Discrete-time dynamic modeling for software and services composition as an extension of the Markov chain approach

Antonio Filieri, Carlo Ghezzi, Alberto Leva and Martina Maggio

Abstract— **Discrete Time Markov Chains (DTMCs) and Continuous Time Markov Chains (CTMCs) are often used to model various types of phenomena, such as, for example, the behavior of software products. In that case, Markov chains are widely used to describe possible time-varying behavior of "self-adaptive" software systems, where the transition from one state to another represents alternative choices at the software code level, taken according to a certain probability distribution. From a control-theoretical standpoint, some of these probabilities can be interpreted as control signals and others can just be observed. However, the translation between a DTMC or CTMC model and a corresponding first principle model, that can be used to design a control system is not immediate. This paper investigates a possible solution for translating a CTMC model into a dynamic system, with focus on the control of computing systems components. Notice that DTMC models can be translated as well, providing additional information.**

I. Introduction

Markov chains are widely used to model random memoryless processes, where the next state depends only on the current state and not on the sequence of events that preceded it, the book [1] discusses theoretical background and some applications of the mentioned modelling formalism, spanning from biology and chemistry to finance; Markov chains are often used also to model computing systems or their components. In the meanwhile, feedback control of computing systems recently is becoming a very popular research topic [2] and the recognized need for adaptation is generally resulting in the introduction of feedback loops, aimed at online adjusting the behavior of the system so as to match some target values and fulfill the given requirements.

Starting from the possibility of controlling a target computing systems that could be modelled as a Discrete Time Markov Chain (DTMC) [3] or Continuous Time Markov Chain (CTMC) [4] we investigated the general problem of translating the given model into a set of equations that could be used in general to provide specific control solutions.

With respect to the mainstream literature on C/DTMCs, see e.g. again [4], the novelty of the proposed approach stems from the highly application-independent problem reformulation in the dynamic system domain, which opens the way to the use of powerful techniques. With respect to the authors' previous research, this paper shows the applicability of the idea to both DTMCS and CTMCS and provides the general framework just envisaged, building on the particular problem of reliability addressed in [3, Section III].

Coming back to the subject of this paper, in general, the goal of the control solution to be devised is to preserve one or more properties of the overall system. For example, again in computing systems, DTMCs and CTMCs are known as a useful formalism to describe systems from the reliability viewpoint and to support reasoning about it. In this case, the chain can easily model web service invocations that may fail using two transitions exiting a certain state and reaching two others, respectively representing success and failure. In this paper we discuss how it is possible to translate a Markov model into a corresponding set of equations, also introducing into the model useful control-related quantities. We will use examples from computing systems control, where the distribution of the incoming entities could for example mean the number of request entering a system and being dispatched through different states. However, the concepts behind the translation are completely general and could in principle be applied to any other domain.

It is worth noticing that most of the approaches adopted in literature for the control of Markov processes cover only special cases. A general control approach for CTMCs has been proposed by Brockett [5]. In this case, the goal of the controller is to set the value of control transition rates in order to obtain a prescribed performance measure, through optimal control. Rates are considered also in this work, as well as transition probabilities that could be influenced by a controller decision. However, our focus is more on the modelling side, discussing the translation of the Markov model into a nonlinear equation system. Also, we explicitly model the queues, which were not taken into account in previous analysis.

The remaining of this paper is organized as follows: Section II describes the generic DTMC and CTMC models and an example of computing systems' entities that are modelled through the formalism. Section III discusses the generic proposed translation, while an example is shown in Section IV. Finally, conclusions are drawn in Section V.

This research has been partially funded by the European Commission, Programme IDEAS-ERC, Project 227977- SMScom.

A. Filieri, C. Ghezzi and A. Leva are with the Dipartimento of Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy {filieri, ghezzi, leva}@elet.polimi.it

M. Maggio is with the Department of Automatic Control, Lund University, Ole Römers väg 1, 223 63, Lund, Sweden martina.maggio@control.lth.se

II. Discrete and Continuous Time Markov Chains

In this paper we mostly focus on software systems for which the behaviour can be modelled through a Markov chain. Markov chains proved to be a valuable mean to link together a system's structure and its behaviour, by allowing randomness to account for unpredictable factors, such as users' behaviour.

The rational for software behaviour modelling lies in abstracting the execution in finite set of significant states. A state may represent either the execution of a certain task of the software control flow or an observable condition, such as the completion of the execution or the occurrence of a failure [6]. The transition form a state to another is structurally allowed by the control flow, but its likelihood depends on user inputs and environmental condition. For example, depending on her (changeable) needs, a user may select specific functionalities, or an external service may fail when invoked. Environmental variability affect the actual behaviour of the system and is captured by transition probabilities and the random residence time in each state.

The two Markov models we consider here are Discrete Time Markov Chains (DTMCs) and Continuous Time Markov Chains (CTMCs). DTMCs can describe the probabilities of moving from a state to another, while CTMCs extends DTMCs by adding a random processing rate to each state, definable thorough an exponential probability distribution. DTMCs have been widely used to reason about usage profile and reliability [7], [8], while CTMCs are suitable to deal with performance, thanks to their ability to represent system's throughput by its processing rate [9].

In adaptive software, the probability value of certain transitions, as well as the processing rate of certain states can be controlled. For example, transitions outgoing from a certain state may correspond to the selection of different implementation alternatives, or the deployment of a component can change its actual processing rate.

Convenient control strategy can thus be applied to make the system satisfy its non-functional requirements, such as to provide the desired reliability or to offer a convenient processing rate [3].

In this section we provide a formal definition of DTMCs and CTMCs. In Section III we will then show how Markov models can be conveniently translated into corresponding dynamic systems and enhanced to deal with queues of requests hold in each computation step.

A. Discrete Time

A Markov chain is a discrete-time random process with the Markov property, i.e., memoryless [10]. A discrete-time random process involves a system which is in a certain state at each step, with the state changing randomly between steps. The steps are often thought of as moments in time, but they can equally well refer to physical distance or any other discrete measurement.

Formally, a *finite* and *labelled* DTMC is a tuple (S, s_0, P, L) where [11]:

- S is a finite set of states,
- s_0 is the *initial state*,
- $P : S \times S \rightarrow [0, 1]$ is a stochastic matrix (i.e. $\forall s_i \in S$ $\sum_{s_j \in S} P(s_i, s_j) = 1$),
- $L : S \rightarrow 2^{AP}$ is a labeling function that marks every state s_i with the Atomic Propositions (AP) that are true in s_i.

We will interchangeably use both the notation $P(s_i, s_j)$ and p_{ij} to refer to the entry (i, j) of the matrix P, corresponding to the probability of moving from state s_i to state s_j. A path through a DTMC is a (possibly infinite) sequence of states $\pi = s_0 s_1 s_2 ...$, where s_{i+1} is reachable from s_i through a transition. The probability matrix P induces a probability space on the set of all possible paths [12]. The probability of a path π with length n can be defined as:

$$\begin{cases} Pr(\pi) = 0 \\ \qquad \text{if } n = 0 \text{ i.e. } \pi = s_0 \\ Pr(\pi) = P(s_0, s_1) \cdot P(s_1, s_2) \cdot ... \\ \qquad \cdot P(s_{n-2}, s_{n-1}) \\ \qquad \text{if } n > 0 \end{cases} \tag{1}$$

For the purposes of this analysis, the model that we want to exploit (for example the execution of a software application or a chemical reaction) is completely described by its *trace*, that is, the corresponding path through the DTMC. We assume, without lack of generality, that there is a single initial state s_0. Also, the notion of *absorbing states* should be introduced. A state s_i is said to be absorbing iff $P(s_i, s_i) = 1$[1].

The properties we are interested to control on a DTMC are the *reachability* probabilities, that is the probability that the execution, started in a transient state s_i (usually s_0), reaches a target state s_j representing the successful completion in any number of time steps.

In the software application case, one could use these properties to represent software reliability, i.e. the probability of successfully execute the global control flow, based on the observer failure probabilities of each task (that may change during time).

B. Continuous Time

DTMCs can model *discrete time* only, that is the transition from a state to another occurs only at discrete time steps. In many applications it would be useful to relax such constraint and to allow transitions to happen at any point of a real-valued time. CTMC provides such relaxation by still providing useful mathematical procedures to verify relevant properties of the system, such as the global processing rate of a complex control flow.

[1]Notice that for the sake of our analysis, the other popular definition of absorbing state as a state to which the system will always eventually return with probability 1, can be reduced to the one in this paper. Essentially, a recurrent state must be alone or part of an absorbing components cluster in the DTMC graph, that can be reduced to a single representative state, that is absorbing as for our definition.

CTMCs extend DTMCs by assigning to each state a *rate* $r_i \in \mathbb{R}_{\geq 0}$, representing the processing rate of that node. This value has to be minded as the rate parameter of an exponential distribution describing the residence time of the execution in the state [10].

Formally, a *finite* and *labelled* CTMC is a tuple (S, s_0, P, L, R) where [11]:

- S, s_0, P, and L are defined as for DTMCs,
- $R : S \to \mathbb{R}_{\geq 0}$ is the *rate vector*, assigning each state to its rate

Every time the execution reaches a state s_i, it will reside there for a random time from $\varepsilon(r_i)$, where $\varepsilon(r_i)$ represents the exponential distribution with rate parameter r_i [13]. When the residence time expires, the execution makes a random move toward the next state according to the transition matrix P.

A path through a CTMC is a (possibly infinite) sequence $s^0 t^0 s^1 t^1 s^2 \ldots$ where t^i represents the residence time in state s^i. A formal definition of it probability can be found in [12]. Informally, in can be computed as the joint probability of the sequence $s^0 s^1 s^2$, as for DTMCs, and the probabilities that the exponentially distributed residence time in each state s^i is t^i.

A different standpoint toward CTMC combine the information of P and R in a rate matrix $\mathbf{R} : S \times S \to \mathbb{R}_{\geq 0}$. An entry $\mathbf{R}(s_i, s_j)$ corresponds to the value $r_i \cdot p_{ij}$, and represents the rate of requests leaving reaching s_j and delivered by s_i. Though the two definitions are equivalent, in the following we will consider separately P and R to easy the exposition in Section III.

The properties of CTMCs we are looking to control concern its *transient behaviour* [12], that is we are interested in the probability that the process, having started in a state s_i (usually s_0), reaches a state s_j (representing the completion of the execution) within the time t imposed by software requirements. In a CTMC setting, this corresponds to ensure a processing rate of (at least) $1/t$.

In the software application case, one could use these properties to represent both software reliability and performance, i.e. the global execution time (or, correspondingly, processing rate) of the control flow, given the observed processing rates of and failure probabilities of its tasks (both task's processing rate and failure probability may change during time).

III. From a Chain to a Dynamic System

The generic node of a chain is shown in Figure 1. In this case, two arcs are entering the node s_i and four arcs are leaving it. If the reference model is a DTMC, the outgoing arcs from s_i represents a probabilistic distribution, hence the probability values on its labels have to sum up to 1. This constraint does not apply in case labels represent rates of a CTMC.

The translation of the Markov chain to a set of equations is almost direct by considering the balance of request flows at each node. Nonetheless, the introduction of additional

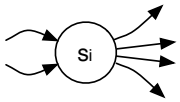

Fig. 1. Representation of a single node.

information is valuable to perform more informative analyses and to allow for more complex analysis and control synthesis. Specifically we want to add to each state a value corresponding to the number of requests waiting to be processed. Such value provides an explicit notion of load and leads the path to enclose convenient queue management strategies for finer grain control of a system's quality of service.

Though we require designers and monitoring systems to fill-in more information than for basic Markov chains, it is usually the case for software systems, especially service-oriented ones, that service rates, distribution probabilities, and queues length could be gathered from the running instances with little effort [14].

Our aim is writing first principle equations involving flows entering and exiting the node, therefore we use the concept of rates. Whenever rates are not available, they can be easily set to one, therefore allowing a DTMC to be modeled as well, also when no additional information is provided. As can be seen in Figure 2, each node has incoming rates of requests and an outcoming rate, that depends on the number of requests that the node has in its own queue, that will be in the following denoted with n and on a control signal u, that could for example account for the ability to hold some of the requests to obtain a certain throughput. In this respect, we introduce explicit modeling of the internal queues of each node, which is not available with any Markov formulation.

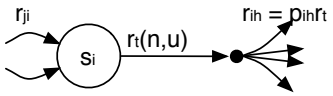

Fig. 2. Representation of a single node within the translation.

The node has an outgoing rate that is indicated with $r_t(n, u)$ in the Figure and is distributed to the connected nodes according to the probability distribution specified by the original Markov chain model. Our state variable is n, i.e., the number of requests that are in the node queue. For a generic node one could write the equation

$$\begin{aligned} n(k) &= n(k-1) + T_s \textstyle\sum_j r_{ji}(k-1) \\ &\quad -T_s \textstyle\sum_h r_{ih}(k-1, n(k-1), u) \end{aligned} \tag{2}$$

where T_s represents the sampling time and the equation is saying that the number of requests that the node has at time k is the previous amount plus the number of incoming request minus the number of processed ones. The equations needs to be coupled with the constraint that

$$p_{ih} = \frac{r_{ih}}{\sum_h r_{ih}} \tag{3}$$

to ensure that the probabilities of the original chain are preserved in the case of a DTMC. In the case of a CTMC, probabilities and rates just need setting up such that the original relationship is preserved and the constraint should be ensured to hold.

One may therefore write, for a single node, that

$$\begin{aligned} n(k) &= n(k-1) + T_s \textstyle\sum_j r_{ji}(k-1) \\ &\quad -T_s \textstyle\sum_h p_{ih} f(n,u) \end{aligned} \tag{4}$$

where the outgoing rate was substituted with a function of the system state, the n term, and of some control action u. At the same time, there are cases in which the probability p_{ih} can be itself a control signal. For example, in the case of computing system modeling, it can represent the choice of a service A over a service B to perform the same operation on some data, where the chain should be representing software service composition and invocation rates and probabilities are included in the system model.

If the introduced function f does not depend on n, the problem is easily to be solved. In general, however, f describes how each node, autonomously or responding to an external input, choose the outgoing rate, therefore it is not possible to assume it does not depend on the number of incoming requests. The relationship between the queued request and the incoming ones make it *dynamically* depend also on the incoming rate.

Denoting with $N(k)$ the column vector representing the system's state and with $F(N(k), U(k))$ the column vector where each element is the function $f_i(n_i, u_i)$ of the i-th node at time k, the state equation of the overall model can be rewritten as

$$\begin{aligned} N(k) &= N(k-1) + T_s R_i(k-1) \\ &\quad +T_s P(k-1) F(N(k-1), U(k-1)) \\ &\quad -T_s F(N(k-1), U(k-1)) \end{aligned} \tag{5}$$

where R_i is the vector of external input rates and P is the probability matrix of the chain. The sum of the two terms $T_s R_i(k-1)$ and $T_s P(k-1) F(N(k-1), U(k-1))$ represents the vector of input rates for each node while the term $T_s F(N(k-1), U(k-1))$ stands for the vector of the total output rates.

Also, one could write as output equation

$$Y(k) = g(F(N(k), U(k)) \tag{6}$$

saying that the output of the system is some kind of combination of the vector of output rates.

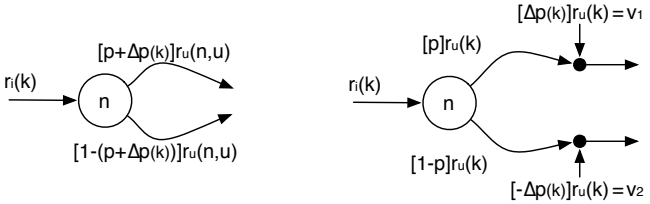

Fig. 3. Managing a variant transition probabilities P matrix via the introduction of fictitious nodes.

Let's start with a couple of simple cases, where the components of (5) have some nice properties. First, if the probability matrix P is constant and $f(n,u) = f(n) + f(u)$ one could write

$$\begin{aligned} N(k) &= N(k-1) + T_s R_i(k-1) \\ &\quad +T_s (P-I) F_n(N) \\ &\quad +T_s (P-I) F_u(U). \end{aligned} \tag{7}$$

Moreover, if one assumes $f_n(n) = kn$, the state equation becomes

$$\begin{aligned} N(k) &= N(k-1) + T_s R_i(k-1) \\ &\quad +T_s (P-I) K N(k-1) \\ &\quad +T_s (P-I) F_u(U) \\ &= [I + T_s (P-I) K] N(k-1) \\ &\quad +T_s R_i(k) + T_s (P-I) F_u(U) \end{aligned} \tag{8}$$

and taking $F_u(U) = \tilde{U}$ this becomes a LTI system. Also, being the values of P in the interval between zero and one, as K, it surely exist a range of sampling times T_s such that the system is asymptotically stable.

In addition, it is possible to bring back the case of a non-constant P matrix, for example due to the presence of probabilities treated as control signals, introducing fictitious input/output nodes.

To briefly discuss this, refer to Figure 3. On the left a node is shown – with just two outputs for simplicity and without loss of generality – where the probability p of routing to one output is expressed as a nominal value $\overline{p}$ (whatever is meant for "nominal") plus an additive variation Δp, representing in the addressed context a correction to $\overline{p}$ coming from some controller. There are thus two control inputs (u and Δp) to alter respectively the overall throughput and its distribution. On the right, the same effect is obtained by adopting as control variables the quantities denoted by v_1 and v_2.

In fact, translating the system on the left of Figure 3 into that on the right, a LTI one can be obtained under the sole hypothesis that the output rate comes from

$$f(n,u) = Kn + u \tag{9}$$

which is quite reasonable, at least in the vicinity of the nominal operating point of choice. To show that, write

$$\begin{aligned} n(k) &= n(k-1) + T_s \left[r_i(k-1) \right. \\ &\quad \left. -f(n(k-1), u(k-1)) \right] \end{aligned} \tag{10}$$

thus

$$\begin{aligned} r_a(k) &= \left(\overline{p} + \Delta p(k)\right) f(n(k), u(k)) \\ r_b(k) &= \left(1 - \overline{p} - \Delta p(k)\right) f(n(k), u(k)) \end{aligned} \tag{11}$$

and bringing (9) in, with trivial computations,

$$\begin{aligned} r_a(k) &= \overline{p}\left(Kn(k) + u(k)\right) \\ &\quad +\Delta p(k)\left(Kn(k) + u(k)\right) \\ r_b(k) &= \left(1 - \overline{p}\right)\left(Kn(k) + u(k)\right) \\ &\quad -\Delta p(k)\left(Kn(k) + u(k)\right) \end{aligned} \tag{12}$$

which has u and Δp as controls, but is apparently nonlinear. Adopting the translation shown on the right of Figure 3, on the other hand, one would write

$$\begin{aligned} n(k) &= n(k-1) + T_s r_i(k-1) \\ &\quad -T_s\left[\overline{p} r_i(k-1) + v_1(k-1)\right] \\ &\quad -T_s\left[(1-\overline{p}) r_i(k-1) + v_2(k-1)\right] \\ &= n(k-1) - T_s\left[v_1(k-1) + v_2(k-1)\right] \end{aligned} \tag{13}$$

leading for the rates to the LTI expressions

$$\begin{aligned} r_a(k) &= \overline{p} r_i(k) + v_1(k) \\ r_b(k) &= (1-\overline{p}) r_i(k) + v_2(k) \end{aligned} \tag{14}$$

having v_1 and v_2 as control inputs. Note that (12) and (14) are related, since by setting $\Delta r = Deltap\,(Kn + u)$, Equation (12) becomes

$$\begin{aligned} r_a(k) &= \overline{p}\,(Kn(k) + u(k)) + \Delta r(k) \\ r_b(k) &= (1-\overline{p})\,(Kn(k) + u(k)) - \Delta r(k) \end{aligned} \tag{15}$$

thus comes to exhibit – with u and Δr as contro inputs – the same structure as Equation (14).

To summarize, the proposed model extends the formalism of Markov Chains, which is a widely used modeling framework in different contexts, although our extension addressed more closely computing systems, that originated our initial case study. Our extension includes queues (and service rates, if one starts from a Discrete Time Markov Chain) and we explicitly write the balance equation for each node. The resulting model is not particularly complex, even though being nonlinear. It can however be reduced to a LTI model in particular cases.

IV. Example

In this section we will show a (small) example, adapted from [6], of how a software can be modelled with a Markov chain and subsequently how its equations can be derived. Other Markov chain models of software applications can be found in [6], [11], [3], [7], [8], [9] where the focus is on the translation from design models or code to Markov chains.

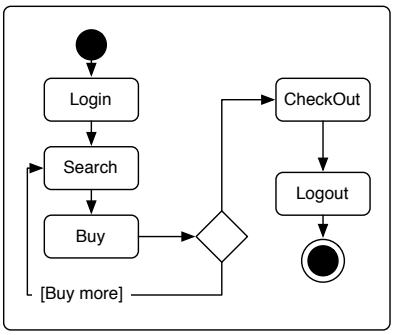

Fig. 4. Activity diagram of the example.

The target application is an e-commerce website that relies to external services for the buying operation. It is for example the case of a bundle for purchasing products offered by different external providers. Once entering the web application the user can decide to buy a product and checkout or repeat the purchase operation to add more items before checking out. The activity diagram of the proposed example can be found in Figure 4. Notice that each external service has a different success probability, in fact the third party service could be unavailable or the invocation could fail. We assume success probabilities to be independent one another. Suppose, for clarity, that only two third party services are available for the *buy* operation. The control objective in this case is to choose between the two services based on their service rates and failure probabilities. Such values may be subject to changes because of external factors, out from our control. The goal is to distribute the incoming traffic between the two external services.

The Markov chain that models the software can be found in Figure 5, where the login state is subsequently followed by a search state and some products will be bought, with probability p from the first service and with probability $1-p$ from the second one. We assume that all the products are available from both the external services, so that their selection will only be based on the provided quality of service. p can be seen as a control signal, since the software itself could decide whichever service to reroute the request to. Contrarily, the user could subsequently do another search with probability pR or checkout with probability $1-pR$, the value of this probability can only be observed (and possibly predicted based on past data) but cannot be directly influenced since dependent on users' behaviour. The figure depicts only the basic software behaviour, without considering possible failures of its components. Such a case can be enclosed by adding a special *failure* state to be avoided, as in [6].

Based on the considerations of Section III, the equations for such a chain are

$$\begin{aligned} n_0(k) &= n_0(k-1) + T_s r_i(k-1) - T_s f_0(n_0, u_0) \\ n_1(k) &= n_1(k-1) + T_s f_0(n_0, u_0) \\ &\quad +T_s(pR) f_2(n_2, u_2) + T_s(pR) f_3(n_3, u_3) \\ &\quad -T_s f_1(n_1, u_1) \\ n_2(k) &= n_2(k-1) + T_s p f_1(n_1, u_1) - T_s f_2(n_2, u_2) \\ n_3(k) &= n_3(k-1) + T_s(1-p) f_1(n_0, u_0) \\ &\quad -T_s f_3(n_3, u_3) \\ n_4(k) &= n_4(k-1) + T_s(1-pR) f_2(n_2, u_2) \\ &\quad +T_s(1-pR) f_3(n_3, u_3) - T_s f_4(n_4, u_4) \\ n_5(k) &= n_5(k-1) + T_s f_4(n_4, u_4) - T_s f_5(n_5, u_5) \\ n_6(k) &= n_6(k-1) + T_s f_5(n_5, u_5) \end{aligned} \tag{16}$$

where r_i represents the number of incoming users, f_0 could be defined as any mechanism for admission control that can be found in the literature for controlling a web server, see e.g., [15], [16], [17], f_1 can take into account the ability to hold some requests into the server and process them in a future time unit, for example as done for mobile phone platforms in a view to lower power and energy consumption [18]. Similar considerations hold for the other nodes of the chain. Notice that, whenever $f_x(n_x, u_x)$ is written, it

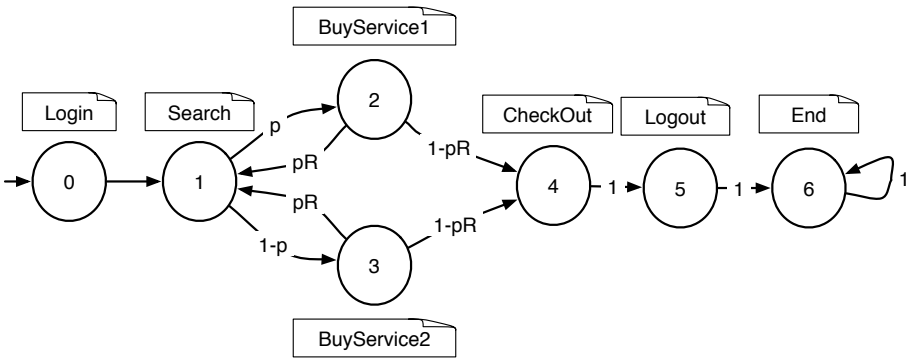

Fig. 5. Markov chain corresponding to the example.

should read $f_x(n_x(k-1), u_x(k-1))$ but the formulation was shortened to the advantage of clarity.

Notice that in this case the only node that has an external incoming rate is n_0, where the term r_i appears, also, some of the functions f_x could be simplified if one assumes there is no control mechanism for those nodes. For example, the logout state n_5 may have no delay and no control, therefore $f_5(n_5, u_5)$ could be just substituted with $n_5(k-1)$. This formulation would allow to model different control strategies, e.g., resource control and admission control, within a unified framework.

V. Conclusion and future works

In this paper we investigated the translation of DTMCs and CTMCs into a set of equations, extending the formalisms with a simple notion of queue. Markov models are widely adopted to represents software behaviour, as well as in many other research fields. The explicit quantification of the number of waiting requests would allow for more expressive modeling of the quality of service, as well as the ability to represent in a unified model scheduling and queues management strategies.

The equation model obtained through our translation is in general nonlinear but not particularly complex. In several cases it can be reduced to a LTI one. Some control related issues were addressed as well, such as the specification of control variables in the form of transition probabilities and processing rates.

We are enhancing our first principle modeling of the queues to support for the representation of standard queues features, such as finite queues and ordering. We aim at defining convenient methodologies to provide a general control strategy suitable for systems representable via DTMCs or CTMCs with state queues, addressing the satisfaction of non-functional requirements for software systems as well as analogous counter-parts in other fields.

Acknowledgment

This research has been partially funded by the European Commission, Programme IDEAS-ERC, Project 227977-SMScom. Also, this work was supported by the Swedish Research Council through the LCCC Linnaeus Center.

References

[1] J. R. Norris, *Markov chains*, ser. Cambridge series in statistical and probabilistic mathematics. Cambridge University Press, 1998.

[2] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury, *Feedback control of computing systems*. Wiley Online Library, 2004.

[3] A. Filieri, C. Ghezzi, A. Leva, and M. Maggio, "Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements," in *ASE*, P. Alexander, C. S. Pasareanu, and J. G. Hosking, Eds. IEEE, 2011, pp. 283–292.

[4] X. Guo and O. Hernández-Lerma, *Continuous-Time Markov Decision Processes: Theory and Applications*, ser. Stochastic modelling and applied probability. Springer Verlag, 2009.

[5] R. Brockett, "Optimal control of observable continuous time markov chains," in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*. IEEE, 2008, pp. 4269–4274.

[6] A. Filieri, C. Ghezzi, and G. Tamburrelli, "A formal approach to adaptive software: continuous assurance of non-functional requirements," *Formal Aspects of Computing*, pp. 1–24.

[7] R. C. Cheung, "A user-oriented software reliability model," *IEEE Transaction on Software Engineering*, vol. 6, no. 2, pp. 118–125, 1980.

[8] L. Cheung, R. Roshandel, N. Medvidovic, and L. Golubchik, "Early prediction of software component reliability," in *ICSE, Leipzig, Germany, May 10-18, 2008*. ACM, 2008, pp. 111–120.

[9] R. Calinescu, L. Grunske, M. Z. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic qos management and optimization in service-based systems," *IEEE Transaction on Software Engineering*, vol. 37, no. 3, pp. 387–409, 2011.

[10] S. Ross, *Stochastic Processes*. Wiley New York, 1996.

[11] M. Kwiatkowska, "Model checking for probability and time: from theory to practice," in *Logic in Computer Science, 2003. Proceedings. 18th Annual IEEE Symposium on*, June 2003, pp. 351 – 360.

[12] C. Baier and J.-P. Katoen, *Principles of Model Checking*. The MIT Press, 2008.

[13] W. Pestman, *Mathematical statistics: an introduction*. Walter De Gruyter Inc, 1998, vol. 1.

[14] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Model evolution by run-time parameter adaptation," in *ICSE*, 2009.

[15] M. Kihl, A. Robertsson, A. Andersson, and B. Wittenmark, "Control-theoretic analysis of admission control mechanisms for web server systems," *World Wide Web*, vol. 11, no. 1, pp. 93–116, 2008.

[16] Q. Sun, G. Dai, and W. Pan, "LPV model and its application in web server performance control," in *Proceedings of the 2008 International Conference on Computer Science and Software Engineering - Volume 03*, ser. CSSE '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 486–489. [Online]. Available: http://dx.doi.org/10.1109/CSSE.2008.1219

[17] N. Gandhi, D. Tilbury, Y. Diao, J. Hellerstein, and S. Parekh, "Mimo control of an apache web server: modeling and controller design," in *American Control Conference, 2002. Proceedings of the 2002*, vol. 6, 2002, pp. 4922 – 4927 vol.6.

[18] M.-R. Ra, J. Paek, A. B. Sharma, R. Govindan, M. H. Krieger, and M. J. Neely, "Energy-delay tradeoffs in smartphone applications," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, ser. MobiSys '10, 2010, pp. 255–270.