export
html)
TAut3-J := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.04; (Save)
TAut3-K := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (Cancel)
TAut3-L := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.29; (Exit)
TAut3-M := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (Cancel)
TAut3-N := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.22; (Remove)
TAut3-O := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.07; (Cancel)
TAut3-P := (Aut2 == Untitled Doc)&&(Aut4 == Server)&&
           ((Aut1 == Leaf)||(Aut1 == Other))*0.14; (OK)
TAut3-Q := (Aut2 == Untitled Doc)&&(Aut4 == Server)&&
           (Aut1 == Root)*0.01; (OK)
TAut3-R := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (OK)
TAut3-S := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.04; (Help)
TAut3-T := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (About)
TAut3-U := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.04; (X)
TAut3-V := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (View html)
TAut3-W := (Aut2 == Untitled Doc)&&(Aut4 == Server)&&
           (Aut5 == Com Browser)*0.01; (Open c/ browser)
TAut3-X := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (Cancel)
TAut3-Y := (Aut2 == Untitled Doc)&&(Aut4 == Server)&&
           (Aut5 == Sem Browser)*0.01; (Open s/ browser)
TAut3-Z := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (OK)
TAut3-a := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (Save)
```

```
TAut3-b := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (New)
TAut3-c := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (Cancel)
TAut3-d := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (OK)
TAut3-e := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (Cancel,No)
TAut3-f := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (OK)
TAut3-g := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01;
(Save,Cancel)
TAut3-h := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.04; (Open)
TAut3-i := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (Cancel)
TAut3-j := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.04; (Open)
TAut3-k := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (Cancel)
TAut3-l := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (OK)
EAut3-m := (Aut4 == Server)*0.04; (No)
EAut3-n := (Aut4 == Server)*0.01; (Save,Cancel)
TAut3-o := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.21; (Save)
TAut3-p := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.14; (Save as)
TAut3-q := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (Cancel)
EAut3-r := (Aut4 == Server)*0.29; (Save)
TAut3-s := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.07; (Save)
TAut3-t := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.07; (OK)
TAut3-u := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (Export
form)
TAut3-v := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (Ok, Cancel)
TAut3-w := (Aut2 == Untitled Doc)&&(Aut4 == Server)*2.64; (Insert)
TAut3-x := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.89; (Import
form)
TAut3-y := (Aut2 == Untitled Doc)&&(Aut4 == Server)*3.36; (OK)
TAut3-z := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.18; (Cancel)
TAut3-a1 := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (Logout)
TAut3-a2 := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (Cancel)
TAut3-a3 := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (OK)
TAut3-a4 := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01; (No)
TAut3-a5 := (Aut2 == Untitled Doc)&&(Aut4 == Server)*0.01;
(Save,Cancel)
TAut3-a6 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.68; (Exit)
TAut3-a7 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.04; (Cancel)
TAut3-a8 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.18; (Remove)
TAut3-a9 := (Aut2 == Titled Doc)&&(Aut4 == Server)&&
            ((Aut1 == Leaf)||(Aut1 == Other))*0.18; (OK)
TAut3-a10 := (Aut2 == Titled Doc)&&(Aut4 == Server)&&
            (Aut1 == Root)*0.01; (OK)
TAut3-a11 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.01; (OK)
TAut3-a12 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.01; (Help,
About)
TAut3-a13 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.01; (X)
TAut3-a14 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.01; (View html)
TAut3-a15 := (Aut2 == Titled Doc)&&(Aut4 == Server)&&
             (Aut5 == c/ Browser)*0.01; (Open c/ browser)
TAut3-a16 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.01; (Cancel)
TAut3-a17 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.01; (Open s/
browser)
TAut3-a18 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.01; (OK)
TAut3-a19 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.01; (Set
Permission)
TAut3-a20 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.01; (OK, Cancel)
TAut3-a21 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.04; (New)
TAut3-a22 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.01; (Cancel)
```

```
TAut3-a23 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.04; (OK)
TAut3-a24 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.01; (Cancel)
EAut3-a25 := (Aut4 == Server)*0.04; (OK)
EAut3-a26 := (Aut4 == Server)*0.01; (No)
TAut3-a27 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.25; (Open)
TAut3-a28 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.07; (Cancel)
TAut3-a29 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.18; (Open)
TAut3-a30 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.18; (Open)
TAut3-a31 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.04; (OK)
TAut3-a32 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.14; (No)
TAut3-a33 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.01; (Cancel)
TAut3-a34 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.11; (Save as)
TAut3-a35 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.04; (Save)
TAut3-a36 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.07; (Save over)
TAut3-a37 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.07; (OK)
TAut3-a38 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.04; (Export
form)
TAut3-a39 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.01; (OK)
TAut3-a40 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.04; (Cancel)
TAut3-a41 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.14; (Insert)
TAut3-a42 := (Aut2 == Titled Doc)&&(Aut4 == Server)&&
             ((Aut1 == Root)||(Aut1 == Other)*0.07; (OK)
TAut3-a43 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.07; (Cancel)
TAut3-a44 := (Aut2 == Titled Doc)&&(Aut4 == Server)&&
             (Aut1 == Leaf)*0.01; (OK)
TAut3-a45 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.01; (OK)
TAut3-a46 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.96; (Import
form)
TAut3-a47 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.14; (Cancel)
TAut3-a48 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.82; (OK)
TAut3-a49 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.04; (Export
html)
TAut3-a50 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.04; (Save)
TAut3-a51 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.01; (Cancel)
TAut3-a52 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.01; (Logout)
TAut3-a53 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.01; (Cancel)
TAut3-a54 := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.01; (OK, No)
TAut2-A := (Aut2 == Titled Doc)&&(Aut4 == Server)*0.82; (Save)
```

Figure 5.5 – List of Local Transition Rates and Synchronizing Events of the SAN

Referencing figure 5.5, it's important to say that the Local Transition Rates begin with letter "T", followed by the correspondent automaton identifier – accordin to figure 5.6. The Sinchronizing Events begin with letter "E", using the sintax <event, rate, probability>. The events where the rate is unitary (equal to 1) show a slave relation in the transition related to the automaton that have the real rate, constituting the master.

Automaton	Identifier
“Tree” Automaton	Aut1
“Document Type” Automaton	Aut2
“Dialogs” Automaton	Aut3
“Server” Automaton	Aut4
“Browser” Automaton	Aut5

Figure 5.6 – Identification of SAN automata

6. CONCLUSIONS

Markov-based usage models have been widely applied to statistical testing, with important benefits, such as described in [Sayre 99b],[Tian 98],[Trammel 95],[Whittaker 94]. Stochastic automata networks add value to traditional Markov chains, both from modeling and computational viewpoints. In this paper, we described an investigation concerning the use of SAN to represent usage models for statistical testing, from the modeling perspective. At this point, some preliminary conclusions are available:

- Some features of the application that were implicit in the Markov chain model were easily made explicit in the SAn-based model - for example, the node type (root, leaf or other). In order to represent such features in the Markov chain, we would need to create many additional states, and the resulting model would be very complex;
- Some automata in the network correspond intuitively to environment requirements, which are very useful for test case generation - for example, the server state; the model captures information about the relative priorities (frequencies) of the different environments that should be tested;
- The conditions associated to the transitions, and their combinations, map directly to test case descriptions.

On the other hand, the high density of information contained in transitions makes model interpretation less intuitive. There are other extensions of traditional Markov-based models for statistical testing; for example, Tian and Lin [Tian 98] apply Unified Markov Models (UMM), which add hierarchy to Markov chains, in usage modeling tasks. SAN are not hierarchical, instead, they allow for many different types of relationships among system components, while keeping the description modular. From a computational point of view, SAN, also have higher scalability: for example, in the case of DocsEditor, we can perform model analysis considering many

servers and clients in the environment, with little impact in the computational effort to compute model properties. Such flexibility is useful for both statistical and load testing.

The experiment described here is a first case study: the target application is relatively simple, and we focused on the modeling process. The next (already started) steps include: analyzing the computational properties of the model, developing a tool for random test case generation from SAN-based usage models, executing the tests, and evaluating the results. Also, we will develop experiments with more complex target applications.

REFERENCES

[Fernandes 98] FERNANDES, P.; PLATEAU, B. & STEWART, W.J. **Efficient Descriptor-Vector Multiplication in Stochastic Automata Networks.** In: *Journal of the ACM – volume 45 – nb. 3*, May 1998.

[Fosse 95] THEVENOD-FOSSE, P.; WAESELYNCK, H. & CROUZET, Y. **Software Statistical Testing.** In: *Predictably dependable computing systems,* Esprit basic research series, Springer-Verlag, 1995.

[Marre 95] MARRE, B.; THEVENOD-FOSSE, P.; WAESELYNCK, H.; LE GALL, P. & CROUZET, Y. **An Experimental Evaluation of Formal Testing and Statistical testing.** In: *Predictably dependable computing systems,* Esprit basic research series, Springer-Verlag, 1995.

[Myers 79] MYERS, G. J. **The art of software testing.** John Wiley & Sons, 1979.

[Plateau 96] PLATEAU, B. & STEWART, W.J. **Stochastic Automata Networks: Product Forms and Iterative Solutions**. Technical Report INRIA nb. 2939. July 1996.

[Sayre 99a] SAYRE, K. & POORE, J.H. **Partition Testing with Usage Models.** In: *Proceedings of the Harlan Mills Colloquium, IEEE,* May 1999.

[Sayre 99b] SAYRE, K. **Improved Techniques for Software Testing Based on Markov Chain Usage Models.** *PhD Thesis.* University of Tenessee, Knoxville, December 1999.

[Stewart 94] STEWART, W.J. **Introduction to the Numerical Solution of Markov Chains.** Princeton University Press, 1994.

[Stewart 95] STEWART, W.J. **Computations with Markov Chains.** Kluwer Academic Publishers, 1995.

[Tian 98] TIAN, J. & LIN, E. **Unified Markov Models for Software Testing, Performance, Evaluation, and Reliability Analysis.** In: *Proceedings of the 4th ISSAT International Conference on Reliability and Quality in Design,* 1998.

[Trammell 95] TRAMMELL, C. **Quantifying the Reliability of Software: Statistical Testing Based on a Usage Model.** In: *Proceedings of the Second IEEE International Symposium on Software Engineering Standards,* Canada, August 1995.

[Whittaker 94] WHITTAKER, J. **A Markov Chain Model for Statistical Software Testing.** In: *IEEE Transactions on Software Engineering, volume 20, nb. 10*, October 1994.