

UML Modelling and Performance Analysis of Mobile Software Architectures

Vincenzo Grassi and Raffaella Mirandola

Dipartimento di Informatica, Sistemi e Produzione
Università di Roma "Tor Vergata"
via di Tor Vergata, 00133 Roma, Italy
{grassi, mirandola}@info.uniroma2.it

Abstract. Modern distributed software applications generally operate in complex and heterogeneous computing environments (like the World Wide Web). Different paradigms (client-server, mobility based, etc.) have been suggested and adopted to cope with the complexity of designing the software architecture of distributed applications for such environments, and deciding the "best" paradigm is a typical choice to be made in the very early software design phases. Several factors should drive this choice, one of them being the impact of the adopted paradigm on the application performance. Within this framework, the contribute of this paper is twofold: we suggest an extension of UML to best modeling the possible adoption of mobility-based paradigms in the software architecture of an application; we introduce a complete methodology that, starting from a software architecture described using this extended notation, generates a performance model (namely a Markov Reward or Decision Process) that allows the designer to evaluate the convenience of introducing logical mobility into a software application.

Keywords. Distributed systems, Architecture modelling, Extensions, Performance analysis, Mobile components

1 Introduction

Distributed software systems operate in large scale computing environments (like the World Wide Web), where the computing sites are both geographically and logically distributed, and some of them may be even physically mobile. Because of the complexity of such systems, it is widely recognized that decisions taken in the first phases of the software lifecycle may have a strong impact on the quality of the final product, thus suggesting the need of an early assessment of non functional quality requirements, like performance. Some of these early decisions concern the architectural organization of the software system, that is its organization in terms of components and patterns of interaction among them [3]. The *client-server* interaction paradigm represents a well established solution in this sense, typically based on the assumption that components are bound to a given site, and may be even unaware of their respective locations during their interactions. However, it has grown the concern that such a "location unaware" perspective could not be appropriate to a context where interactions among components may have very different characteristics and

constraints, depending on components location. For this reason, there is a growing consensus about the idea of introducing the concept of “location” already at the design stage. As a consequence, “location aware” interaction paradigms have emerged in recent years, based on the notion of *code mobility* [7], where components of an application may (autonomously, or upon request) decide to move themselves to different locations, during the application lifetime. Modern software technologies (e.g. Java) provide the tools to implement these paradigms. Among the arguments in favor of these new paradigms there is the consideration that in a large scale distributed environment it may be convenient to move components close to their current partners, with the goal of transforming non local interactions into local ones. The rationale behind this idea is that local interactions are more efficient, and hence code mobility could improve the system efficiency. However, moving components does not come at no cost, so it is also recognized that the arguments in favor of code mobility are not valid in general, and hence the choice between location unaware interaction paradigms like client-server, and location aware ones like code mobility should be performed on a case-by-case basis [7].

As a consequence, our aim is to provide tools that support the designer of complex distributed software systems, in making the architectural choices that are more promising for the software system under development. To this purpose, the methodology we present in this paper gives insights about the impact of the two above mentioned interaction paradigms on some performance-related quality attributes. To reach this goal, the methodology exploits information extracted from design artifacts to build a model of the system dynamics. The model generation procedure lends itself to automation, to minimize the effort required to the designer to use this methodology. To guarantee a wide applicability of this methodology, we assume that the design artifacts are expressed in a UML-based notation, since this formalism represents a *de facto* standard in the field of industrial software design. The model we derive from the application description expressed in UML is a Markov decisional process (MDP), since we intend to exploit its stochastic-decisional features to model both the uncertainty about the system dynamics and the (possibly optimal) design choices that can be adopted, and that can have different impacts on the system performance.

The paper is organized as follows. In section 2, we briefly introduce the current UML approach to mobility modeling, as outlined in [4], and then describe our extensions to this formalism to express the possible use of mobile code paradigms in the design of a software system. In section 3, we present some basic notions about MDPs. In section 4 we briefly discuss how we exploits the features of this kind of model in our framework, and then present an algorithm to derive a suitable MDP from a software system description expressed in the notation defined in section 2. Related work is reviewed in section 5. Throughout the paper we use a simple example of network management application [1] to show the use of our methodology. Finally, section 6 concludes the paper.

2 UML and Mobility

As remarked in the introduction, code mobility is not only a technology popularized by the success of technologies like Java, but also represents a design paradigm that

can give a new perspective in the way the software architecture of distributed applications is conceived and deployed [7, 18]. From an architectural viewpoint, one of the main impacts of this paradigm is on the design of components interaction, since it can turn non local interactions into local ones. To express and analyze this new paradigm, we need a suitable notation. In the next two subsections, we show the standard UML approach to mobility modeling, and our suggested modification.

2.1 UML Standard Approach

The need of a notation for components mobility in a UML framework has already emerged, so that UML already provides some mechanisms with this goal. They are mainly based on the use of a tagged value `location` within a component to express its location, and of the `copy` and `becomes` stereotypes to express the location change of a component, where the former can be used to specify the creation of an independent component copy at a new location (like in the *code on demand* and *remote evaluation* paradigms), and the latter to specify a location change of a component that preserves its identity (like in the *mobile agent* paradigm) [7]. In [4] these mechanisms are applied to a collaboration diagram (CD). As an example of this modeling approach, figure 1 shows a network management scenario, where a network manager `n` periodically collects informations about the state of some remote devices. In this example, `n` starts a migrating agent `c` that moves from node to node collecting information, eventually delivering the collected information to `n`, for further elaboration. At each node, the collecting agent engages a sequence of `N` message exchanges with a local `DEVICE` component to collect information.

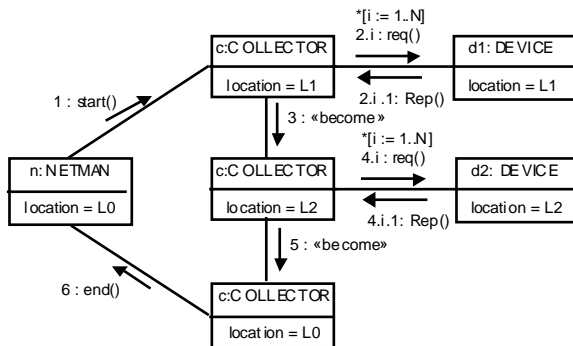


Fig. 1. Network management example, with a migrating agent

However, this modeling approach presents some drawbacks, since it mixes together two different views, one concerning the interaction *style* (e.g. the fact that a component behaves as a mobile agent), and the other one concerning the actual sequence of messages exchanged between components during a particular interaction. Moreover, this approach leads to a proliferation of objects in a CD, that actually represent the same object at different locations. While acceptable when the number of locations and mobile components is very small, this approach quickly leads to hardly read diagrams when the number of components and/or locations grows.

For these reasons we suggest a different approach to model components mobility in a UML framework, as shown in the next subsection.

2.2 Suggested Extension

Our approach is based on the use of both collaboration and sequence diagrams (SD), with a separation of concerns between them, similarly to (Petriu, in [24]). In our case the SD describes the actual sequence of interactions between components, which is basically independent of the adopted style and obeys only to the intrinsic logic of the application, while the CD only shows who interacts with whom and how (i.e. according to which style), without showing the actual sequence of exchanged messages, thus avoiding object proliferation.

In this new approach, we describe the SDs using the standard UML notation, while we suggest an extension of the UML semantics for the associated CD. Indeed, we are interested in distinguishing a style where components are unaware of their respective locations during interactions, and hence do not change them (e.g. client-server style), and a style where they do change location with the goal of turning non local interactions into local ones. To this purpose we introduce a new stereotype `moveTo` that applies to messages in the CD, and that, when present, indicates that the source component moves to the location of its target before starting a sequence of consecutive interactions with it. If no other information is present, this style applies to each sequence of interactions shown in the associated SD, between the source and target components of the `moveTo` message; otherwise a condition can be added to restrict this style to a subset of the interactions between two components. Figures 2.a and 2.b show two CD fragments, for the two styles described above: figure 2.a models a style where neither C_i nor C_j change their location during any interaction between them, while figure 2.b models a style where C_i moves to the location of C_j , before starting any sequence of interactions with it.

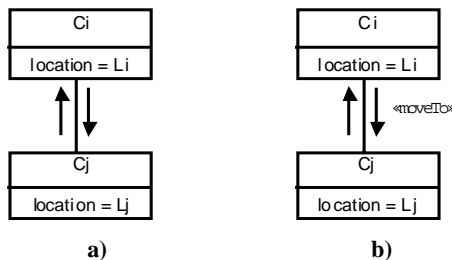


Fig. 2. a) Interaction style without mobility, b) Interaction style with mobility

Note that in the definition of the `moveTo` semantics there is no indication about whether the moving decision is autonomously taken by the source component (as in a *mobile agent* style) or forced in some way by another component (as in *code on demand* or *remote evaluation* styles), neither about the role of the source component in the interaction (e.g. whether it starts interaction or not). This does not mean that in a more complete modeling environment this information is not necessary, but we believe that the adopted notation is sufficient to illustrate the approach we are suggesting.

According to our modeling framework, the application described in figure 1 can be modeled as shown in figures 3 and 4. Figure 3 shows a SD that describes in detail the “logic” of the interaction, i.e. the sequence of messages exchanged among the components. In this diagram no information is present about the adopted style, that is whether or not some component changes location during the interactions. This style information is provided by the CDs in figures 4.a and 4.b. The CD in figure 4.a models a style where component mobility is not considered, and each component is bound to a given location. In particular, this diagram shows that *n* and *c* are colocated, and hence *c* interacts remotely with the devices. On the other hand, for the same application, the CD in figure 4.b corresponds to the case where a mobile style is adopted. More precisely, the diagram shows that only *c* can change location, and according to the `moveTo` semantics presented above, it moves to the location of *n*, *d1* or *d2* before interacting with them. Hence, the location change of *c* can occur at the beginning of each of the four interaction sequences enclosed by a dashed line, shown in figure 5, if at that time *c* and its partner do not have the same location.

Note that in figure 4.b the location of *c* is left unspecified, since it can dynamically change. In general, we could also give it a specified value in the diagram, that would show the “initial” location of the mobile object in an initial deployment configuration.

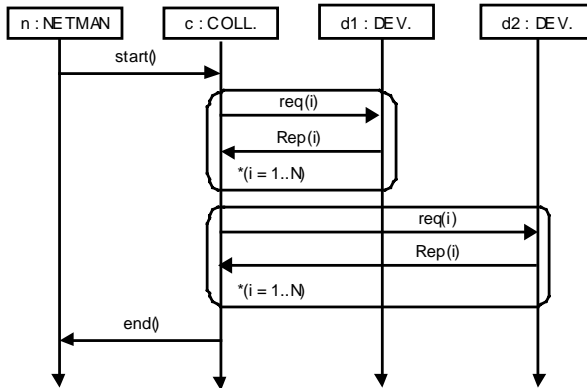


Fig. 3. SD modeling the interaction “logic”

As outlined in the introduction, there is no guarantee that the adoption of a mobile style is always advantageous from a performance viewpoint. Hence, at the early design stage where the above diagrams can be drawn, there can be uncertainty about the opportunity of adopting this style. We suggest that it is worth modeling also this uncertainty within the framework described above; then, in the next section, we will show how we can extract information (including the uncertainty about mobility) to build a system behavior model whose solution provides insights about the opportunity of adopting mobility.

To model uncertainty about mobility of components, we propose a new stereotype `moveTo?`, that extends the semantics of the `moveTo` stereotype described above. When a message between two components in a CD is labeled with `moveTo?`, this means that the source component “could” move to the location of its target at the be-

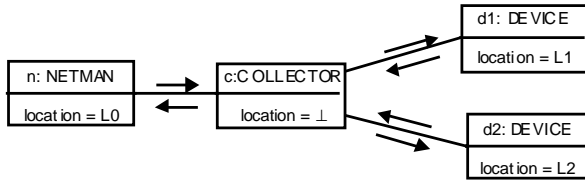


Fig. 4.a. CD modeling an interaction style without mobility

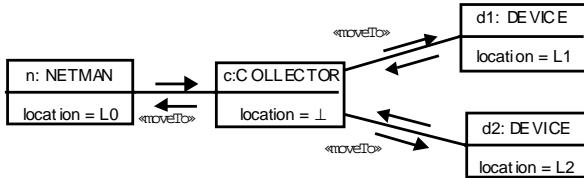


Fig. 4.b. CD modeling an interaction style with mobility

gining of a sequence of interactions with it. In a sense, this means that, based on the information the designer has at that stage, he considers acceptable both types of behaviors. Hence, the general framework we are suggesting to model the interaction style in a distributed software system, consists of a CD where some messages are unlabeled, some can be labeled with the `moveTo` stereotype, and some with the `moveTo?` stereotype. The former two cases correspond to a situation where the designer feels confident enough to decide about the best interaction style, while the latter to a situation where the designer lacks such a confidence. Figure 6 shows the CD for the same example used before, but modeling the case where the designer is uncertain about the effectiveness of designing `c` as a mobile agent.

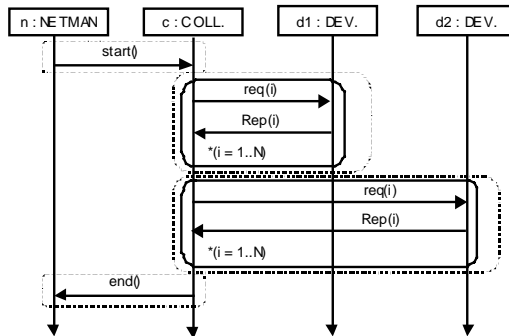


Fig. 5. Distinct sequences of consecutive interactions between components

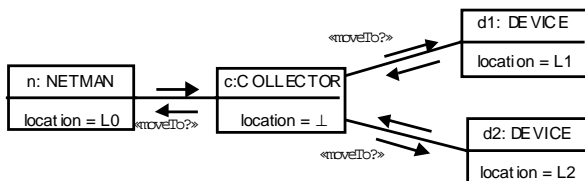


Fig. 6. CD modeling uncertainty about interaction style with mobility

As a final remark, we would like to point out that our proposal does not simplify the overall complexity of the application description, since it basically splits into two diagrams types the information included in one diagram, as in the standard UML notation. Our goal, from the notation viewpoint, is rather to make clearer the model description, thanks to the separation of concerns between the two diagram types.

3 MRP and MDP Models

Our goal is to build a stochastic model that describes the system dynamics, whose evaluation provides information about the performance we can expect by adopting one of the interaction styles described in the previous section. We focus on performance measures that can be expressed as function of the number and types of interactions among components. To this purpose, we assume that the diagrams described in the previous section are augmented with appropriate annotations expressing the “cost” of each interaction, with respect to a given performance measure [24]. For example, if the measure we are interested in is the generated network traffic, the corresponding cost of each interaction is the number of bytes sent over a network link. This cost may be, in general, location dependent.

The stochastic model we build is a Markov Reward Process (MRP) [17] when the CDs modeling the interaction style only use the `moveTo` stereotype, and a Markov Decision Process (MDP) [17] when the CDs modeling the interaction style also use the `moveTo?` stereotype. In the following, we briefly review these two models.

In general, a MRP models a state transition system, where the next state is selected according to a transition probability that only depends on the current state. Moreover, each time a state is visited or a transition occurs, a *reward* is accumulated, that depends on the involved state or transition. Typical measures that we can derive from such a model are the reward accumulated in a given time interval, or the reward accumulation rate in the long period.

A MDP extends the MRP model by associating to each state a set of alternative *decisions*, where both the rewards and the transitions associated to that state are decision dependent. A *policy* for a MDP consists in a selection, for each state, of one of the associated decisions, that will be taken each time that state is visited. Hence, different policies lead to different system behaviors, and to different accumulated rewards. In other words, a MDP defines a family of MRPs, one for each different policy that can be determined. Methodologies exist to determine the optimal policy with respect to some optimality criterion (e.g. minimization of the accumulated reward).

4 Performance Model Generation from UML Diagrams

In this section we show how we can derive a suitable MRP or MDP based on information extracted from the SD and CD that defines the interactions and interaction style among components. For the sake of simplicity, the methodology is defined in terms of a single CD and a single SD, where only sequential or iterative interactions are considered. In case of multiple SDs or conditional selection of

interactions within a SD, the methodology should be appropriately extended, by weighting the contribution of different SDs or alternative interactions within a SD. However, this extension appears quite straightforward, and we feel that the case considered here is enough to give the flavour of our methodology.

In our modeling framework, a system state corresponds to a possible configuration of the components location, while a state transition models the occurrence of an interaction between components or a location change, and the associated reward is the cost of that interaction. Hence we use MRPs or MDPs where the reward is associated to transitions only. In case of MDP, the decisions associated to states model the alternative choices of mobility or nomobility as interaction style between some components, for those components that are the source of a `moveTo?` message.

Since a MRP can be considered as a special case of MDP (obtained when no `moveTo?` message is present) we show only the general methodology for the derivation of a MDP. We first define some elementary generation rules, and then use these rules to define the MDP generation algorithm.

4.1 Elementary Rules

Let C_i, C_j be two components in the CD that models the interaction style, and let (L_a, L_b) denote any system state where the location of C_i and C_j are L_a and L_b , respectively, with $L_a \neq L_b$, while let (L_b, L_b) denote any state where their locations are the same. Moreover, in case of transition from (L_a, L_b) to (L_b, L_b) , we mean that (L_b, L_b) only differs in the shown state component from (L_a, L_b) , while all the other state components maintain the same value as in (L_a, L_b) . The general idea behind the following rules is that the frequency of the interactions between C_i and C_j can be derived from the SD as the ratio between the number of interactions between C_i and C_j and the total number of interactions that can be counted in the SD [23], including in this number also “interactions” corresponding to the location change of components. Similarly, the average cost of an interaction between C_i and C_j is derived as the sum of the cost of all the interactions between C_i and C_j divided by their number. All the costs must be considered location dependent, i.e. they depend on L_a and L_b , in general.

Since a state models a possible configuration of component locations, a transition from a state to itself models a normal¹ interaction between components, while a transition to a different state models a location change.

Rule A: if C_i and C_j are source and target of a non labeled message, then a transition exists from any state (L_a, L_b) or (L_b, L_b) to itself, with frequency and reward given by, respectively:

$$\begin{aligned} \text{freq}((L_a, L_b) \rightarrow (L_a, L_b)) &= \text{freq}((L_b, L_b) \rightarrow (L_b, L_b)) \\ &= \frac{\text{total number of "normal" interactions between } C_i \text{ and } C_j}{\text{total number of interactions}} \end{aligned} \quad (1)$$

¹ In the following, we call “normal” any interaction different from a location change.

$$\begin{aligned}
 \text{rew}((La,Lb)-->(La,Lb)) &= \text{rew}((La,La)-->(La,La)) \\
 &= \frac{\sum \text{cost of an interaction}}{\text{all the interactions between } C_i \text{ and } C_j} \\
 &= \frac{\text{total number of interactions between } C_i \text{ and } C_j}{\text{total number of interactions between } C_i \text{ and } C_j} \tag{2}
 \end{aligned}$$

Fig. 7. Derivation of MDP transitions from CD, according to rule A

Rule B: if C_i and C_j are source and target of a message labeled with the `moveTo` stereotype, then a transition exists from any state (L_b, L_b) to itself, with frequency and rewards given by (1) and (2) respectively; moreover, a transition exists from any state (L_a, L_b) to (L_b, L_b) , with frequency and reward given by, respectively:

$$\begin{aligned}
 \text{freq}((La,Lb)-->(Lb,Lb)) \\
 &= \frac{\text{total number of "move to" interactions from } C_i \text{ to } C_j}{\text{total number of interactions}} \tag{3}
 \end{aligned}$$

$$\text{rew}((La,Lb)-->(Lb,Lb)) = \text{migration cost of } C_i \text{ from } L_a \text{ to } L_b \tag{4}$$

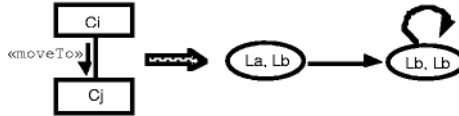


Fig. 8. Derivation of MDP transitions from CD, according to rule B

Rule C: if C_i and C_j are source and target of a message labeled with the `moveTo?` stereotype, then a transition exists from any state (L_b, L_b) to itself, with frequency and reward given by (1) and (2), respectively.

On the other hand, in any state (L_a, L_b) two alternative decisions can be taken: C_i moves or does not move from L_a to L_b . In the former case a transition occurs from (L_a, L_b) to (L_b, L_b) , while in the latter no state change occurs; the corresponding transition frequencies and rewards are:

- decision " C_i moves to L_b ": frequencies and rewards of the transition from any state (L_a, L_b) to (L_b, L_b) are given by (3) and (4), respectively;
- decision " C_i does not move to L_b ": frequencies and rewards of the transition from any state (L_a, L_b) to itself are given by (1) and (2), respectively

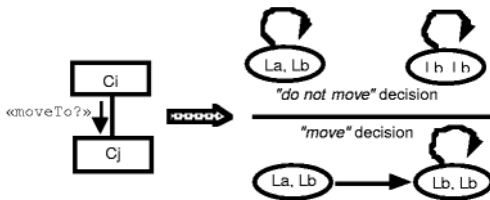


Fig. 9. Derivation of MDP transitions from CD, according to rule C

4.2 State Space Generation

Let C_1, C_2, \dots, C_n be the system components. The set S of all the possible system states can be defined as:

$$S = \text{Loc}(C_1) \times \text{Loc}(C_2) \times \dots \times \text{Loc}(C_n)$$

where $\text{Loc}(C_i)$ is the set of all the possible locations of component C_i , because of its mobility.

$\text{Loc}(C_i)$ can be determined by building an oriented graph G (not necessarily connected) obtained from the CD that models the interaction style among components. The nodes of G correspond to the components in CD, plus additional “pit” nodes, one for each different location value reported in the CD. Oriented arcs in G correspond to messages labeled with the `moveTo` or `moveTo?` stereotypes in the CD; moreover, for each component C_i in CD that has a defined location value L_i , an oriented arc exists in G from the node corresponding to C_i and the pit node corresponding to L_i . $\text{Loc}(C_i)$ is the set of all the pit nodes that can be reached in G starting from the node corresponding to C_i . Figures 10.a and 10.b show the graphs corresponding to the CDs in figure 4.a and 4.b (or 6), respectively, where the pit nodes correspond to square nodes. For example, from the graph in figure 10.b, considering the components ordered as $n, c, d1, d2$, we get $S = \{(L0, L0, L1, L2), (L0, L1, L1, L2), (L0, L2, L1, L2)\}$. For simplicity, since only the location of c can change, in the following we will consider as system state the location of this component only, with $S = \{L0, L1, L2\}$.

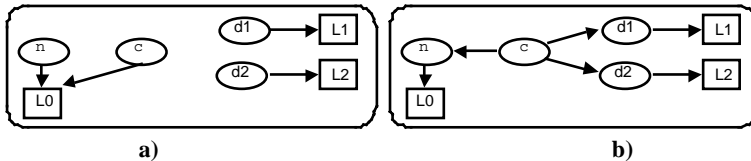


Fig. 10. Graphs of reachable locations for each component. a) for the CD in fig. 4.a, b) for the CDs in figs. 4.b and 6

4.3 MDP Generation

Once we have built the process state space, we can complete the MDP generation. The generation algorithm is presented in appendix 1. In this section we briefly comment the algorithm and then apply it to the network management example.

The core of the algorithm consists of the application of rules *A*, *B* and *C* shown in section 4.1 to each pair of interacting components. In this way we obtain, for each pair, its contribution to the frequencies and rewards of state transitions (construction of the $F(s)$ and $D(s)$ sets in the algorithm). To obtain the overall transition rate between two states, we must sum each contribution obtained in this way, concerning the two states. Similarly, to get the average reward associated to that transition, we must sum the reward coming from each considered contribution, weighted by its relative frequency. Moreover, rule *C* generates contributions to state transitions and rewards that depend on alternative decisions concerning the interaction style between two components. The set of the possible combinations of such decisions concerning any pair of interacting components represents the set of possible “global” decisions in

a state (construction of the DecVec(s) set in the algorithm); each of these decisions is associated with a corresponding set of transition frequencies and rewards.

Let us apply this procedure to the considered example. The performance measure we are interested in is the generated network traffic. Hence, the cost associated to each interaction is the corresponding number of bytes sent over network links, derived from opportune annotations in SD. Let us define:

- s : size in bytes of a start() message;
- r : size in bytes of a req() message;
- R : size in bytes of a Rep() message;
- e : size in bytes of an end() message;
- m : size in bytes of a c component during a location change.

The cost of any interaction is equal to the size of the corresponding message, given above, when the interacting components are not colocated, and zero otherwise.

Now, let us apply the algorithm. As shown in appendix 1, we must first calculate its input. For each pair of interacting components, we get from the SD in figure 5:

- pair (n, c):
 - number of normal interactions: $Ni(n, c) = 2$
 - number of “move to” interactions: $Nm(n, c) = 2$
 - cost of a normal interaction: $Ci(n, c) = (s+e)/2$
 - cost of a “move to” interaction: $Cm(n, c) = m$
- pair ($d1, c$):
 - number of normal interactions: $Ni(d1, c) = 2N$
 - number of “move to” interactions: $Nm(d1, c) = 1$
 - cost of a normal interaction: $Ci(d1, c) = (r+R)/2$
 - cost of a “move to” interaction: $Cm(d1, c) = m$
- pair ($d2, c$):
 - number of normal interactions: $Ni(d2, c) = 2N$
 - number of “move to” interactions: $Nm(d2, c) = 1$
 - cost of a normal interaction: $Ci(d2, c) = (r+R)/2$
 - cost of a “move to” interaction: $Cm(d2, c) = m$

From these values, we get the total number of interactions $N_{tot} = 4N + 6$.

Now, let us apply rules *A*, *B*, and *C* to each state in the set $S = \{L0, L1, L2\}$, considering any possible pair of interacting components. In this way, we obtain the sets $F(s)$ and $D(s)$ of transition frequencies and rewards (possibly associated to alternative decisions), each concerning a particular pair of components. Then, we have to appropriately combine the elements of these sets, to get the overall (decision dependent) state transition frequencies and rewards.

As an example, let us consider state $L0$. In this state, the possible decisions concern whether component c moves to the location of $d1$ and $d2$ when interacts with them. Then, the set DecVec($L0$) consists of the following four alternative global decision vectors:

- decision 1: [c does not move to $d1$, c does not move to $d2$]
- decision 2: [c moves to $d1$, c does not move to $d2$]
- decision 3: [c does not move to $d1$, c moves to $d2$]
- decision 4: [c moves to $d1$, c moves to $d2$].

For each decision, we must calculate the corresponding overall transition frequencies and rewards (note that the cost of an interaction between colocated components is zero). For the sake of conciseness, we only show the results obtained for state $L0$, limited to decision 2:

$$\begin{aligned} \text{freq}(L0 \rightarrow L0) &= \frac{N_i(n,c) + N_i(d2,c)}{N_{\text{tot}}} = \frac{2 + 2N}{4N + 6} \\ \text{rew}(L0 \rightarrow L0) &= \frac{N_i(n,c) \cdot 0 + N_i(d2,c)C_i(d2,c)}{N_i(n,c) + N_i(d2,c)} = \frac{N(r+R)}{2 + 2N} \\ \text{freq}(L0 \rightarrow L1) &= \frac{N_m(d1,c)}{N_{\text{tot}}} = \frac{1}{4N + 6} \\ \text{rew}(L0 \rightarrow L1) &= \frac{N_m(d1,c)C_m(d1,c)}{N_m(d1,c)} = m \\ \text{freq}(L0 \rightarrow L2) &= 0 \\ \text{rew}(L0 \rightarrow L2) &= 0 \end{aligned}$$

As outlined in section 3, the resulting MDP, obtained from the algorithm, defines a family of MRPs, each corresponding to a different policy. For the considered example, figure 11 shows the transition diagrams of two possible MRPs belonging to this family. For simplicity, the figure only shows the possible transitions, without the associated rates and rewards. Figure 11.a shows the transition diagram corresponding to a policy where component *c* changes location only at the beginning of interactions with *m* and *d1*, while it does not change location for interactions with *d2*. On the other hand, the diagram in figure 11.b corresponds to a “full mobility” policy, where *c* always moves to the location of its partner.

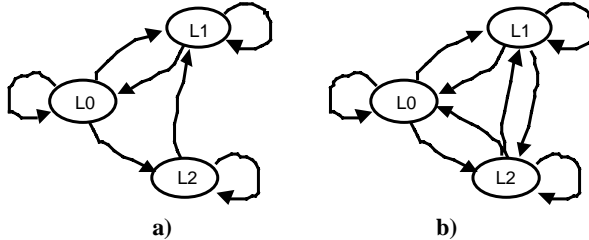


Fig. 11. MRP transition diagrams for two possible policies. a): a “partial mobility” policy; b): “full mobility” policy

Once we have generated the MDP, we can solve it to determine the optimal policy, that is the selection of a decision in each state that optimizes the reward accumulated in the corresponding MRP. In our example, the optimality criterion could correspond to the minimization of the reward accumulation rate, that is the minimization of the network traffic per unit time. Of course, the optimal policy depends on the values given to the system parameters (in our example, the size of the messages and of the possibly mobile component *c*). Different values for these parameters model different scenarios. If, for example, the optimal policy for a given set a parameter values corresponds to the diagram of figure 11.b, this suggest that the optimal design decision for that scenario is the definition of *c* as a mobile agent that visits all the locations of its partners.

5 Related Work

In this section we present a short survey of the existing work on performance modeling generation starting from UML based software models by identifying two main areas: “static”architectures, where no kind of mobility is present, and “mobile”architectures.

In the former area, several papers have presented methodologies for the generation of performance evaluation models starting from UML diagrams, possibly augmented with opportune performance annotations [6, 8, 9, 11, 12, 13, 14, 24]. These papers consider different target models, spanning simulation models, Petri nets and queueing networks. Recently, the growing interest in software architectures has brought to extending performance model generation to also encompass the software architecture concept, with particular emphasis given to the impact on performance of the software organization into components and patterns of interaction [2, 15, 21, 24]. In all these papers the target performance model is a queueing network.

On the other hand, in the latter area of mobile architectures, no methodology has been presented for the generation of performance models starting from suitable UML diagrams, to the best of our knowledge. Some papers have been presented, that analyze different forms of code mobility in “isolation” [16, 22], independently of their utilization within particular applications, thus providing some form of general guidelines that can help in taking decisions during the design of an application. Other papers consider particular applications that exploit code mobility [1, 10, 19, 20], and evaluate their performance using “ad hoc” models. [5] presents the definition of a general methodology that can be used as a support for the designer during the early phases of the development of a specific software application, providing insights about the consequences of architectural choices that can be taken during those phases. However, this paper uses an *ad hoc* designed formal language for the software description.

6 Conclusions

With respect to the existing literature, the main contributions of this paper can be synthesized as follows. We have suggested an extension of UML to model the possible adoption of mobility based paradigms in software architectures. The extension allows a clear distinction between the interaction style and the actual interactions among components. Moreover, we have provided a methodology that, starting from a set of (extended) UML diagrams, derives a performance model that allows the designer, early in the design phase, to evaluate the convenience of introducing logical mobility into a software application. Finally, the definition of algorithms within the methodology is a step ahead towards automatic generation of software performance models.

Future work includes the extension of the methodology to include a wider range of Sequence Diagrams, a refinement of mobility modeling, to model more precisely different mobile code paradigms, and the inclusion of physical device mobility. Different types of performance models (e.g. queueing networks) should also be considered.

Acknowledgements. Work partially supported by MURST project „SALADIN: Software architectures and languages to coordinate distributed mobile components“.

References

1. M. Baldi, G.P. Picco “Evaluating the tradeoffs of mobile code design paradigms in network management applications” in Proc. *20th Int. Conf. on Software Engineering* (ICSE 98), (R. Kemmerer and K. Futatsugi eds.), Kyoto, Japan, Apr. 1998.
2. S. Balsamo, P. Inverardi, C. Mangano “An Approach to Performance Evaluation of Software Architectures”, Proc. of *First Int. Workshop on Software and Performance* (WOSP 1998), September 1998, SantaFe, USA, 1998.
3. L. Bass, P. Clements, R. Kazman, *Software Architectures in Practice*, Addison-Wesley, New York, NY, 1998.
4. G. Booch, J. Rumbaugh, I. Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley, 1999.
5. V. Cortellessa, V. Grassi “Performance Evaluation of Mobility-based Software Architectures” Proc. of the *2nd Int. Workshop on Software and Performance*, WOSP2000, Ottawa, Canada, Sept. 2000
6. V. Cortellessa, G. Iazeolla., R. Mirandola “Early Performance Verification for Object-Oriented Software Systems”, *IEE Proceedings on Software*, July 2000. .
7. A. Fuggetta, G.P. Picco, G. Vigna “Understanding code mobility” *IEEE Trans. on Software Engineering*, vol. 24, no. 5, May 1998, pp. 342-361.
8. Kahkipuro P. “UML based Performance Modeling Framework for Object-Oriented Distributed Systems”, Proc. of *2nd Int. Conf. on the Unified Modeling Language*, October 28-30, 1999, USA, LNCS, Springer Verlag, vol.1723, 1999.
9. King, P. and Pooley, R. “Using UML to Derive Stochastic Petri Net Models”, Proc. of the *15th UK Performance Engineering Workshop*, Dept. of Computer Science, The University of Bristol, N. Davies and J. Bradley editors, UKPEW '99 July 1999.
10. D. Kotz, G. Jiang, R. Gray, G. Cybenko, R.A. Peterson “Performance analysis of mobile agents for filtering data streams on wireless networks ” in *3rd Int. Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*(MSWiM 2000), Aug. 2000.
11. Merseguer, J., Campos, J. and Mena E. “Performance Evaluation for the design of Agent-based Systems: A Petri Net Approach”, Proc. of *Software Engineering and Petri Nets* (SEPN 2000), June 2000, Aarhus, Denmark, 2000.
12. Mirandola R. and Cortellessa, V. “UML based Performance Modeling of Distributed Systems” Proc. of *3rd International Conference on the Unified Modeling Language*, October 2-6, 2000, York, UK, LNCS, Springer Verlag, 2000.
13. Petriu, D. Shousha, C., Jalnapurkar, A. “Architecture based Performance Analysis Applied to a Telecommunication System”, *IEEE Trans. on Software Engineering*, Nov. 2000.
14. Pooley, R. and King, P. “The Unified Modeling Language and Performance Engineering”, *IEE Proceedings Software*, vol.146, no.1, pp. 2-10, February 1999.
15. Williams, L.G. and Smith, C.U. “Performance Evaluation of Software Architecture”, Proc. of *1st Int. Workshop on Software and Performance*, WOSP1998, Sept. 1998, SantaFe, USA.
16. A. Puliafito, S. Riccobene, M. Scarpa “An analytical comparison of the client-server, remote evaluation and mobile agent paradigms” in *1st Int. Symp. on Agent Systems and Applications and 3rd Int. Symp. on Mobile Agents* (ASA/MA 99), Oct. 1999.
17. M.L. Puterman, *Markov Decision Processes*, J. Wiley & Sons, 1994.
18. G.C. Roman, G.P. Picco, A.L. Murphy “Software engineering for mobility: a roadmap” in *The future of SW engineering* (A. Finkelstein ed.), ACM Press, 2000, pp. 241-258.

19. G. Samaras, M.D. Dikaiakos, C. Spyrou, A. Liverdos “Mobile agent platforms for Web databases: a qualitative and quantitative assessment” in *1st Int. Symp. on Agent Systems and Applications and 3rd Int. Symp. on Mobile Agents (ASA/MA 99)*, Oct. 1999.
20. T. Spalink, J.H. Hartman, G. Gibson “The effects of a mobile agent on file service ” in *1st Int. Symp. on Agent Systems and Applications and 3rd Int. Symp. on Mobile Agents(ASA/MA 99)*, Oct. 1999.
21. B. Spitznagel, D. Garlan “Architecture-based performance analysis” in Proc. 1998 Conference on Software Engineering and Knowledge Engineering, June 1998.
22. M. Strasser, M. Schwehm “A performance model for mobile agent system” in *Int. Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 97)*, vol. II, (H.R. Arabnia ed.), Las Vegas 1997, pp. 1132-1140.
23. S.M.Yacoub, B. Cukic, H.H. Ammar “Scenario-based reliability analysis of component-based software” in Proc. of *ISSRE’99*, 1-4 Nov. 1999, Boca Raton, Florida.
24. *WOSP2000*, Proc. of the *2nd Int. Workshop on Software and Performance*, Ottawa, Canada, Sept. 2000

Appendix

In this appendix we present more formally the MDP generation algorithm, outlined in section 4.3.

For each state $s \in S$, let us define:

$F(s) = \{[succ_j, f_j, c_j]\} (j = 1, \dots, |F(s)|)$ a sequence of 3-tuples, where $succ_j \in S$ is a destination state, f_j is a transition rate, and c_j is a transition cost;

$$D(s) = \left\{ \left[[d_{j1}, d_{j2}], [succ_{j1}, succ_{j2}], [f_{j1}, f_{j2}], [c_{j1}, c_{j2}] \right] \right\} (j = 1, \dots, |D(s)|)$$

a sequence of 4-tuples, where each element of the 4-tuple consists of a pair; in particular d_{j1} expresses the selection of the no-mobility option as interaction style between two components (source and target of a `moveTo?` message) and d_{j2} expresses the selection of the alternative mobility option for the same pair of components; then, $succ_{jr} \in S$, f_{jr} , and c_{jr} ($r = 1, 2$) are the corresponding destination state, transition rate, and transition cost.

Input:

- the state set S (see section 4.2);
- the set $P = \{[Ch, Ck]\}$ of pairs of interacting components, obtained from CD (each $[Ch, Ck]$ is a pair of components in CD that are source and target, respectively, of a message);
 - for each pair in P , the number N_i of normal interactions, obtained from SD ;
 - for each pair in P , where the two components are the source and target, respectively, of a `moveTo` or `moveTo?` message, the number N_m of possible location changes, obtained from SD and CD (equal to number of consecutive interactions sequences between the pair of components that can be isolated in SD);
 - for each pair in P , the average cost C_i of a normal interaction, obtained from annotations in SD (equal to the sum of all the costs that characterize the interactions between the pair of components, divided by the corresponding N_i);
 - for each pair in P , where the two components are the source and target, respectively, of a `moveTo` or `moveTo?` message, the average cost C_m of a location change for the source component;

- the total number of interactions $N_{tot} = \sum_{\substack{\text{all the pairs} \\ \text{in } P}} (N_i + N_m).$

Output:

for each state $s \in S$, the set of alternative decisions concerning components mobility in that state, and the associated transition rates and transition rewards.

Algorithm:

for each state $s \in S$:

$F(s) := ?$;

$D(s) := ?$;

for each pair $[Ch, Ck] \in P$:

$F(s) := F(s) \cup \{3\text{-tuples generated by rules } A, \text{ and } B\}$;

if $(Ch$ and Ck are colocated in state s)

then $F(s) := F(s) \cup \{3\text{-tuple generated by rule } C\}$

else $D(s) := D(s) \cup \{4\text{-tuples generated by rule } C\}$

/ the application of rules A, B, and C requires the use of the N_i , N_m , C_i , C_m , and N_{tot} quantities provided as input */*

endfor;

build the set $DecVec(s) := \prod_{j=1}^{|D(s)|} \{dj1, dj2\}$;

/ DecVec(s) is the set of all decision vectors concerning components mobility in state s; the cardinality of DecVec(s) is $|D(s)|^2$; each element of DecVec(s) is a vector V with $|D(s)|$ components, where the j-th component $V[j]$ takes as value one of the two decisions $\{dj1, dj2\}$,*

for each $V \in DecVec(s)$:

for each state $s' \in S$:

$$freq(s \rightarrow s') := \sum_{\substack{j=1 \\ \text{s.t. } succ_j = s'}}^{|F(s)|} f_j + \sum_{\substack{j=1 \\ \text{s.t. } V[j]=d_{jr} \\ \text{and } succ_{jr} = s'}}^{|D(s)|} f_{jr}$$

$$rew(s \rightarrow s') := \frac{\sum_{\substack{j=1 \\ \text{s.t. } succ_j = s'}}^{|F(s)|} c_j \cdot f_j + \sum_{\substack{j=1 \\ \text{s.t. } V[j]=d_{jr} \\ \text{and } succ_{jr} = s'}}^{|D(s)|} c_{jr} \cdot f_{jr}}{freq(s \rightarrow s')}$$

endfor
endfor

endfor
 $\{dj2\}$ */