Military Technical College
Kobry Elkobbah,
Cairo, Egypt

5th International Conference
on Electrical Engineering
ICEENG 2006

Mobility Modeling Techniques

Mahmoud Abdalla M., Ismail Abd-Elghafar, Mohamed G. and Mahmoud* Hassan

ABSTRACT

Performance is critical to the success of any software system, especially large and real time ones. Performance for such systems should be predicted as early as in the requirements analysis and design phases of the development process and before code implementation. This is essential to save the investments of money and time. Several S*oftware Performance Engineering* (SPE) approaches have been proposed to predict and validate the performance of software systems from its architectural designs. Some of them have been applied successfully to static (non-mobile) systems. Performance modeling and analysis of mobile systems is more complex than non-mobile ones. Although mobile systems are gaining more and more widespread and importance, the means for their specification are still underdeveloped. The obstacle that faces extending static software performance prediction approaches to be applied to mobile systems is to find a way to model the mobility behavior of software components.

In this paper, we are concerned with two performance prediction approaches that have been provided with mobility modeling techniques. This enables us to use them for performance validation of mobile systems. The paper's main focus is on presenting two mobility modeling techniques that were proposed for these two approaches in detail. Our objective is to study, analyze and compare them. The framework of each approach is also presented to see how both the technique and the approach fit together.

1. INTRODUCTION

Wireless networks are becoming everywhere; these networks provide mobile users with all types of data communications while being in move and from anywhere. Performance is critical to the success of any software system, especially large and real time ones. For mobile software systems, performance is a necessity for reasons as follows:

First, mobile users have different needs from PC and Web users, for example, mobile users are more likely to be fulfilling an immediate need, not leisurely browsing for information. In addition, mobile users are subjected to environmental distraction, not sitting in a quiet room. Mobile applications should be of a higher standard of usability than stationary applications. They must match the standards of successful consumer products: intuitive user interface, instant relevance and fast response time.

Second, mobile wireless applications exhibit a high degree of complexity and need a lot of investments, the matter that imposes that their performance should be validated early and during the architectural design phase of the development process.

In the literature, several approaches have been proposed to integrate the process of performance modeling and analysis of non-mobile systems in the architectural design stage [3], some of these approaches were applied successfully [9], [6]. Few of them have been extended to address the problem of performance validation of mobile systems.

The main obstacle that faces the process of performance validation of mobile systems is how to model mobility behavior of software components. The means for mobility specification is still underdeveloped, there is no widely accepted standard way for expressing mobility in the architectural specifications of UML [1] and the area of mobility modeling is an open one [5].

We can define mobile software architectures with various mobility styles. *Physical mobility* refers to the style of moving of portable computing devices and their application. *Logical mobility* refers to the style of movement of the software only leaving the device on which it resides. *Logical mobility* can be distinguished into two types: Mobile components can migrate to new location while preserving their identities (mobile agents), another type for mobility is creation of copies of software components at the location of the component that starts the interaction (code on demand), or at the location of the component that accept the interaction (remote evaluation).

The problem of devising a technique that precisely models all these aspects of mobility is seriously difficult. Few techniques have been proposed for software mobility modeling, some of them restricted themselves to only one style of mobility, others claims that both mobility styles are handled. Several mobility modeling techniques were proposed within the context of complete performance prediction approaches which were previously applied to static systems, while other techniques were presented in isolation of any approach. Our interest is in the former ones.

The focus of the paper is on mobility modeling techniques, so we present here two of them in detail. Performance prediction approaches including these techniques are also introduced to show how they fit together. The paper's objective is to study, compare and evaluate two mobility modeling techniques.

This paper is organized as follows. In section 2 we introduce a mobility modeling technique proposed by S. Balsamo and M. Marzolla, and the associated performance prediction approach. In section 3 we present another technique proposed by V. Grassi and R. Mirandola along with the approach employing that technique. Section 4 is an example for solution of QN model using WinPEPSY tool. Section 5 is analysis and comparison between the two techniques. Section 6 is our conclusion.

2. Mobility modeling Technique No.1

We here show the technique proposed in [5] to model the mobility of software entities. The technique depends on two factors:

- Using the UML diagrams in a devised and intelligent way, specially the Use Case and Activity diagrams to model mobility. For example, unlike the traditional way that employs Use Cases to model software functionalities, the technique uses them to model mobility behaviors.
- Using standard UML SPT profile for performance annotations [4], [7].

The technique proposes a set of assumptions as follows:

- Mobile entity may be physical device moves in the real space or a software component that migrates from one device to another.
- The mobile system is perceived as a collection of devices (processors and communication links)
- Computations on the system are seen as a set of activities executing on devices.
- A configuration is a system state in which a set of activities (computations) are assigned to certain devices.
- When a mobile entity travels through the system, it activates a sequence of configurations and for each entity location, there is a specific configuration.
- Once a configuration is activated, the mobile entity starts interaction with the system, while interacting, it can not move (i.e., system configuration can not change). A further movement for the entity is possible only when an interaction is completed.

2.1 Steps of the technique

The proposed technique goes through three basic steps to model mobility behavior.

***Step1*:** this step models both of the mobile entities and the possible mobility behaviors of each entity. This is done using Use Case diagram. Actors model mobile entities, while use cases model possible mobility behaviors. By a mobility behavior we mean a certain path in which the mobile entity may move. For example, an entity moves from location of LAN1 to that of LAN2 then to LAN3. In each of the three locations, there will be a certain configuration for the system. For example, C1 is the configuration of the system when the mobile entity is in LAN1 and likewise is C2 and C3. In each configuration, there will be a corresponding specific scenario for the interactions of the mobile entity with the system, for example, scenario1 then scenario2 then scenario3.

***Step2*:** This step is a high level modeling for each mobility behavior. Each possible moving path taken by the mobile entity (which was represented by a Use Case in step1) is expanded to an Activity Diagram; each of its nodes represents the whole execution scenario performed by the mobile entity for certain configuration. Activity diagrams of this step are referred to as high-level Activity Diagrams.

***Step3*:** this describe the interactions of each scenario, each node in the high-level Activity Diagram is expanded into another Activity Diagram. These diagrams are called low-level Activity Diagrams. While step1, 2 model mobility of the mobile component, this step models the execution behavior (dynamics) of the component.

2.2 Example

The following example illustrates the technique. A mobile user with a Laptop is running an application for retrieving video from the server connected to LAN3 in wires. LAN1, LAN2 and LAN3 are wireless networks connected to the Internet. We consider that three different configurations can occur due to moving the user as shown in Figure 1.

Figure 1a, shows configuration1, the user is in LAN1 and communicates with the server through the path LAN1-Internet-LAN3.

Figure 1b, shows configuration2, the user is in LAN2 and communicate through the path LAN2-Internet-LAN3.

Figure 1c, shows configuration3, the user is in LAN3 and communicates with the server directly through LAN3.
Moving the laptop in such a way represents just one possible mobility behavior (LAN!-LAN2-LAN3). This behavior is modeled by a Use Case as in figure 3.

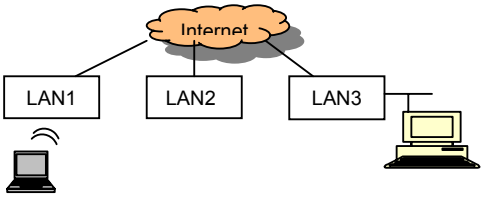


Figure 1a (configuration1)

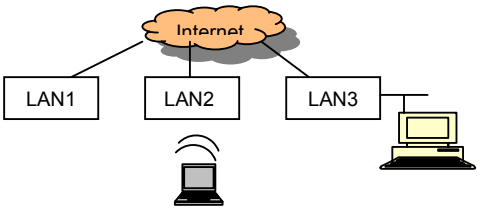


Figure 1b (configuration2)

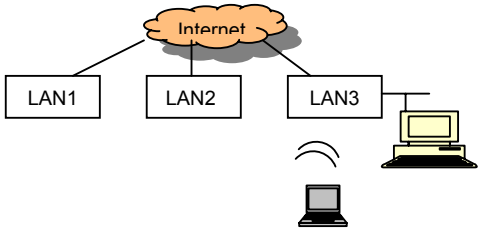


Figure 1c (configuration3)

Figure 1 Mobile user with mobility behavior (B1)

Figure 2 is the Deployment Diagram model "DD" of the physical structure of the system (processors and communication links and interconnections). Each node represents a resource. Performance annotations should be entered on the diagram; however, we omit them in order not to clutter the diagram. It is worth to note that, DD is not involved in mobility modeling process; however it is needed by the simulator to build the simulation model.

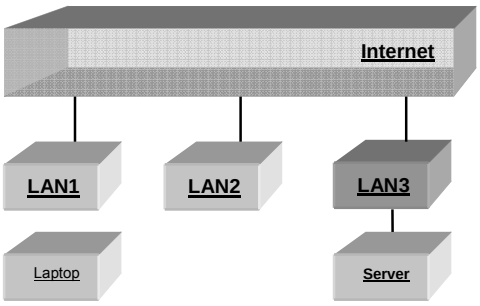


Figure 2 Deployment Diagram of the Mobile System

Figure 3 shows a mobile user with two possible mobility behaviors (B1, B2). In B1, he moves from LAN1 to LAN2 then LAN3. In B2, he enters LAN2 then moves to LAN1. The probability of occuring each possibile behavior (p1, p2), as well as the other performance annotations should be entered on the diagram. An actor models a class of mobile entities which represents a specific workload type for performance model. An actor should be defined by a stereotype "ClosedWorkload" or "OpenWorkload", also by tagged values "PApopulation" or "Arrivalrate" respectively. These stereotypes and tagged values are necessary for the

simulator to build performance model and are considered input parameters for it. Each Use Case models one possible mobility path. The probability for each path should be entered.

Figure 4 gives the complete view of the technique, stereotypes and tagged values are not entered in order not to clutter the figure.

Figure 4b models the mobility behavior as each Use Case (path of mobility) is expanded by a complete Activity Diagram (AD). The use case for behavior1 is expanded by a three node AD, each node represent a whole scenario. Scenario1 represents the interactions made due to the execution of the mobile entity when it is located in LAN1, likewise scenario2 and scenario3 when the mobile entity is at LAN2 and LAN3.This AD models the interactions of one mobility path in an abstracted form, so it is called "high-level AD". Situations of non-deterministic interactions (i.e. an interaction that follows by several successors) and concurrency can be modeled here also [12].

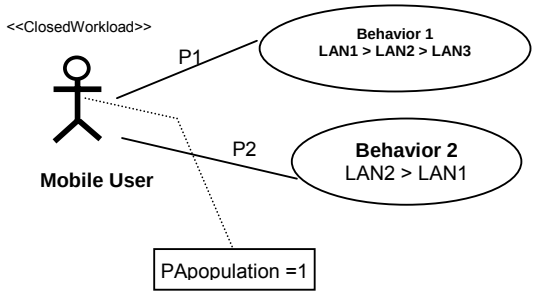


Figure 3 Modeling of Mobility Paths

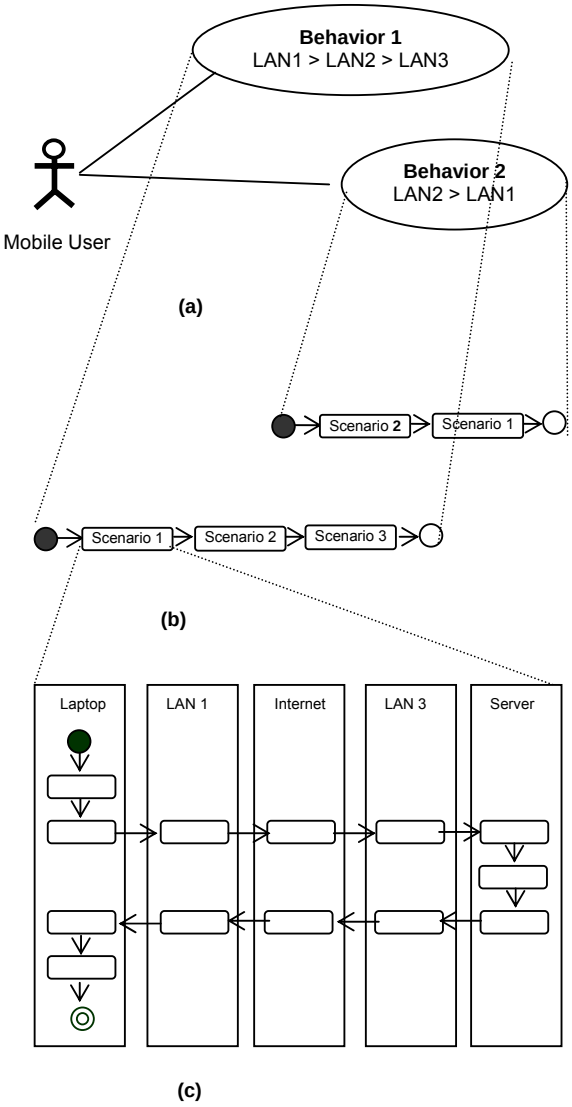


Figure 4 The three steps of Mobility Behavior Modeling Technique

Activity Diagrams provide Fork and Join nodes to represent the case where the component would copy itself then both of them start execution at the same time. Physical mobility is modeled at this step, this is clear from the sequence of different scenarios, there is one scenario for the executions in each location. Figure 4c shows step 3 that models the execution behavior of the mobile component at different locations in the system. Each scenario node in the high-level AD is expanded by a complete AD showing the interactions made due to the component execution in that scenario. ADs at this step are called low-level ADs. By using swimlanes in this step, it can be shown the association between the interactions and their locations. Due to that code mobility appears at this step as each interacting component is shown associated with the location of its execution. All the above diagrams along with the performance annotations are entered into the graphical interface of ArgoUML tool that translates them into XMI text. The simulator is a prototype

tool specially designed to read this data and build the performance simulation model in the form of a simulation program. Our focus is on mobility modeling, so, we did not go through the details of building the simulation model. For such details refer to [5], [2], [3].

2.4 Simulation model-Based Performance Prediction Approach

The performance simulation model-based approach proposed in [2], [3] to integrate performance modeling and validation of traditional non-mobile software systems into the development process was extended in [5] to be applied to mobile systems. This approach uses the previous technique to model mobility in the architectural designs diagrams.

The approach uses UML software specification models (Use Case, Activity and Deployment Diagrams) annotated with tagged values and stereotypes for performance to drive a performance model automatically by using a prototype tool, UML-Ψ (UML performance simulator). The technique has the advantage of using the annotations of the standard UML profile for Schedulability, Performance and Time (SPT) [4].

ArgoUML CASE Tool, freely available on the Internet is used to develop the software design diagrams using its graphical interface. Performance annotations are added to the diagrams as stereotypes and tagged values drawn from the profile. ArgoUML tool translates these diagrams along with the annotations to XMI text. The XMI representation of the diagrams is then converted to a process oriented simulation model by the prototype tool (UML-Ψ) which executes an algorithm [2] for that purpose. The simulator implements the model as a simulation program which is executed to evaluate the performance of the system. Simulation results are reported back to the original UML diagrams as tagged values, so they are available to designer and integrated in the UML specifications.

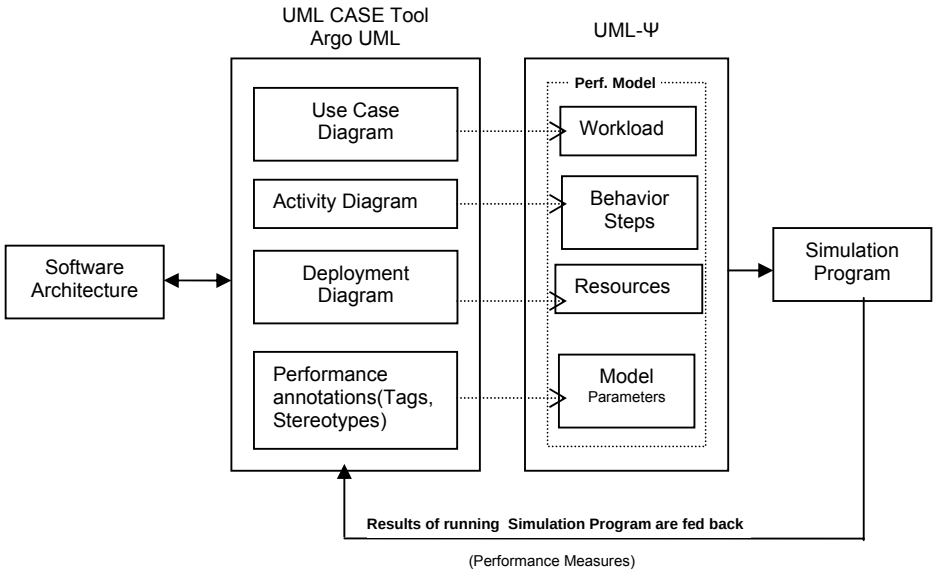


Figure 5 Simulation-Based Performance Prediction Approach

The information required for UML-Ψ to build the performance model is taken from the UML diagrams and the added annotations as shown in figure 5. Use Case diagram will provide information about workload; Activity diagram provides information about software behavior, while the deployment diagram feeds the information about the hardware resources which

execute the software. Tagged values will provide for the model parameters. For more detail see [5], [2], [3].

3. Mobility Modeling Technique No.2

This technique was proposed within a framework of an approach called "PRIMAmob-UML" [13] for performance analysis and prediction of mobile software architectures. It is based on using both the Collaboration Diagram (CD) and the Sequence diagram (SD) to model code mobility. This means that it is a UML based technique.

The SD describes only the sequence of exchanged messages between system software components (interaction logic) without showing the style (mobile or static) in which these interactions take place. SD can also be used to model execution interactions within a component. SD is drawn using the standard UML notations without any modification. We use timed SD where approximate occurrence time of every interaction is annotated on the leftmost vertical axis.

The CD shows only the components that are interacting and the style they are using to do that. CD does not show any information about the sequence of interactions, and it is drawn using standard UML notations in addition to two new devised stereotypes that are used to label the interactions modeled in the CD. These two stereotypes are "moveTo" and "moveTo?".If a stereotype "moveTo" labels an interaction in the CD, it means that the source component will move to the location of its target before starting a sequence of consecutive interactions with it. This style of interaction will be applied to each sequence of interactions shown in the associated SD, between the source and the target component of the "moveTo" message.

There is no guarantee that moving a component from one location to another (mobile style) before interacting will realize better performance than the case when it interacts with it from its location (static style). At architecture design stage, the designer is not certain about which style will give better performance, so, it was decided to model this uncertainty. Modeling of the uncertainty (possible using any of the two styles) enables including both the two styles in the generated performance model, hence, we can choose early the style for our design that gives better performance. The second stereotype is "moveTo?" It is proposed for modeling uncertainty. When a message between two components in the CD is labeled with"moveTo?", this means that the source component "may" move to the location of its target before starting a sequence of interactions with it.

As a result, in the CD drawn in this technique, some interactions are not labeled and these refer to ordinary interactions without any mobility. Some are labeled with "moveTo", and are referring to component mobility before interaction. The interactions labeled with "moveTo?" means that the designer is not sure about which style to choose for better performance. It is worth to note that both interaction logic and mobility style are modeled as separate concerns. This means that both aspects will not be modeled in the same diagram. Each diagram models one concern only.

3.1 Example:

Consider a travel agent manager software system consists of four components (m, t, a1, a2). The component "m" which is an instance of class "TRAVM" periodically collects information from different nodes. It then bid on the collected resources in order to

automatically deliver a lower cost for the travel tickets. "m" starts a component "t" which contacts different nodes by exchanging a number of messages (1..n) with each node to collect the required information. The example shows two remote nodes only at two different locations. The software component "t" can perform communication in two styles, either stays at its location and communicates remotely with the two locations of "a1:AUCT" and "a2:AUCT" (static), or migrates to each of them and communication is made locally at their locations (mobile interaction style). As shown, at each of the two locations, "t" communicates with an instance of class "Auctioneer" (a1,a2) which deliver a bid for the itinerary requested by the travel manager component.

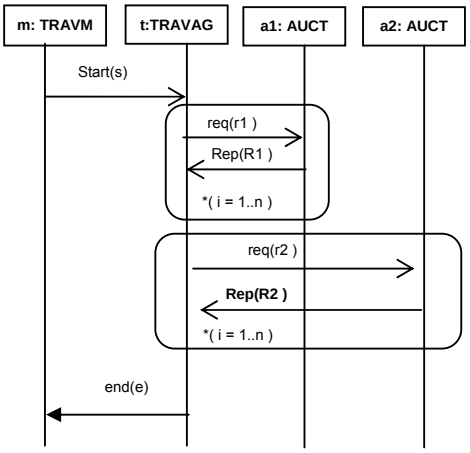


Figure 6 modeling the sequence of interactions of the system using SD

Figure 6 shows SD that models the sequence of the interactions (interaction logic) among the software components. SD does not give any information about their locations or about the communication style (mobile or static components).

Figure 7 shows CD of the system, it shows the software components along with their locations using the tagged value "Location", but it does not indicate the interactions' sequence that was shown clearly in the SD. According to this technique, as there is no any notation marking the interaction messages, we directly know that the components have a static communication style and are bound to their given locations. Components "m" and "t" are at the same location (L0), while "t" interacts remotely with "a1" and "a2" at L1 and L2 respectively.

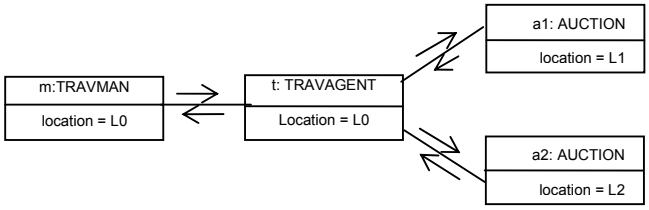


Figure 7 Using CD to model interaction style Without mobility (static system)

Figure 8 shows how the technique uses CD to model component mobility by the innovated stereotype "moveTo". The outgoing interaction arrows from Component "t" shows that it is the only one that can move to the locations L1, L2, L0 according to the semantic of "moveTo" to interact locally at these locations. The location of "t" is left unspecified since it can dynamically change. Initial location of "t" can be given in a separate CD diagram.

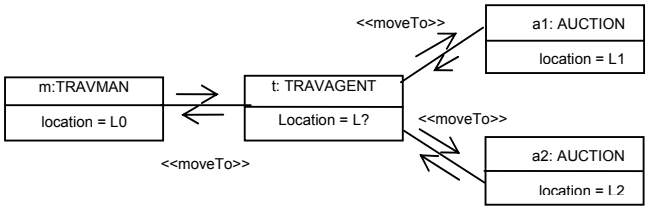


Figure 8 Using CD to model interaction style With mobility (mobile system)

Figure 9 CD models the uncertainty about the interaction style of "t" with a1 and a2.

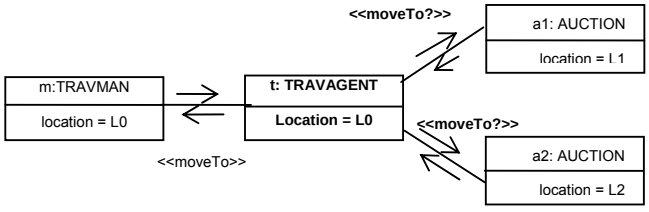


Figure 9 Using CD to model uncertainty about which style to choose, static or mobile

The "moveTo?" that annotates the outgoing arrows from "t" to both a1 and a2 means that it may move to any of them. It is the case the designer is uncertain about the effectiveness of changing the location of "t" when it interacts with a1 and a2. The "moveTo" shows that "t" moves to location of "m" (if not already there) before interacting with it.

3.2 PRIMAmob-UML Performance Prediction Approach for Mobile Systems

After we have presented our 2nd technique to model mobility, we here introduce the approach within which the technique was presented. This would point out how both of them fit together and gives a clear understanding about the implementation of the technique. PRIMAmob-UML approach was derived from previous approach called PeRformance IncreMintal vAlidation in UML (PRIMA-UML) [12], so, comes PRIMAmob-UML approach [13] as an extension with the capability to validate performance for mobile systems. It is worth to note that PRIMA-UML is derived from the first SPE approach of C. U. Smith [10],[11]. The steps of PRIMAmob-UML are the same as that of PRIMA-UML with some added ones, which are needed for mobility modeling. We will designate these added steps by "-mob".

3.2.1 Steps of the approach

- *Step 1* Annotate the Use Case Diagram
- *Step 2* For each Use Case identify and annotate the Sequence Diagrams corresponding to the key scenarios.
- *Step 2-mob* for each Use Case identifies and annotates the corresponding Collaboration Diagram.
- *Step 3* process all the annotated SDs to obtain a meta-EG.
- *Step 3-mob* Process all the annotated CDs and the meta-EG to obtain the mob?-meta-EG.
- *Step 4* Annotate the Deployment Diagram (DD) and tailor the mob?-meta-EG to the annotated DD to generate the mob?-EG-instance.
- *Step 5* Drive an EQNM from the annotated DD.
- *Step 6* Assign numerical parameters to the mob?-EG-instance.
- *Step 6-mob* Perform stand alone analysis on the mob?-EG-instance.
- *Step 7-mob* Merge the mob?-EG-instance and the EQNM into the performance model called mob?-EQNM.
- *Step 8* Solve the performance model.

3.2.2 Example:

To simplify the matters, we will use the same example of travel management software system whose components are shown in Figure 6 .The system's functionality is represented by a single Use Case, see Figure 10. The steps stated above will be applied to the example.

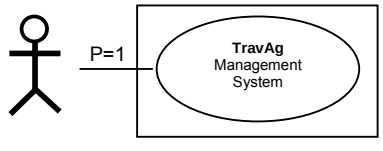


Figure 10 (Step 1) TravAG Software Management System

Step 1: annotate the Use Case Diagram (UCD) of the system. Each Use Case should be annotated by the probability of being in use, as we have only one Use Case, its probability is P=1 as shown in Figure 10. A Use Case is an abstraction of a set of scenarios. Our Use Case is represented by

only one execution scenario (shown in Fig. 6), while there exist three different mobility styles for that scenario(shown in Fig. 7, 8, 9).

Step 2: draw all SDs that represent the possible key scenarios for executing the Use Case and find the probability of each of them. Key scenarios are those affecting performance. In our example the only Use Case is represented by only one scenario for simplicity, so, its probability will be one and it is the key scenario, it is shown in Figure 6.

PRIMA-UML uses timed SDs, i.e. approximate occurrence time of every interaction is annotated on a vertical leftmost axis. Every interaction in the SD (horizontal arrow) is annotated with its name and size of exchanged data. The SD represents a scenario made by the interactions between the system components while they are performing the system task. These components are drawn as rectangles on the top row of the diagram; vertical axis represents time, while horizontal arrows represent the interactions between components.

Step2-mob: this is an added step to PRIMA-UML to model mobility. The CDs corresponding to different mobility styles required to be modeled in our application are drown and annotated with information about component location and mobility using tagged values "Location" and stereotypes "moveTo" and "moveTo?". For our example we will consider the three CDs of Figures 7, 8, 9 because we plan to study the three cases: static, mobile and uncertainty about mobility.

Step 3: we will process the only SD to generate the 1st version of software Execution Graph model (meta-EG) as shown in Figure 11. This is a software model in isolation of any hardware platform and without any mobility modeled into it. PRIMA-UML [12] provides an algorithm to perform this transformation from SD to meta-EG. Each node in the "meta-EG" model is annotated with a tuple in the form of (l(s),A1,A2,t). Each node models an interaction of those modeled in the SD by horizontal arrow, i.e. a set of operations (code block) carried out by the component (A1) that precedes the arrow before interacting with component (A2) that follows it. The interaction is labeled by l(s), where "l" is the interaction name and "s" is the size data sent from A1 (after finishing its operations) to A2 before interacting with it, and "t" is interaction time. Figure 11 shows the translation of SD to meta-EG. For each scenario (SD) we should build a "meta-EG", we have only one "meta-EG" in our example. To explain

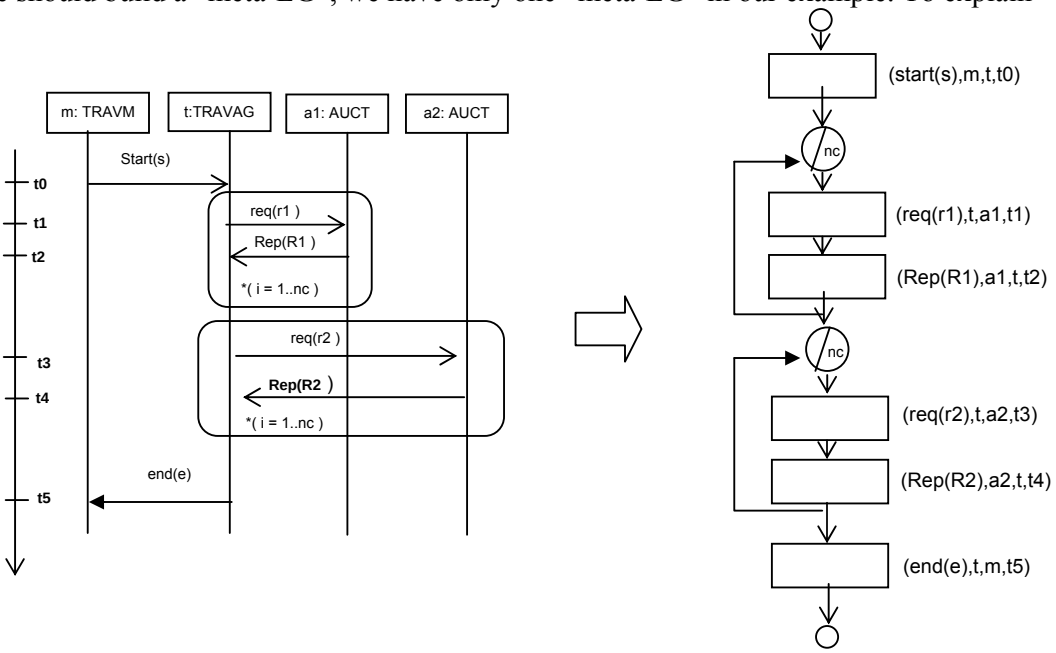


Figure 11 (Step 3) Driving the meta-EG model from SD

more, the tuple (req(r1),t,a1,t1) means that the component "t" will execute a block of code then interacts with component "a1" by sending it a message of size "r1" at time t1 and the interaction name is "req". The meta-EG extracts all the data modeled in the SD.

Step 3-mob: each one of the annotated CDs shown in Figures 7, 8, 9 are processed along with the previously obtained meta-EG to obtain mob?-meta-EG, which is the 2nd version of the system software model as shown in figure 12. The CD includes elements that model mobility. The approach in [13] provides an algorithm for performing this transformation. The obtained "mob?-meta-EG" software model adds two kinds of nodes to the original model (meta-EG), the "mv" node which models the cost of code mobility and "mob?" node that models the uncertainty about mobility. The "mob?" node has two different outcomes, "Yes" and "No", the branch following "Yes" branch models the case of component mobility, while the branch following "No" models the case of static component, i.e. "mob?" is a branching node.

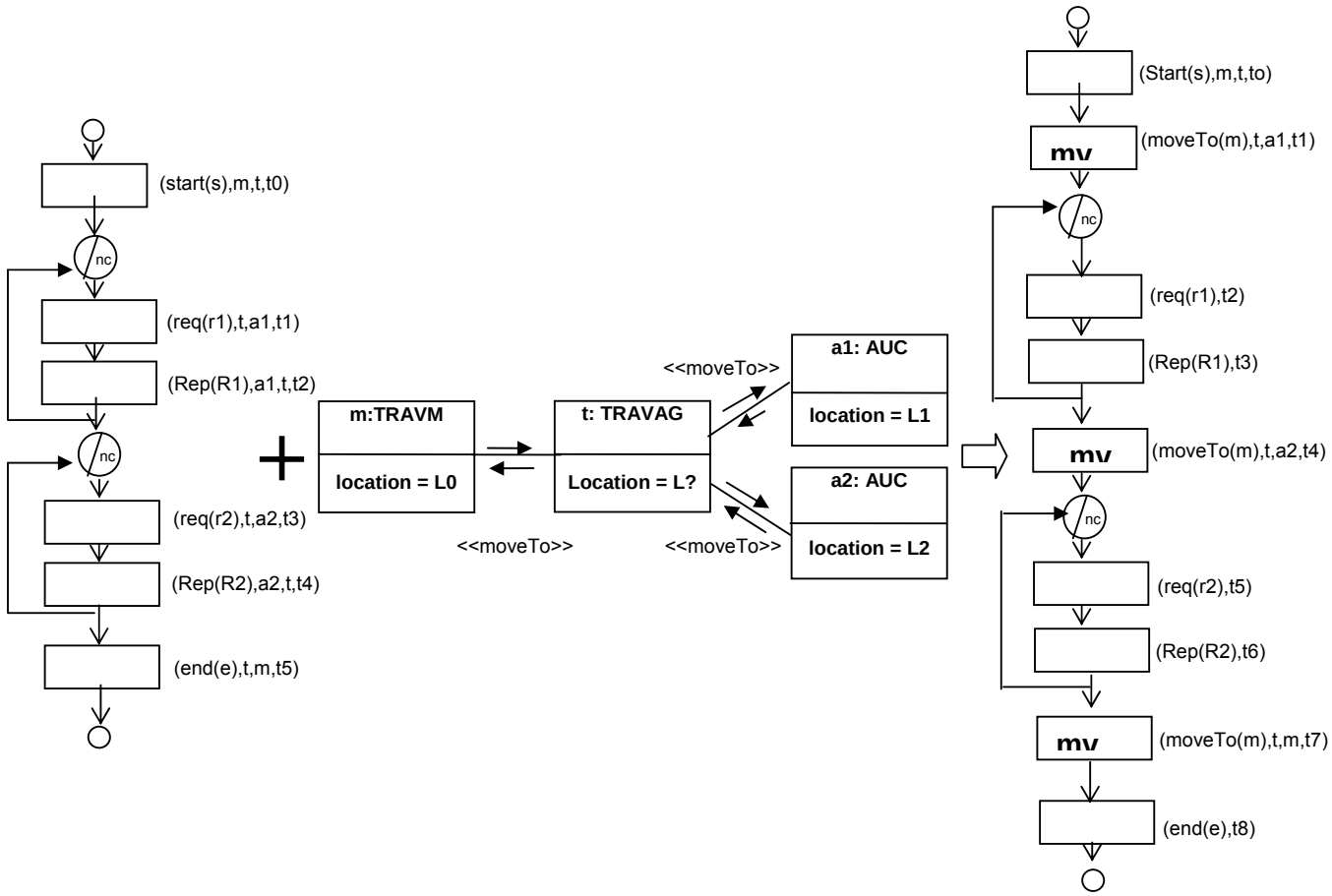


Figure 12 (Step 3-mob) meta-EG model is processed with the CD that models mobility to drive mob?-meta-EG model

Processing the meta-EG with the CD of figure 7 (static style) will produce the same meta-EG without any change and this is not shown here. Figure 12 shows the processing of the meta-EG with the CD of figure 8 where mobile style is adopted. The result is mob?-meta-EG model at the rightmost.

To explain more, consider the top "mv" node labeled (moveTo(m),t,a1,t1), this means that a mobile code block of size m will move from location of "t" to the location of "a1" at time t1. Figure 13 shows the processing of meta-EG with the CD of Figure 9. This gives a complete view including four possibilities for mobility. Path (1) is the same as we got in Figure 12, while path (4) is the case of static interaction style.

The meta-EG on the leftmost of Fig. 12 models the overhead of software execution and components interactions when all the components are at the same location (static style). The "mob?-meta-EG" on the rightmost models both the execution and interaction cost, and additionally, it models the mobility as described by the accompanied CD in the form of an overhead "mv" node. By processing the CD and the meta-EG together, the meta-EG nodes supposed to be executed after component mobility (as shown in the CD) are mapped to "mob?-meta-EG" as local interactions preceded by "mv" node that represent the overhead of component mobility. Node tuple information such as names of interacting components on the nodes of the meta-EG are omitted in mob?-meta-EG as there is no need for them after extracting mobility information into mob?-meta-EG.

To explain more about the translation process of figure 12, take the first node in meta-EG and the corresponding CD, we find that component "m" will execute a code block to initialize an instance of "t" of the class TRAVAG, then "m" interacts with "t" at the same location by sending it a message of size s bits and at time t0. This node is mapped as it is to the right side.
For the 2nd and 3rd node in the meta-EG, it is clear from CD that component "t" will move to location of a1 first, then, both "t" and "a1" will exchange messages in the form of requests and replies locally. The cost of mobility of "t" is modeled by "mv" node with annotation (moveTo(m),t,a1,t1). The "t" and "a1" of the tuple carries information about the channel on which component "t" moves, while m is the code size of "t".

In figure 13 we notice that in path (1) all the gray colored nodes are marked by two entries tuples which means local interactions, while the three "mv" nodes represent the cost of mobility of "t". In path (4) all operational nodes are annotated with four entries tuples, hence they represent remote interactions. For example, the tuple (req(r1),t,a1,t1) means that the node will finish executing its code then sends a message of size r1 is sent from component "t" to component "a1" through the channel between them at time t1. Paths 2, 3 in Figure 13 shows the cases where partial mobility was adopted,

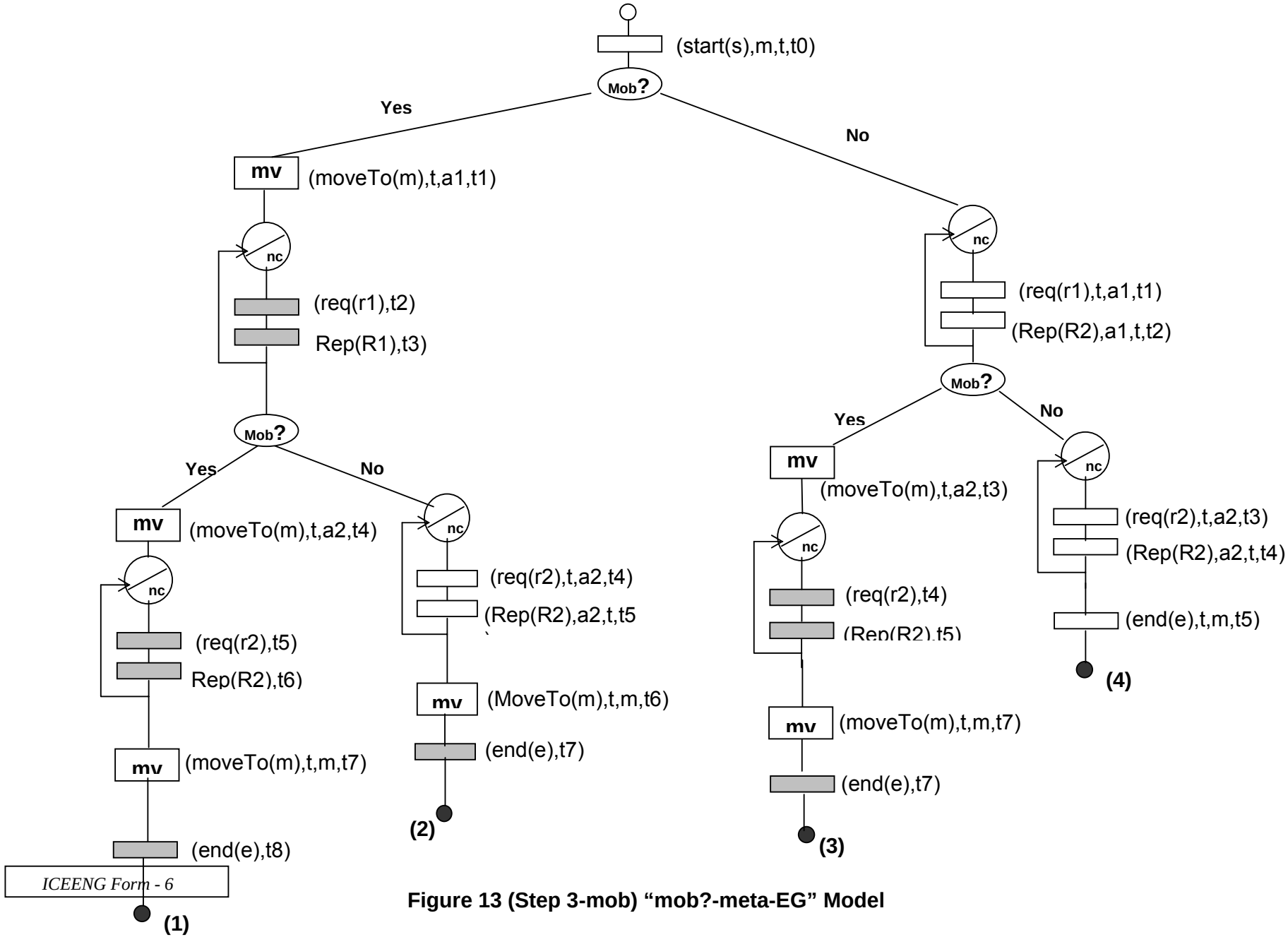


Figure 13 (Step 3-mob) "mob?-meta-EG" Model

Step 4: in this step, the Deployment Diagram (DD) of the system is drawn and annotated. Annotations should include the speed of the connecting links between nodes (in bits/sec). If any additional data about nodes capabilities and configurations is available, it is preferable to put it on the DD. Figure 14 shows the annotated DD diagram of the system.

The annotated DD is used to tailor the "mob?-meta-EG" model obtained in

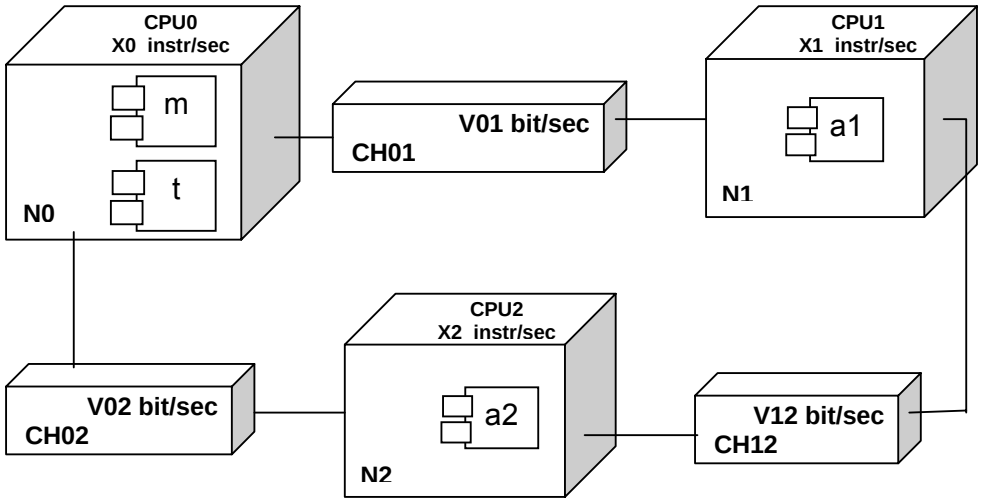


Figure 14 (step 4) annotated Deployment Diagram of the system

figure 13 to its execution environment. This requires to process both the annotated DD (in Fig. 14) and the "mob?-meta-EG" (in Fig. 13) together to get the 3rd version of software model, it is called a "mob?-EG-instance" model. This model includes information about software execution, mobility behavior and hardware environment (such as channels speeds and CPUs speeds). Tailoring an EG to a platform described by an annotated DD consists of two steps:

First, drive the communication cost for each EG node from three data pieces.

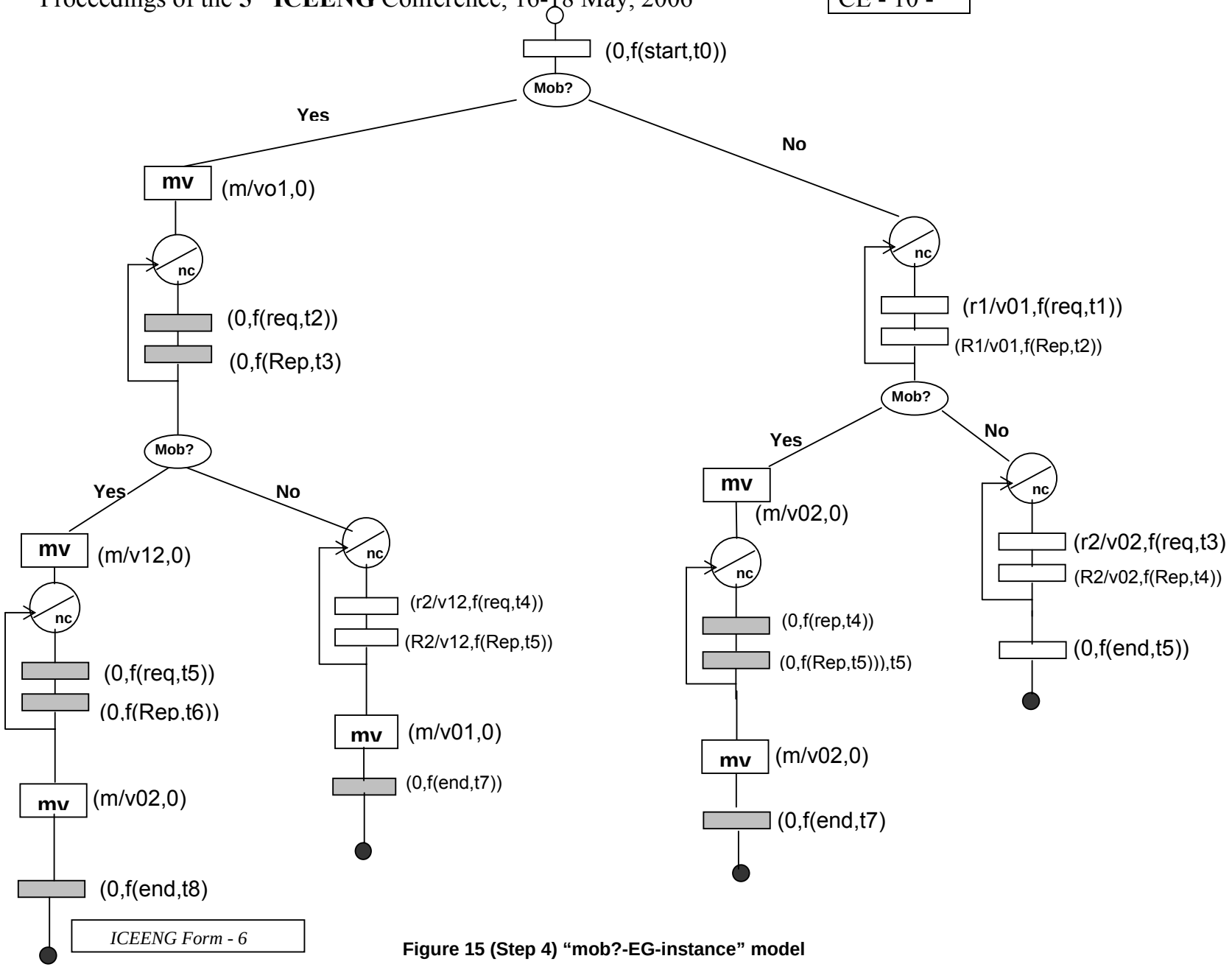
- From EG model we pick the size of interacting data found in each EG node tuple .
- From the DD, we exploit the mapping of software components to the nodes. Links speeds are also obtained.
- From DD, we know whether the interaction is local or remote.

The cost of communication between two components is equal to the ratio, message size/ link speed, and in case of local communication (i.e., both of the two communicating components are on the same node in the DD), the cost of communication is considered zero, as it is too small compared to with remote interaction.

Second, we derive a first estimate for computation cost for each node from:

- Name of the node
- Difference between the node time and the time of the previous node.

We will leave that cost as a function of names and times of nodes. So, step 4 will annotate each node in the mob?-meta-EG graph obtained in step 3-mob by a tuple of two entries only (communication cost, computation cost). This step enables us to try different hardware configurations with the same software system to study hardware alternatives. Figure 15 shows mob?-EG-instance resulted from step 4.



ICEENG Form - 6

Figure 15 (Step 4) "mob?-EG-instance" model

Step 5: The DD drawn in step 4 is used also to drive Extended Queuing Network Model (EQNM) representing the hardware platform that hosts our software system (in addition to driving the "mob?-EG-instance" in step 4). Sufficient information about internal configuration of each node of the hardware platform must be provided to perform this step. This information includes number of CPU's in each node and their speeds, disks…etc. This information allows us to assign numeric parameters to service centers of the EQNM (such as service time). In our example, we consider that each hosting node is equipped with only one CPU and all terminals are connected to node N0. The EQNM we got then is the first hardware model version as it contains information about the hardware environment only. It defines the components of the model and their connections (topology), a further step is needed to parameterize that model and that comes later. Figure 16 shows EQNM model for the case in which the system is completely static as in path 4. Figure 17 shows the case of complete mobile system as in path 1.

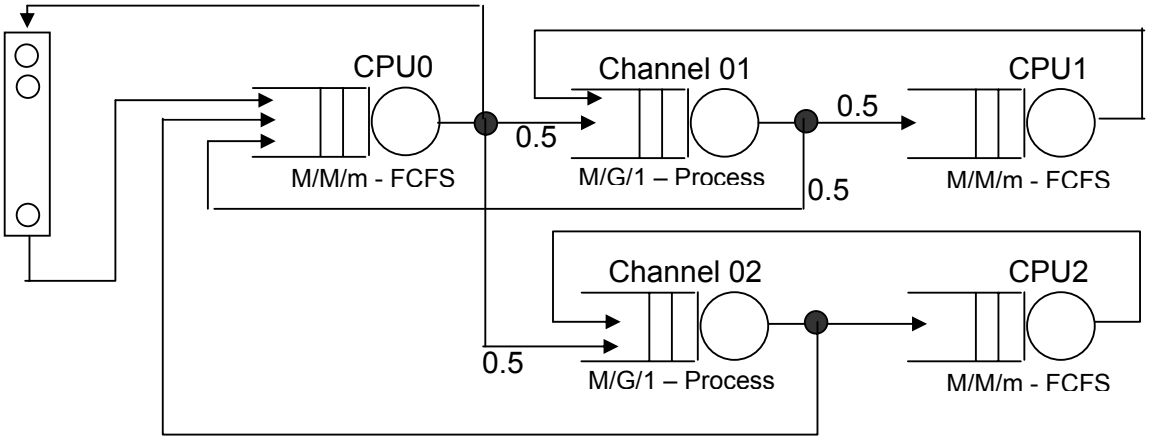


Figure 16 (Step 5) EQNM for complete static style system (path 4)

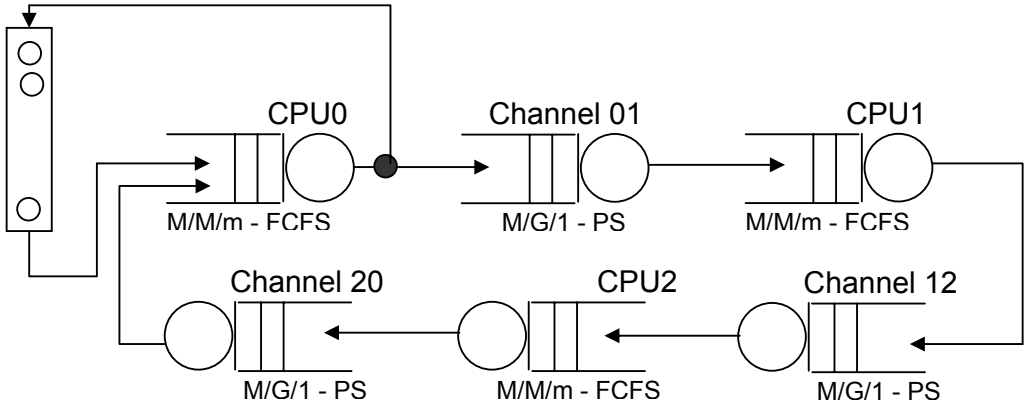


Figure 17 (Step 5) EQNM for complete mobile style system (path 1)

Step 6: in the "mob?-EG-instance", each node is annotated by a two entry tuple, the first entry is the communication cost and it is either zero for local communications or a function of the ratio message size/link speed which is easy to calculate. The second entry is the computation cost of the node and it is expressed as a function of both node name and time as in Figure 15.

In this step, due to lack of information in early design stage, we will assign numerical values as estimates for the computation cost. We first add the estimated probability of executing each path of the "mob?-EG" in figure 15. Second, in order to calculate computational cost, normally a *resource demand vector* [9] should be associated to each graph node.

Figure 18 Resource Demand Vector

devise	Service requested
CPU	3(k instr.)
Disk	2(disk I/Os)

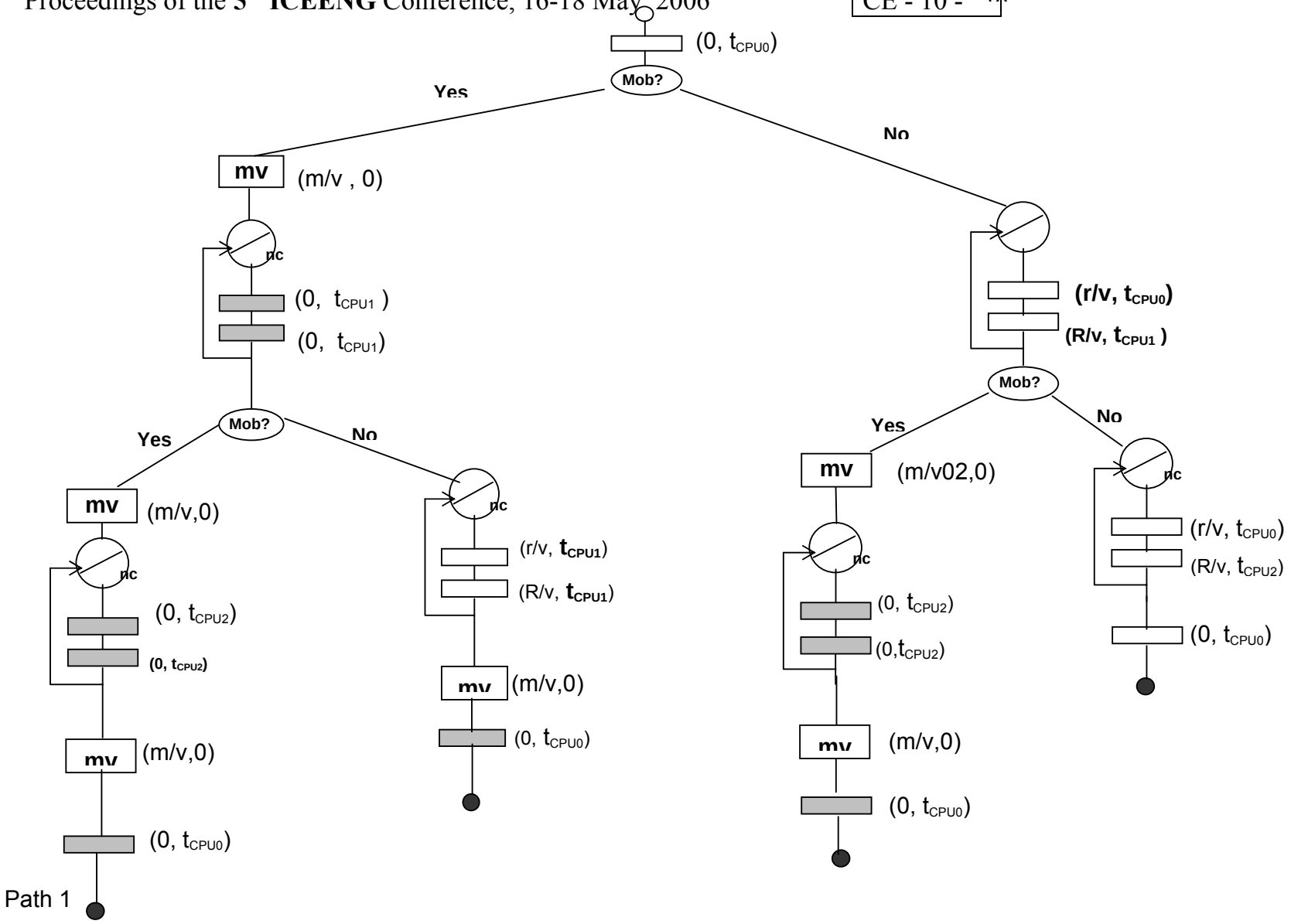
The vector as in figure 18 shows the node devices that contribute in executing the code block of the node, in front of each device; there exist the amount of service requested from it to execute the node code. For example, figure 18 shows that 3 disk I/Os and 3 k CPU instructions are required for the node to be executed.

As it is difficult to provide such detailed information at early design stages, this step depends on the experience of the performance analyst in determining the suitable estimates for the values of these victors. For our example, we will assume that each EG block of code requires one unit of CPU time (t_{CPU}). Figure 19 shows the replacement of computation cost by $t_{CPU.}$

Step 6-mob: to this end, we are able to start our first performance analysis as we have now the 3rd version of the software model in Figure 15. This figure models three aspects of the software system, first is the execution of the components' code and their interactions, different mobility behavior (strategies) are represented by the different paths of the graph and finally, the hardware features important for software execution. Each node is labeled by a tuple of a pair of entries (communication cost, computation cost). This pair may be expressed in explicit numbers, symbolic expressions (as in figure 15), or as upper/lower estimated bounds according to the quality of the information provided to the analyst then. Summing communication and computation costs for the whole nodes of one path will give the total cost of it which the response time of the path, see figure 19. The response times of the four paths can be compared to choose the lowest response time mobility strategy.

Analysis example:

This example points out a type of analysis that can be done on the software model only. We will put these assumptions which are needed due to the lack of information in the early design stages. Consider each request message is of size r bits and each reply message of size R bits. Consider that all the CPUs are identical. All communication links are of the same speed (v). Applying these assumptions to Figure 15, we get Figure 19, specially drawn for this analysis. These assumptions simplify the analysis as follows:



Comm. Cost = 0+ m/v +0+0+ m/v +0+0+ m/v +0 = 3m/v

ICEENG Form - 6

Comp. cost = th+ t_{CPU0} + 2 nt_{CPU2} + 2 nt_{CPU1} + t_{CPU0}

Figure 19 (Step 6) Analysis of software model

- Communication cost of paths 2, 3 is the same.
- Computation cost of all paths is the same and equal to $[th+(2+4n)t_{CPU}]$.

So, our analysis can be restricted to only three paths, 1, 2(3), 4. The analysis will be considered for communication cost only. Summing the cost of nodes of each path as in figure 19, the result be as in the following table:

	Communication cost	Computation cost
Path 1	3m/v	$th+2\ t_{CPU0} +2n\ t_{CPU1} +2n\ t_{CPU2}$
Path 2(3)	2m/v + n(r+R)/v	$th+2\ t_{CPU0} +3n\ t_{CPU1} + n\ t_{CPU2}$
Path 4	2n(r+R)/v	$th+2\ t_{CPU0} +2n\ t_{CPU0} +n\ t_{CPU1} +n\ t_{CPU2}$

We can conclude the following:
Communication cost of path 1 is less than path 4 if, n>= 3m/2(r+R).
Communication cost of path 1 is less than path 2(3) if, n>= m/(r+R).
Communication cost of path 2(3) is less than path 4 if, n>= 2m/(r+R).
Consider the value of the ratio m/(r+R) = x, where m the size of the mobile component (mobile agent) in bits and n is the loop number for requesting and replying messages, we can conclude the order of our choice for the mobility strategy in the following table:

	n<x	X<n<(3x/2)	(3x/2)<n<2x	n>2x
Best path	4	4	1	1
Good path	2(3)	1	4	2(3)
Worst path	1	2(3)	2(3)	4

Taking the 2nd column in the table where x<n<3x/2 as an example, we find that the design of the lowest response time is path 4, i.e. complete static system. The one that follows is path 1, i.e. mobile style. The biggest response time is when adopting partial mobility style. The analysis performed here is a stand alone one, i.e. it consider our software system as if it was the only one running on the hardware platform without any contention for hardware resources and without bottlenecks. This is the best environment we can expect for the system, so, if the response time obtained in this analysis is unsatisfactory, we should redesign the software architecture.

Step 7-mob: this step parameterize the EQNM that was developed in step 5. This is done by mapping the software model (mob?-EG-instance) onto the defined EQNM. This determines the environment based parameters of the model such as job classes, job service demands at different centers and job routing among the network centers.

Step 8: the set of EQNM models for the different paths are solved for different configurations and parameters. IBM/RESQ2 toll is used. The results will point out the change of response time as we change a parameter such as the number of jobs. No more details will be given as our interest is the mobility modeling techniques.

4. QN Performance Model Solution

In this section we will explore solving the QN model for the example that we have discussed earlier in 3.1. It is a mobile user that can connect wirelessly to two servers at locations L1 and L2. The user can use two styles, static style where he exchanges a set of n messages in the form of requests and replies with each server to satisfy the data he needs. In the 2nd style of communication, the user can fire a mobile agent towards the first server only. Due to the characteristics of the mobile agent, it will reach to the 1st server (CPU1) and do what is required, it then, autonomously will move it self to the 2nd server (CPU2) and search there about the user's needs and at last it moves itself to the location of the user carrying the answer to him.

We solve the two QN models in figures 16, 17. Figure 16 represent the case of static system and figure 17 represents the case of using the mobile agent paradigm. We use the Internet free QN Tool called "WinPEPSY". The assumptions that we have made are as follow:
The mobile agent code size is 64 KB.
$T_{CPU0} = T_{CPU1} = T_{CPU2} = 0.1$ msec, this is the time needed by a CPU to execute a block of code of one thousand machine instruction.
The value of r is 0.5 KB and R = 2.5 KB.
The wireless channel is considered a 54 Mb/sec (802.11a).
The value of "n" is taken as a parameter for the workload, it represents the number of the exchanged messages between the client and the server to satisfy a request, it also represents how large is the workload. We calculate the response time for both systems for two different of n (n=10 and n=100), these two values represent light and heavy workloads. The number of the jobs in the system is another parameter for the workload and we used it as the variation of the x-axis. The table below is extracted from the results of the tool.

line no.	System	n / NO.Jobs	10	20	30	40	50	60	70	80	90
1	Static Sys Res. Time (msec)	10	25.8	50	72	94	116	138	160	182	205
2		100	278	501	722	939	1262	1385	1603	1824	2042
3	Mobile Sys Res. Time (msec)	10	115.6	209	304	398	493	598	683	778	872
4		100	227	424	622	823	1020	1220	1427	1622	1819
5		100	225	423	621	820	1021	1217	1417	1620	1820

100	110	120	130	140	150	160	170	180
226	248	270	293	315	337	359	379	400
2267	2477	2710	2929	3149	3369	3593	3814	4031
967	1061	1158	1251	1364	1447	1535	1631	1721
2017	2220	2420	2621	2820	3010	3219	3419	3622
2026	2224	2419	2619	2823	3020	3219	3420	3609

Chart (1)

The chart plots line 1, 2 of the table, this chart compares the value of the response time when n changes from 10 to 100 in case of using the traditional static style for communication. It is clear that at a transaction of 100 messages, the response time of the transaction will be greater than that of 0nly 10 messages. It is clear that, as the number of jobs in the system increased, response time increases also.

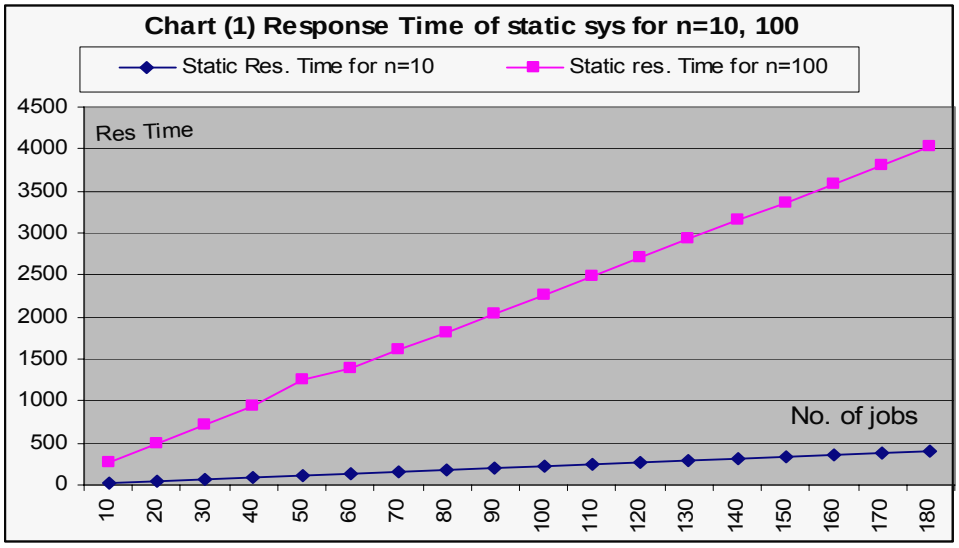
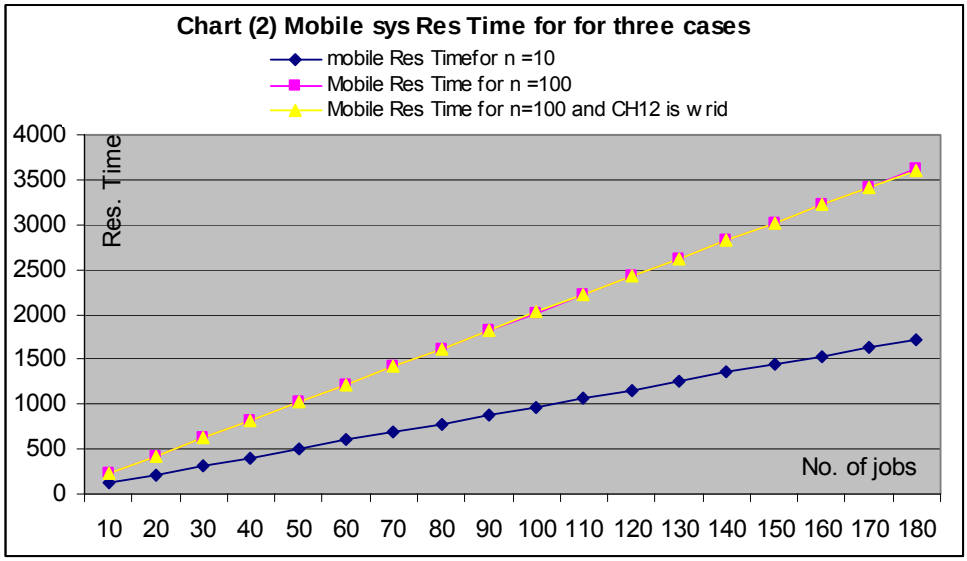


Chart (2)

Chart (2) compares the response time in case of the three cases for the mobile system. First, if n =10 (blue), and second when n = 100 (red) and the last one when n=100 and the mobile agent will use a wired network of speed 1000 Mb/sec to move on Channel12 (yellow).



The yellow and red curves are nearly coincident, meaning that it does not make any difference for the channel12 to be wired or wireless. This is because the agent size is small. Also, as before, response time for n=100 is greater than that for n=10.

Chart (3)

This chart compares the response time of the static system with that of the mobile system in case of n = 10. It is clear that the static system is highly better than the mobile agent system. This is due to the small size of the exchanged data for completing a transaction compared with the size of the mobile agent code. The time required to move the agent is greater than that for moving the data.

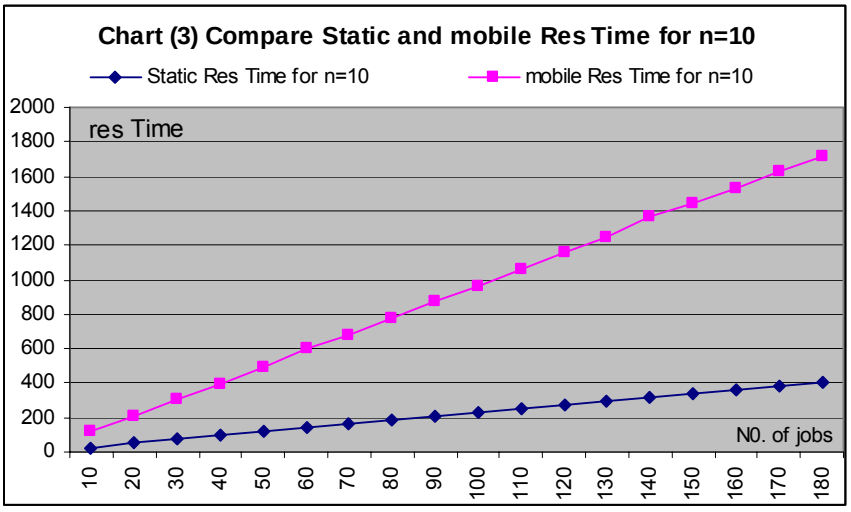
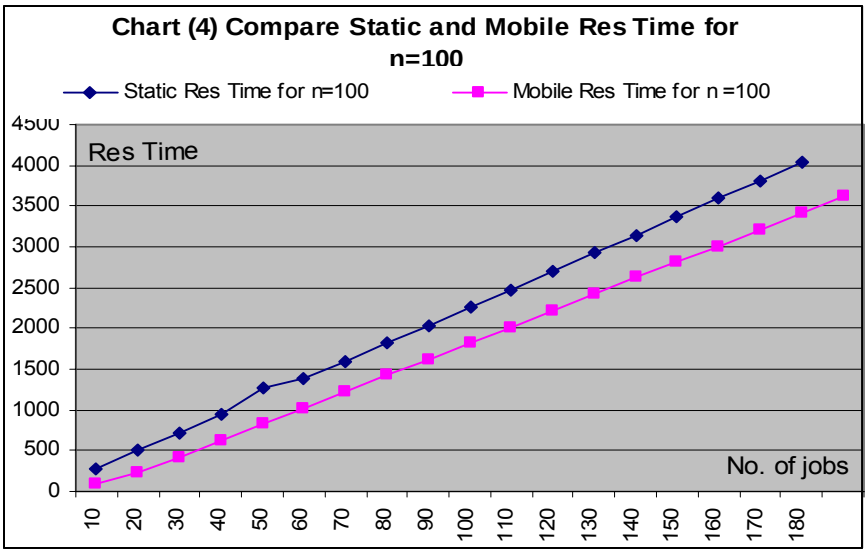


Chart (4)

Chart (4) shows the opposite of chart (3). Here, the response time of the mobile agent is less than that of the static system. The reason, n =100 and the amount of the exchanged data is large compared to the mobile agent code size. It is clear also that as the number of jobs in



the system increases, the response time increases also. We predict that if we take larger values for n, the static response time curve (blue) will go higher, making the difference much larger.

٥. Analysis and comparison

After presenting of two approaches for performance prediction of mobile systems and two techniques for mobility modeling, we here give our analysis for these topics and our point of view about certain aspects:

1. *Performance prediction approaches*: both of the two techniques were introduced within the context of a specific performance prediction approach. This refers to their limitation as they are not applicable to other approaches of performance prediction.
2. *Use of UML:* as all the currently proposed techniques for mobility modeling, these two presented use UML. This means that, UML became a widely used standard architectural description language, however, it lacks for mechanisms to model to model mobility. Most of the mobility techniques suggest extensions to UML to model mobility and that what the 2nd technique did.
3. *UML diagrams* used: the reason behind the widely acceptance of UML as a standard modeling language is its inclusion of a set of diagrams that model the different aspects of software. The first technique uses *Use Case Diagram* to model mobile entities and possible mobility paths and Activity *Diagrams* to model execution interactions within each path. The 2nd technique uses the *Sequence Diagram* to model execution interactions and *Collaboration Diagram* to model components locations and their movement.
4. *Mobility types*: software mobility can be categorized into two broad types, physical mobility and logical mobility. It is hard to find a technique that models both types into the software architectural design. The first technique claims it models both types, while the second models only logical mobility (case of mobile agent).
5. *Automation:* the first approach uses a prototype tool specifically designed for it, hence the tool is limited to the technique and the approach proposed. The 2nd approach does not use any tool to automate the process except only in the last step. It uses the IBM/RESQ2 queuing network tool for solving the QN-based performance model. The presence of a tool is essential for the success of any approach. The only commercially available software performance tool is SPE•ED and until now, it is limited to OO and distributed software.
6. *mobility and execution modeling separation*: first technique claims it separates between mobility modeling and execution modeling, however this is not clear. The 2nd technique does separate between the two concerns
7. *development possibility*: mobility modeling technique No.1 has a little chance to be developed and merged into a solid performance prediction approach for reasons such as using detailed ADs that describe both code execution and mobility behavior, using a prototype tool dedicated to that approach only. The 2nd technique has a better chance to be evolved as it is based on a well established approach (SPE).
8. *PRIMAmob-UML methodology is applicable* and it is based upon the SPE approach of C.U.Smith, what is needed is the presence of a tool to facilitate these cumbersome, lengthy calculation and modeling steps.

٦. Conclusions

As mobile devices are spreading and becoming part of our daily life in the last few years, mobile software systems have found great interest from software companies and engineers. Performance of such systems is critical for their success and should be predicted during their architectural design phase. This is due to their complexity and expensive development cost. To predict performance of mobile systems, we have to take into account their mobility behavior. The means for mobility specifying are still under development [1] and so far, there is no standard way for expressing mobility in UML [5]. Hence, our conclusion is that approaches proposed for mobile systems performance prediction is still taking the first steps in their way of maturity, however, few approaches for performance analysis and prediction for static systems have realized considerable advance [6]. We see that presence of a standard way for mobility modeling would encourage UML and performance tools manufactures to include this part in their tools. The presence of tools is necessary for developing the process of performance prediction of mobile systems.

References

[1] H. Baumeister, N. Koch, P. Kosiuczenko, and M. Wirsing. " ***Extending Activitiy Diagrams to model mobile systems***". In M. Aksit, M. Mezini, and R. Unland, editors, Netobject-Days, volume 2591 of lecture notes in Computer Science. Springer, 2003. ISBN 3-540-00737-7.

[2] S. Balsamo and M. marzola, "***A simulation –Based Approach to software Performance Modeling***", ESEC/FSE'03, Septemper 1-5, 2003, Helsinki, Finland.

[3][32] S. Balsamo and M. Marzola, ***"Simulation modeling of Software Architectures***", Proc. Of European Simulation Multiconference, pages 562-567, Nottingham, June 2003b.

[4] Object Management Group. ***UML profile for schedulability, performance and time specification***. Final adopted specification ptc/02-03-02, OMG, March 2002.

[5] Simonetta Balsamo, Moreno Marzolla "***Towards performance Evaluation of Mobile Systems in UML***"Proc. Of ESMc'03, Napoles, Italy, Oct. 27-29, 2003, pp. 61-68, EUROSIS-ETI, ISBN 90-77381-04-x.

[6] Simonetta balsamo, Antinisca Di Marco, Paola Inverardi and Marta Simeomi , ***"model-Based Performance prediction in software development: A Survey",*** IEEE transactions on Software Engineering, vol. 30, No. 5, May 2004.

[7] Murray woodside, Dorina Petriu, "***Capabilities of the UML Profile for Schedulability Performance and Time (SPT)***"April 2004.

[8] Jose Merseguer, Javier Campos***, "Exploring roles for the UML Diagrams in software Performance Engineering***". Proceedings of the International Conference on Software Engineering Research and Practice, SERP '03, June 23 - 26, 2003, Las Vegas, Nevada, USA, Volume 1. CSREA Press 2003, pp.43-47, ISBN 1-932415-19-X

[9] Murray woodside, Dorina Petriu, "***Capabilities of the UML Profile for Schedulability Performance and Time (SPT)***"April 2004.

[10] C.U. Smith and L.G. Williams, "**Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software"**. Addison Wesley, 2002.

[11] C.U. Smith and L.G. Williams, "**Performance Engineering Evaluation of Object-Oriented Systems with SPE.ED"**, Springer LNCS 1245, pp. 135-153, 1997.

[12] Vittorio Cortllessa and Raffaela Mirandola, " ***PRIMA-UML: a performance validation incremental methodology on early UML diagrams***", Science of Computer Programming 44 (2002) 101 – 129.

[13] Vincenzo Grassi, Raffaela Mirandola, " **PRIMAmob-UML: a methodology for performance analysis of mobile software architectures**", Proceedings of the 3rd international workshop on Software and performance, Rome, Italy, Pages: 262 - 274 , 2002, ACM Press New York, NY, USA.

[14] C.U. Smith and L.G. Williams***, "Performance Engineering Evaluation of Object-Oriented Systems with SPE.ED", Springer LNCS 1245, pp. 135-153, 1997.***

[15 L.G. Williams and C.U. Smith ***"Performance Evaluation of Software Architecture", Proc. ACM Int'l Workshop Software and Performance, pp. 164-177, 1998.***