RELIABILITY ASSESSMENT OF E-COMMERCE APPLICATIONS

V. S. Alagar
Concordia University
Montreal, Quebec, Canada

O. Ormandjieva
Concordia University
Montreal, Quebec, Canada

Keywords: E-Commerce, reliability prediction, software measurement, Markov model.

Abstract: The paper discusses a formal approach for specifying time-dependent E-Commerce applications and proposes a Markov model for reliability prediction. Measures for predicting reliability are calculated from the formal architectural specification and system configuration descriptions. Our methods have been implemented and a set of sample results obtained from it for a simple system is given.

1 INTRODUCTION

The *reliability* of a software system is defined in (IEE,) as the ability to perform the required functionality under stated conditions for specified period of time. In this paper the software system under discussion is an E-Commerce system. An E-Commerce can be viewed as a large and complex distributed system whose heterogeneous components interact in various ways to achieve the result of an application. Often, the performance of an application initiated at a site is rated as good if the server at that site is robust and maintains its links to other web objects throughout a transaction session. Such a rating does give a subjective qualitative assessment, but does not provide a scientific quantitative measurement of the reliability of the site or the system itself. This paper proposes a rigorous methodology for predicting the reliability of E-Commerce applications using Markov models constructed from a formal model of the E-Commerce application.

Many techniques exist to test and statistically analyze traditional software. However, these methods can not be readily applied to E-Commerce systems. In a recent paper Kallepalli and Tian (Kallepalli and Tian, 2001) have surveyed the characteristics of Web applications and usage and proposed a statistical testing method for Web applications. Their approach relies on usage and failure information collected in the log files. They define *Web failure* as the inability to correctly deliver information or documents required by the users. Based on this definition of failure, they classify types of failures and provide a method for testing *source or content* failures. We complement and enrich this work by offering a formal time-constrained model of a simple E-Commerce model.

We propose an early reliability prediction based on the analysis of the formal architecture model of the E-Commerce applications using Markov models. A Markov model dynamically adapts to time-dependent system configurations that satisfy the architectural design (Ormandjieva, 2002). We view the E-Commerce applications as large Markov systems, in which state changes within each object of the system occur with certain probabilities.

Following the work (Papazoglou, 2000), we use agent paradigm to formally model E-Commerce systems. Agents are usually qualified by their *goals*, *knowledge*, *beliefs*, and *intentions*. We follow the property-oriented definition and classification from (Alagar et al., 2002), and introduce primitive agent types in modeling an E-Commerce system.

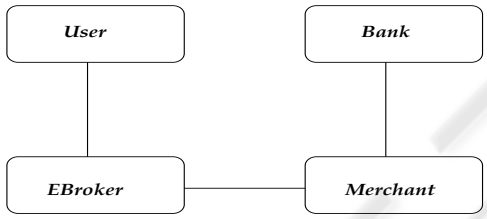
The organization of the rest of the paper is as follows. A formal model of the E-Commerce is given in Section 2. Section 3 formally describes the method of modeling the E-Commerce application as a Markov system. Section 4 presents the reliability prediction measures obtained from our implementation for a simple system. Section 5 concludes the paper with a discussion on our ongoing research directions.

S. Alagar V. and Ormandjieva O. (2004).
RELIABILITY ASSESSMENT OF E-COMMERCE APPLICATIONS.
In *Proceedings of the First International Conference on E-Business and Telecommunication Networks*, pages 30-37
DOI: 10.5220/0001399900300037

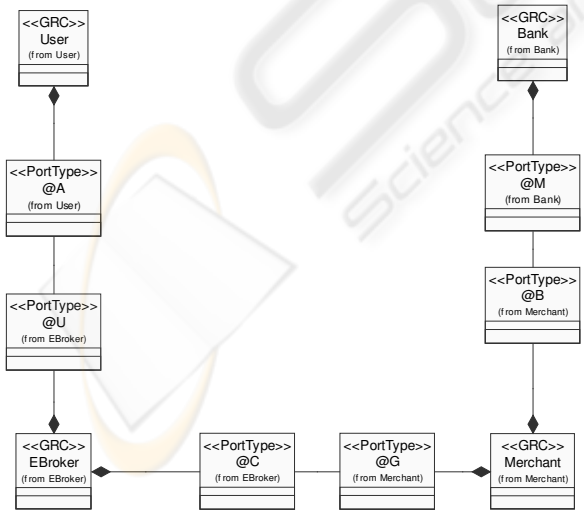
2 E-COMMERCE MODEL

In this paper, an *agent* is a computational entity handling sequences of messages (events) to implement the expected behavior of an agent as understood in agent technology (FIP, ; Papazoglou, 2000). Agents can receive and emit events, generate events internally, and react to events received from its environment. Such handling is fully determined by the implementation of the event. Peer-to-peer communication between agents is enabled by bidirectional connections transmitting events according to a specified protocol. We refer to (Franklin and Graesser, 1996; Alagar et al., 2002) for agent classification, agent types, agent behaviors, and their conformance to FIPA agents. We use the notation from (Alagar et al., 2002) for describing an E-Commerce system.

For illustration, we develop the formal model of an E-Commerce system consisting of four agent types User, EBroker, Merchant, and Bank. An instance of an agent type is an agent. The high-level architecture shown in Fig. 1(a) captures the allowed connections between agents in an application.



(a) Generic E-Commerce Architecture



(b) Agent Types and Port Types

Figure 1: High-Level Architecture Diagrams

2.1 Formal Models

From the architecture shown in Fig. 1(a) we determine the agent types and *port types* for each agent type. A port-type of an agent type [] is an abstraction of connection point where events are emitted and received from another agent type. The symbol @ is used to introduce port types. An agent of the agent type $A[L]$, where L is the list of port types, is created by instantiating each port type in L by a finite number of ports and assigning the ports to the agent. Any message defined for a port type can be received or sent through any port of that type. For instance, $A_1[p_1, p_2 : @P; q_1, q_2, q_3 : @Q]$, and $A_2[r_1 : @P; s_1, s_2 : @Q]$ are two agents of the agent type $A[@P, @Q]$. The agent A_1 has two ports p_1 and p_2 of type $@P$, and three ports q_1, q_2, q_3 of type $@Q$. Both p_1 and p_2 can receive or send messages of type $@P$; the ports q_1, q_2, and q_3 can receive and send messages of type $@Q$. Sometimes the port parameters of agents are omitted in our discussion below. Messages are allowed to have parameters.

An E-Commerce system, consistent with the architecture in Figure 1(a), consists of a finite set A of agents $A_1, \ldots A_n$, where each A_i is an instance of one of the agent types shown. From the description of the problem we extract the messages (events) for communication and partition the set of the messages into cohesive subsets, creating port-types. For instance, the EBroker agent type requires two port types: one for communication with the User agent and the other for communication with the Merchant agent. Figure 1(b) shows a refinement of Figure 1(a) obtained by adding the port type associations to agent types. The model is further enriched by including abstract data types to express abstract computations done by agents.

2.2 Behavior of the System

An agent can handle one external event at a time. We assume that there is no connection delay, and an agent emits an event only if the receiver of the event is prepared to accept it. That is, emitting and absorbing an event is considered as one atomic action. Thus, in a specific transaction in the system, the activity of each agent A on a set of connections C is observed as the finite sequence of events which A handles on C in that session. Notice that the connections C are at the ports of A and the ports of those agents interacting with it in that session. The *trace* of A on C is the set of all computation paths in the state machine description of A. The set of all traces of A is referred to as the *behavior* of A. Fig. 2 and Fig. 3 show the state machines for the agents in our E-Commerce system. To illustrate the usefulness of time constraints we have specified time constraints for EBroker actions. In general

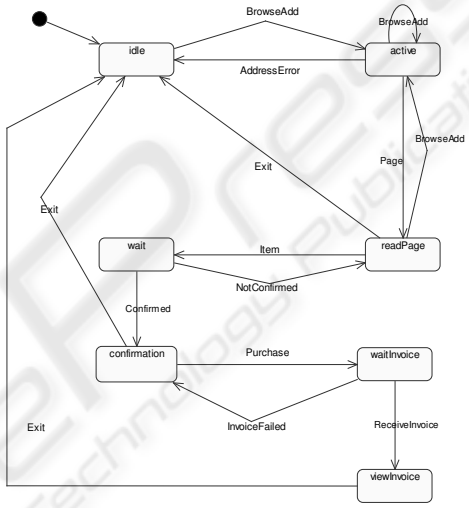
Merchant and Bank agents should also respect timeliness, and appropriate time constraints can be introduced in their specifications.

A successful transaction is conducted by the agents in the following manner. The E-Commerce system is initiated with the message *BrowseAdd* from a user to the broker. Both User and EBroker agents synchronize on this event: the agent User goes to *active* state, the agent EBroker goes to the state *getProduct*. The other agents do not change their states. The agent EBroker sends the message *GetProductInfo* to the Merchant agent within 2 units of time from the instant of receiving the message *BrowseAdd*, and they both simultaneously change their states to *waitProduct*, and *product* respectively. The Merchant agent responds to the agent EBroker with the message *ProductInfo* which cause them to simultaneously change their states to *idle*, and *browseSucceed*. Within 2 units of time of receiving the webpage from the merchant the EBroker communicates to the User agent with the message *page*, causing them to simulataneously change their states to *idle* and *readPage*. This completes the first phase of a successful user interaction, where the user has received a web page for browsing.

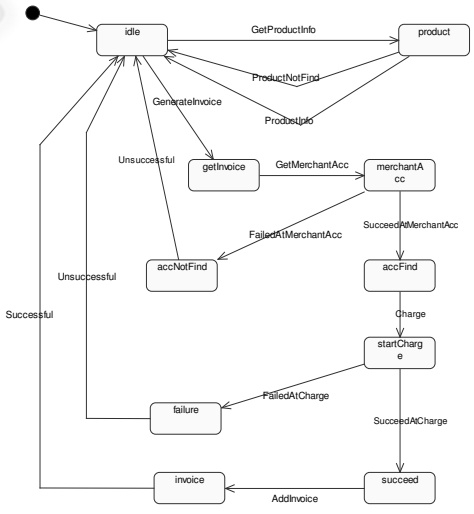
The user may decide to exit or may enter the next phase of transaction by initiating the message *Item* to the agent EBroker. In the later case, they synchronize and change their states to *wait* and *startNegotiation*. The user continues to wait until the EBroker agent completes a sequence of internal computations triggered by the internal events $\langle$*AddStatistic, TotalQuantity, GetMinPrice, GetRiskBalance, CalculateAcceptablePrice* $\rangle$. All brokers share a table of information on user requests. Each entry in the table is a tuple containing *userid*, *merchantid*, *productid*, *quantity_requested* and *priceofferred*. Based on this table of information and a formula for risk-profit analysis, the broker determines a price of a product. We have used abstract data types to specify tables and databases within agent types.

After completing the risk-profit analysis, the offer made by the user is either accepted or rejected by the agent EBroker. The decision to accept or reject the price offer is communicated to the user within the time interval [t+5, t+8], where t is the time at which the message *item* was received by the broker. For a successful transaction, the message *Confirmed* is exchanged between the User and EBroker agents, causing their simultaneous transitions to the states confirmation and *idle*. This completes the second phase of successful transaction. At this instance, the User agent is in state *confirmation* and the other agents are in idle states.

The user can exit from the system without making a purchase at this stage. The product is purchased by the user and the invoice is received in the next and final phase of the transaction. The message *Purchase* is sent by the agent User to the agent EBroker, and the User agent goes to *waitinvoice* state where it waits until receiving the message *ReceiveInvoice* from the agent EBroker. The EBroker agent communicates with Merchant agent through the message *GenerateInvoice* and they simulataneously change their states to *askMerchant* and *getInvoice*. At this instant, the User is in state *waitinvoice*, the agent EBroker is in state *askMerchant*, the agent Merchant is in state *getinvoice*, and the agent Bank is in state *idle*. The states of User and EBroker do not change until the agents Merchant and Bank collaborate to produce the invoice.



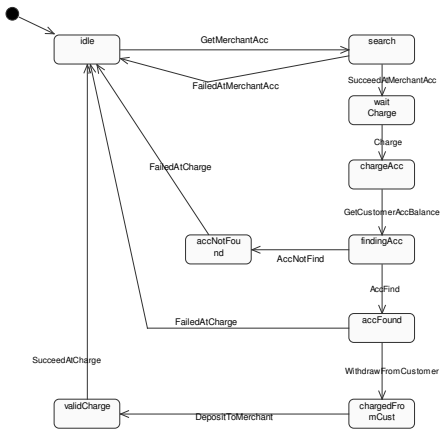
(a) Statechart Diagram for User



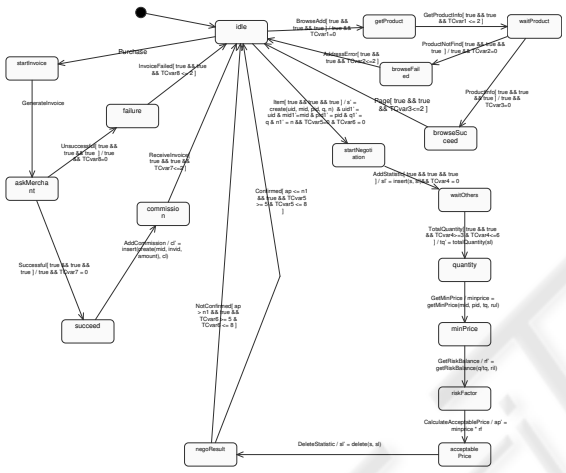
(b) Statechart Diagram for Merchant

Figure 2: Behavior Descriptions - 1

To produce the invoice, the agent Merchant sends the message *GetMerchantAcc* to the agent Bank, re-



(a) Statechart Diagram for Bank



(b) Statechart Diagram for EBroker

Figure 3: Behavior Descriptions - 2

ceives back the reply *SucceedMerchantAcc*, and then responds through the message *Charge*. At the end of this sequence of message exchanges, they reach the states *startCharge* and *chargeAcc*. The Merchant agent waits in state *startCharge* until the Bank agent completes a sequence of internal computations and sends the message *SucceedAtCharge* to it. Upon receiving this message, the Bank agent goes to *idle* state, the Merchant agent goes to *succeed* state. After completing the internal computation to record the invoice, the *Merchant* agent communicates to EBroker through the message *Successful* and their states to *idle*, and *succeed*. At this instance, the agents Bank and Merchant are in their *idle* states, the agent EBroker is in state *succeed*, and the agent User is in state *waitinvoice*. The agent EBroker records the commission earned in this transaction within 2 time units of receiving the message *ReceiveInvoice* and sends the invoice to the User. Having sent the invoice, the agent EBroker goes to its *idle* state. The agent User executes the internal message *Exit* and goes into its *idle* state.

Figure 4 shows an E-commerce system configured with three users, one E-broker, two merchants and three banks. Each instance of User type models a user with one port of type @A to communicate with an agent of EBroker type. The E-broker agent in the subsystem is an instance of EBroker agent type having three ports of type @U, one port for each user; the two ports of type @M are for communication with the merchants in the system. Each merchant agent in the system is an instance of Merchant agent type with a port of type @G for communicating with the broker agent, and three ports of type @B to communicate with the banks in the system. The agents are linked through connectors at their respective compatible ports for communication. For instance, the port $@A1$ of user $U1$ is linked to the port $@U1$ of the E-broker $E1$. That link is not shared by any other agent. Consequently, the architecture specification ensures trust in communications.

The following section introduces a rigorous methodology for predicting the reliability of E-Commerce applications using Markov models constructed from a formal model of the E-Commerce application.

3 MARKOV MODELS

Markov models are one of the most powerful tools available to engineers and scientists for analyzing complex systems. The basic concepts are explained below.

3.1 Markov Models: Basic Concepts

Analysis of Markov models yield results for both the time-dependent evolution of the system and the steady state properties of the system. The Markov property states that given the current state of the system, the future evolution of the system is independent of its history.

The Markov model of an E-Commerce component may be represented by a state diagram. The states represent the stages in the E-Commerce component that are observable to the users, and the transitions between states have assigned probabilities. An algebraic representation of a Markov model is a matrix, called *transition matrix*, in which the rows and columns correspond to the states, and the entry p_{ij} in the i-th row, j-th column is the transition probability for being in state j at the stage following state i. We use transition matrix representation in reliability calculation algorithms.

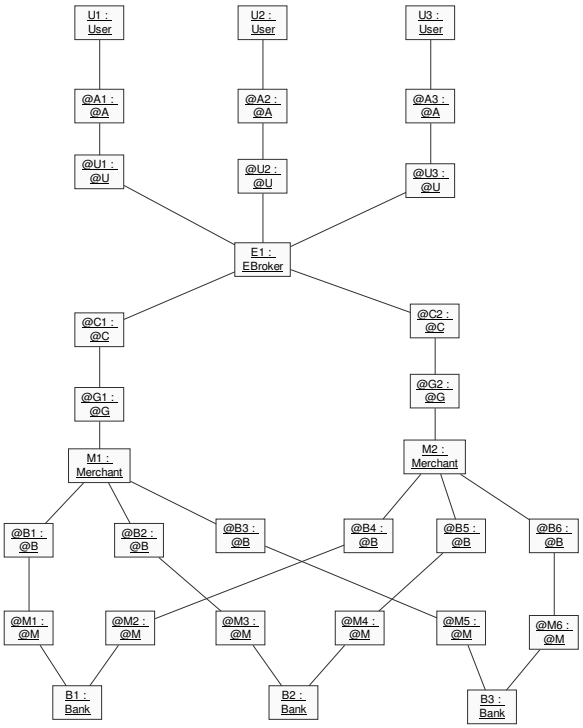


Figure 4: Subsystem Architecture

3.2 Discussion

Initial transition probabilities, obtained from various sources including log files and other subjective opinions of experts can not be used for predicting the reliability of the system. We contend that the reliability should be calculated from the *steady state* of the Markov system. A steady state or equilibrium state is one in which the probability of being in a state before and after transitions is the same as time progresses. Computing the steady state vector for the transition matrix of a large system is hard. However, as in our approach, when the system is modularly constructed it seems possible to partition the system into smaller components, which might reduce the complexity of computing steady state vectors.

3.3 Algorithm

We construct the Markov model of a E-Commerce system in three steps. In the first step we construct the Markov models for E-Commerce objects. In the second step we construct the Markov models for every pair of interacting objects in the system configuration specification. Finally in the third step we construct the Markov model for the fully configured system.

3.3.1 Step 1: Markov Models for Objects

We associate with each E-Commerce object in the architecture another finite state machine, called its *Markov* model. The states in the Markov model of an object are the states of the object in its state machine description. A transition between two states in the Markov model is defined only if there exists at least one transition between those states in its state machine. In the absence of statistical information gathered by experts on the usage and failure, we will assume that all the external events have equal probability in each state. For the transition from state i to state j in the Markov model, a fixed probability p_{ij} of it going into state j at the next time step is calculated as follows:

1. The initial probabilities for all the transitions in the state machine of the reactive object are calculated. The algorithm for calculating such probabilities for a state is based on the following assumptions: 1) all external events that can happen at the state have the same probability; 2) all internal events that can happen at the state have the same probability, and (3) these are in general different.
2. In case there is more than one transition $\{l_1, \ldots, l_n\}$ of the same type (shared/internal) from state$_i$ to state$_j$, then the above mentioned transitions are substituted by one whose probability is $P = 1 - (1 - P\{l_1\}) \times \ldots (1 - P\{l_n\})$
3. The probabilities of all the transitions for a state have to sum to 1.

3.3.2 Step 2: Markov Model for Object Pairs

The interaction between two objects is due to shared events. We compute the state machine for an interacting pair of objects and compute the Markov model with transition probabilities from the transitions at each state of the product machine.

Algorithm for Transition Matrix for the Synchronous Product Machine

Let E_1 and E_2 be the sets of internal events in the statecharts P and Q of interacting objects, and F denote the set of shared events. Let M_1 and M_2 be the transition matrices for P and Q. Let R be the synchronized product machine of P and Q. Algorithm SPM computes the transition matrix M of R by first computing the synchronous product machine R, and next determining the transition probabilities for transitions in each state of R. If all the transitions in a state are labeled by internal events or if all of them are labeled by shared events the probabilities are obtained by normalizing the probabilities in their respective machines. However, if both internal events and

shared events occur at the state, the probabilities for the shared events are calculated first, and the remaining measure is distributed to transitions labeled by internal events.

Algorithm SPM

Step 1. $p = 1$; // row sum

Step 2. $x_1 = \{e|\ e$ is a shared event occurring at state i (P) and at state j $(Q)\ \}$
$x_2 = \{e|\ e$ is an internal event occurring at state i $(P)\} \cup \{e|\ e$ is an internal event occurring at state j $(Q)\ \}$

Step 3. If $x_1 \neq \emptyset$ // calculate probabilities for transitions due to shared events
then $NF = 0$ (Normalization Factor); $set_1 = \emptyset$;

Step 3.1 For each event $e \in x_1$ find the (set of) states $i^{'}$ (P) and $j^{'}$ (Q) such that $i \xrightarrow{e} i^{'}$, $j \xrightarrow{e} j^{'}$

Step 3.2 $y = y \cup \{i^{'}, j^{'}\}$, if $\{i^{'}, j^{'}\} \notin y$

Step 3.3 $NF = NF + M_1[i, i^{'}] \times M_2[j, j^{'}]$

Step 3.4 $M[(i, i^{'}), (j, j^{'})] = M_1[i, i^{'}] \times M_2[j, j^{'}]$

Step 3.5 $set_1 = set_1 \cup (j, j^{'})$

Step 4. If $x_2 \neq \emptyset$ // calculate probabilities for transitions due to internal events
then $NF^{'} = 0$ (Normalization Factor); $set_2 = \emptyset$;

Step 4.1 For each event $e \in x_2$, if $e \in M_1$ then

find the state $i^{'}$ (M_1) such that $i \xrightarrow{e} i^{'}$;
$y = y \cup \{i^{'}, j\}$ if $\{i^{'}, j\} \notin y$;
$M[(i,j)(i^{'}, j)] = M_1[i, i^{'}]$; $NF^{'} = NF^{'} + M[(i,j)(i^{'}, j)]$;
$set_2 = set_2 \cup (i, i^{'})$

else

find the state $j^{'}$ (M_2) such that $j \xrightarrow{e} j^{'}$;
$y = y \cup \{i, j^{'}\}$ if $\{i, j^{'}\} \notin y$;
$M[(i,j)(i, j^{'})] = M_2[j, j^{'}]$; $NF^{'} = NF^{'} + M[(i,j)(i, j^{'})]$;
$set_2 = set_2 \cup (j, j^{'})$

Step 5. If $x_1 = \emptyset \wedge x_2 = \emptyset$, the (i, j) row is deleted from M

Step 6. If $x_1 = \emptyset \wedge x_2 \neq \emptyset$
For each $(i^{'}, j^{'}) \in set_2$ do

$$M[(i,j),(i^{'},j^{'})] = \frac{M[(i,j),(i^{'},j^{'})]}{NF^{'}}$$

Step 7. If $x_1 \neq \emptyset \wedge x_2 = \emptyset$
For each $(i^{'}, j^{'}) \in set_1$ do

$$M[(i,j),(i^{'},j^{'})] = \frac{M[(i,j),(i^{'},j^{'})]}{NF}$$

Step 8. If $x_1 \neq \emptyset \wedge x_2 \neq \emptyset$ do
For each $(i^{'}, j^{'}) \in set_2$ do

$$M[(i,j),(i^{'},j^{'})] = \frac{(1 - NF) \times M[(i,j),(i^{'},j^{'})]}{NF^{'}}$$

Step 9. Fill in the matrix M with 0 where there are no entries.

3.3.3 Step 3: Markov Model for a System

A system configuration, when partitioned into slices based on synchronization criteria, produces two types of subsystem components: (1) linear configuration, and (2) non-linear configuration. As an illustration, a simple system in which one user is interacting with a browser to access information, as shown in Figure 5(a) is linear, whereas the system in which users join the system at different times, as shown in Figure 5(b) is non-linear.

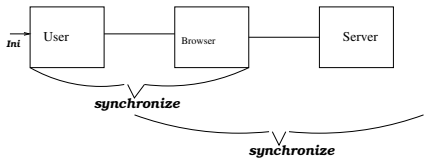
Case 1: Linear System

In a linear system, objects synchronize in the past. If $o_1, \ldots, o_n$ are objects in the linear system and $M_1, \ldots, M_n$ are respectively their transition matrices, then the transition matrix M of the linear system is computed as follows:

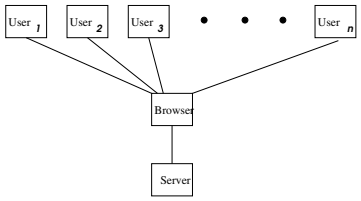
1. Compute $M = M_1 \otimes M_2$ (Apply Algorithm SPM)
2. for $j = 3$ to n compute $M = M \otimes M_j$ (Apply Algorithm)

Case 2: Non-linear System

In a non-linear system, as in Figure 5(b), several objects interact with an object. These interactions may be initiated at different times. The synchronous product machine dynamically changes as and when objects join or leave the system, and hence the transition probability matrices also change, and should be recomputed. We explain the computation procedure for Figure 5(b). For simplicity of discussion we assume that users join the system one at a time, but don't leave the system. Let $0, k_1, k_2, \ldots, k_{n-1}$ be the intervals of successive arrivals of users. That is, for $j > 1$ the j-th user joins the system $k_{j-1} (> 0)$ time units after $j - 1$-st user joined the system. Let M_1 be the transition matrix of the Markov model for the linear system computed at time step k. The transition matrix for $M_{(n)}$, $n \geq 2$, users interacting with one browser and one server object is calculated as follows:



(a) Linear Architecture



(b) Non-Linear Architecture

Figure 5: Linear and Non-Linear Models

$$M_{(2)} = M_1^{k_1} \bigoplus M_1$$
$$M_{(3)} = [M_{(2)}]^{k_2} \bigoplus M_1$$
$$\vdots$$
$$M_{(j)} = [M_{(j-1)}]^{k_{j-1}} \bigoplus M_1, 2 \leq j \leq n$$
$$\vdots$$

In the above calculation, the symbol $\bigoplus$ denotes the direct product operator for matrices. The justification for direct product computation is based on the observations:

- the event *received* by the system from a new *User* object can come only when the system is in any one of its states in which it can be accepted,
- the incoming event is not time-constrained with respect to the Markov model of the system. that is M_1 for the new interaction is independent of $[M_{(j-1)}]^{k_{j-1}}$, and
- when a new object joins the system the size of the new Markov matrix increases m-fold, where m is the size of the Markov matrix of the new object.

For the non-linear system in Figure 5(b), assume that four users join the system,one at a time, at times 0, 2, 4, 5. So, $k_1 = 2; k_2 = 2; k_3 = 1$. The transition matrix of the system at different time points are shown below:

At time 0 or 1: (1 user): M_1
At time 2: (2 users): $M_{(2)} = M_1^2 \bigoplus M_1$
At time 4: (3 users): $M_{(3)} = [M_{(2)}]^2 \bigoplus M_1$
At time 5: (4 users): $M_{(4)} = [M_{(3)}] \bigoplus M_1$

Let us consider the general case when $r > 1$ users simultaneously join the system, say when there are $j-1$ users in the system. It is easy to see that the transition matrix for the new configuration with $r + j - 1$ users is $M_{(r+j-1)} = [M_{(j-1)}]^{k_{j-1}} \bigoplus M_1^{(r)}$, where $M_1^{(r)}$ is the direct product $M_1 \bigoplus M_1 \bigoplus \ldots M_1$, taken r times. When r users leave the system, the transition matrix is computed as follows: Let there be $j(\geq 2)$ users in the system when $r(1 < r \leq j)$ users leave. If $r = j$, then the transition matrix is not defined. If $r < j$, there are $j - r$ users left in the system. If d is the interval of time that elapsed between the latest time when there were $j - r$ users in the system and the current instant, then the new transition probability matrix is $[M_{(j-r)}]^d$. The rationale is that the transition probability matrix $M_{(j-r)}$ for $j - r$ users have evolved over d time steps.

For the E-Commerce system shown in Figure 4 the Markov matrix can be computed using the above method. The details are left to the reader.

4 RELIABILITY MEASURES

We define the reliability prediction for a system configuration composed from k objects as the level of certainty quantified by the source $excess - entropy$:

$$Reliability(Subsystem) = \sum_{i=1}^{k} H_i - H$$

where $H = -\sum_i v_i \sum_j p_{ij} \; log \; p_{ij}$ is a level of uncertainty of the Markov system corresponding to a subsystem; v_i is a *steady state distribution vector* for the corresponding Markov system and the p_{ij} values are the transition probabilities. H_i is a level of uncertainty in a Markov system corresponding to a reactive object. For a transition matrix P the steady state distribution vector v satisfies the property $vP = v$. The level of uncertainty H is related exponentially to the number of paths that are "statistically typical" of the Markov system. Thus, higher entropy value implies that more sequences must be generated in order to accurately describe the asymptotic behavior of the Markov system.

The reliability prediction for a system is defined as the least reliability measure value among its m subsystems:

$Reliability(System) = min\{Reliability(Subsystem_i)\}_i^m$

We chose the minimum value due to the safety-critical character of the real-time reactive systems. Higher value of reliability measure implies less uncertainty present in the model, and thus higher level of software reliability.

The Markov model of a configured system changes when the system undergoes change. The calculation of the Markov matrix for the reconfigured system would allow to compare the systems based on reliability prediction. If the system configuration C_{j-1}

changes to the configuration C_j, we need to calculate the reliability of the configuration C_j and compare it with the reliability of the configuration C_{j-1}:

$$Reliability(C_{j-1}) = min\{Reliability(S_i)\}_i^m,$$

where S_i is a subsystem of C_{j-1}, and

$$Reliability(C_j) = min\{Reliability(S_i')\}_i^m,$$

where S_i' is a subsystem of C_j. If $Reliability(C_j) \geq Reliability(C_{j-1})$, then the uncertainty present in the reconfigured system is less than the uncertainty that existed in the current system. The reliability measurement will allow the reconfigured system to be deployed. However, if $Reliability(C_j) < Reliability(C_{j-1})$, then there is more uncertainty present in the reconfiguration. This would suggest to determine the subsystem(s) of C_j that are responsible for lowering the overall reliability.

5 CONCLUSIONS AND RESEARCH DIRECTIONS

The goal of this paper is to show that a rigorous approach to reliability prediction of E-Commerce systems is possible, when we construct a formal model of it. The E-Commerce model introduced in this paper is simple, yet a realistic model which is complex enough to serve as a test bed for experimentation. In a practical setting, the number of E-Commerce components and their interactions will be much larger. There are also other factors such as resource constraints, load factor, and communication complexity. From a reliability point of view, we require a good formal model which takes these factors into account. In the formal model proposed in this paper the load factor and communication delays can be brought in as synchronization constraints, and resources can be modeled within each class and timing constraints may be imposed on database transactions. Calculation of transition probabilities for large evolving configurations involves multiplying fairly large matrices. The density of the transition probability matrix of a system depends on the number of transitions in the product matrix, which due to synchronization constraints, might be sparse. The sparsity of the matrix and the availability of very fast powering and multiplication algorithms for matrices may be used to speed up reliability calculation for changing configurations.

Our research is based on the premise that E-Commerce systems should be analyzed for several improperties before deployed for public use. Security and reliability are two important properties. Reliability promotes trust and security preserves trust. We have a full implementation of the reliability calculation based on the methods discussed in this paper. One of our ongoing efforts is centered on formalizing security policies and integrating them with a component-based implementation of our model.

REFERENCES

In *IEEE Standard Glossary of Software Engineering Terminology*. IEEE 610.12.1990.

In *FIPA Communicative Acts Library. FIPA Specification Repository (2001)*. Foundation for Intelligent Physical Agents, G.

Alagar, V., Holliday, J., Thiagarajan, P., and Zhou, B. (2002). Agent types and their formal descriptions. In *Technical Report, Department of Computer Engineering, Santa Clara University, Santa Clara, California, U.S.A*.

Franklin, S. and Graesser, A. (1996). Is it an agent or a program?: A taxonomy for autonomous agents. In *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*. Springer-Verlag.

Kallepalli, C. and Tian, J. (Nov. 2001). Measuring and modeling usage and reliability for statistical web testing. In *IEEE Transactions on Software Engineering, (Vol.27, No.11), pp.1023–1036*. IEEE.

Ormandjieva, O. (2002). Deriving new measurements for real-time reactive systems. In *Ph.D. thesis, Concordia University, Montreal, Canada*.

Papazoglou, M. P. (April 2000). Agent-oriented technology in support of e-business. In *Communications of the ACM, Vol.44. No.4*.