Two Early Performance Analysis Approaches at work on Simplicity System*

Antinisca Di Marco[1,2] and Francesco Lo Presti[2]

[1] Computer Science Department, University College London, Gower Street, London WC1E 6BT, UK a.dimarco@cs.ucl.ac.uk
[2] Dipartimento di Informatica, Università di L'Aquila, L'Aquila, Italy lopresti@di.univaq.it

Abstract. A broader and broader range of ICT devices and network-based services are made available to users. Today, to access them, the user has to use heterogeneous hardware and software technologies, has to properly configure them, and must be authenticated and charged in different way. The result is a growing complexity which can become a serious obstacle to both users and service providers.
The IST-Simplicity project aims at designing and deploying a brokerage level able to reduce such a complexity by decoupling user needs and user devices, as well as service deployment and fruition, from the underlying networking and service support technologies. Due to its nature, the project has to guarantee high performance and the early evaluation of the Simplicity Architecture becomes a key ingredient to guide the developers towards an effective and efficient high level architecture.
In this paper, we propose and analyze two models of the a portion of the Architecture of the Simplicity System, namely the User Terminal System, having two different level of details. We report the lesson learned and the comparison of the two considered approaches.

1 Introduction

As technology develops, a broader and broader range of ICT devices and network-based services are made available to users. As for today, to use these services, the user has to access them through heterogeneous technologies and protocols, must use different devices, must properly configure them, must be authenticated and charged in different way by the different services. This implies an enormous burden of complexity for users, service providers and network operators, complexity which can become a serious obstacle to the development and deployment of new services.

To overcome these limitations, the IST-Simplicity project [2] aims to ease the user interaction with devices and new services. The project goal is to design and deploy a brokerage level able to decouple user needs and user devices, as well as service deployment and fruition, from the underlying networking and

* This work has been partially supported by IST FP6 SIMPLICITY project. Antinisca Di Marco was also partially supported by the Segravís Grant.

service support technologies. Within Simplicity, users are characterized by a personalized profile to be used for different services/transactions, eventually based on different classes of terminal. Such profile allows an automatic, transparent personalization and configuration of terminals/devices, and provides a simple and uniform way to be recognized, authenticated, located and charged. The user profile is stored in a Simplicity Device (SD). Users personalize terminals and services by the simple act of plugging the SD in the chosen terminal. The Simplicity System encompasses the Simplicity Device and a Brokerage Framework. The Brokerage Framework provides policy-based mechanisms (e.g., policies for mobility support, QoS, security) to orchestrate and adapt network capabilities, taking into account user preferences and terminal characteristics.

Due to the stringent performance requirements its services have to guarantee, it is important to analyze the performance of the Simplicity System. To be effective, though, such an analysis should be made at the first phases of the development process. Indeed, as in most complex software systems, performance problems often result from early design choices and inaccurate decisions at early phases may imply an expensive rework, possibly involving the overall system. The benefits of early performance evaluation prevents late performance fixes of the implementation by guiding the software developers in the decisions at the design level.

In this paper, by combining different performance evaluation methodologies, we plan to identify the presence of potential bottlenecks to be addressed during the design and development of Simplicity User Terminal System, and to study how the strengthening of the hardware capabilities can be useful to overcome performance problems originated from specific software design solutions. We propose and analyze two models of the User Terminal System that includes the local portion of the Simplicity Brokerage Framework. The first model is based on a product form QN model [5] representing the software architecture of the User Terminal System. In this model each service center models a different software component of the system and not a hardware device (such as CPU, disk, and so on). The formalism used for the second model is the Layered Queueing Network (LQN) model [9], an extension of the well-known Queueing Network model, developed especially for modeling concurrent and/or distributed software systems. In this modeling also hardware platform components are considered.

The approaches we used, as others in the literature [1], allow early performance analysis to support software designers during the software development process. These approaches aim at addressing performance issues as early as possible in order to reduce wrong design decisions. Both of them try to adapt a system performance analysis methodology to software systems and they assume as available, at design time, information about the hardware platform the software will run on. By filling the knowledge gap between the software and the performance worlds, these methodologies provide the designers an easy and quick validation technique of the software design against the performance requirements.

The generated models allowed us to analyze the performance of the User Terminal System under different setting, in order to identify software bottlenecks and hardware limitations. Starting from this models, we have also explored the benefits of the multi-threaded implementation and the prudent deployment of the software components onto hardware devices.

Moreover, the approaches allow to assess the relative benefits of design alternatives. This process guarantees a refinement of the software architecture/design, the definition of the potential best deployment of software onto the hardware platform and finally hardware platform limits identification.

The paper is organized as follows: Section 2 outlines the Simplicity system, while in Section 3 it is reported a more detailed description of the User Terminal Simplicity Subsystem. Section 4 introduces the importance of the performance analysis early in the software life-cycle and presents the two approaches we have used to analyze the User Terminal Subsystem. It also reports the modeling of considered portion of the Simplicity project in both the methodologies and the analysis results obtained. In Section 5 we discuss final remarks on the used approaches and their comparison, while Section 6 concludes the paper.

2 Simplicity System

In this section we briefly review the Simplicity Architecture [2]. It is basically composed by a Brokerage Framework and the Simplicity Device.

The key role of the Simplicity Device (SD) is to store in a secure and safe way users profiles, preferences and policies to allow dynamic and automatic discovery and registration of terminal and network capabilities. It also facilitates service adaptation to various network technologies and related capabilities. The SD requires memory for storage of information and is typically located within the user devices, *e.g.*, a cellular phone, a PDA. The Brokerage Framework comprises a Terminal Broker module, which is primarily used to allow the interaction of the SD with both the terminal and the network, and a Network Broker module.

The Terminal Broker (TB) is also hosted by the user terminal/device. The TB is the entity that interfaces the user to the network. TB provides the user with a mean to read/write/modify the personalization information stored in the SD. The TB can configure applications and network settings, and can dynamically download plug-ins and applications.

The Network Broker (NB) provides: i) a platform for service deployment, advertisement, personalization; ii) service adaptation capabilities to the considered context (location, time, etc.); iii) orchestration of events and the handling of simultaneous access of several users to the same resources, services, and locations. The platform takes into account user profiles, terminal capabilities as well as the network side features. In order to achieve the above objectives, the NB needs to communicate with the Terminal Broker in order to retrieve information about the user profile, his/her context and the device capabilities.

The architecture of the Terminal and Network Broker is based on the concept of subsystems. A subsystem represents a piece of software that is able to perform some system functionality. Subsystems communicate indirectly by means of a specific subsystem, the Mediator, which relays asynchronous messages - called events - between them.

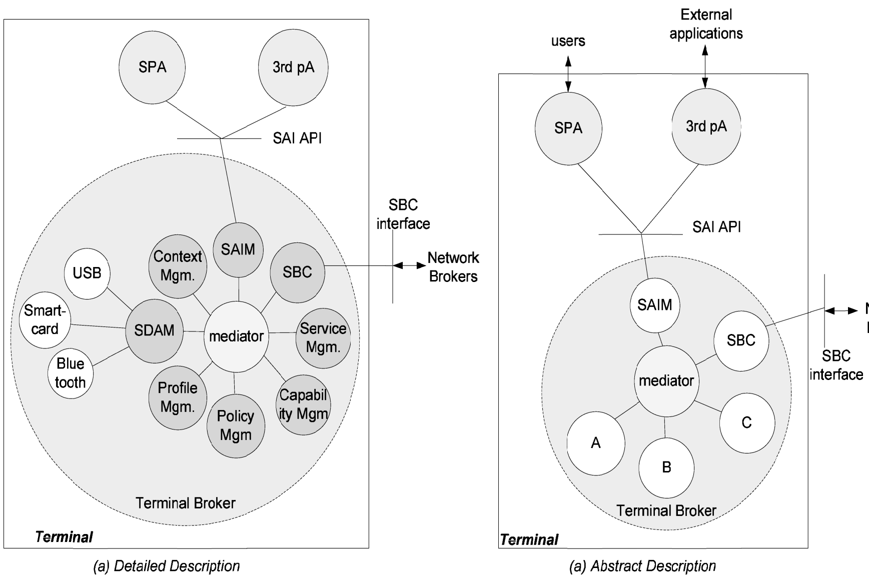

Fig. 1. Architecture Overview of the User Terminal System.

3 Architecture of the User Terminal System

In this section we describe in more detail the User Terminal System Architecture. Figure 1 (a) shows the detailed software architecture of the User Terminal System. Terminal broker functionalities are encapsulated in separate subsystems. The communication among the subsystems is based on the publish/subscribe event dispatching protocol where the Mediator covers the role of the event dispatcher. Messages are sent by one subsystem and received by one or more subsystems. Senders and receivers do not know each other. Rather, the Mediator uses a set of rules to determine which messages are to be forwarded to which receivers. Every TB has a single Mediator which is responsible for all messages relaying within it. The Mediator itself is not proactive; it simply reacts to messages that have been sent by the various subsystems.

If a subsystem needs to communicate with a subsystem on another Broker an external communication takes place. External communications are relayed via the Mediator to the `Simplicity Broker Communication` (SBC) Subsystem which forward it across the network to remote subsystems, *e.g.*, subsystems in the Network Broker. SBC provides transparent access to remote Network Brokers, while hiding the details about the specific transport protocol used to establish and manage the connection between brokers.

The `Simplicity Application Interface Manager` (SAIM) subsystem allows a 3rd party and/or legacy applications (*3rd pA* in the Figure 1) to run on top of the Terminal Broker, and to use the functionalities of Simplicity, without being aware of the detailed mechanisms of the Simplicity brokerage framework. This subsystem offers 3rd party applications a standardized interface. The SAIM also interfaces the Simplicity Personal Assistant (SPA) that is a Simplicity subsystem that directly interface the user. The SPA is mainly involved in user authentication and users preference management.

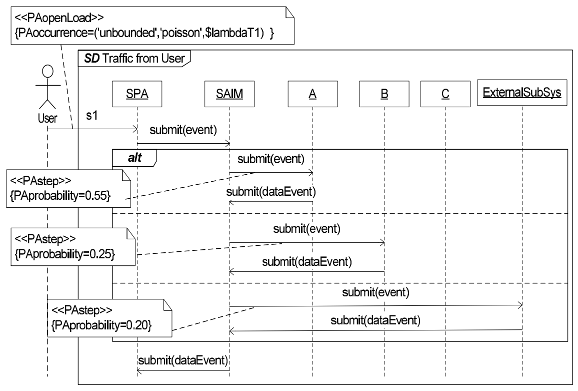

Fig. 2. Traffic from the Simplicity Users

For the sake of simplicity, we do not give details on the other subsystems in Figure 1(a) which will not be object of our performance investigation in this paper. For the performance analysis, we will instead refer to the abstract architecture of the TB (see Figure 1 (b)) where, except for the SAIM, Mediator and SBC subsystems, three more abstract subsystems, namely A, B and C, are considered. We assume that such subsystems provide the services required by the the other subsystems. We decided to use this abstract view to simplify the presentation without entering in deep details in the description of specific Simplicity Subsystems.

For the analysis, we assume the terminal broker subsystems serve three type of requests: the first type are user requests and are entered via the SPA subsystem; the second are 3rd Party Applications requests and the last are requests from the NB. We assume these requests are for services provided by one or more of the subsystem A, B and C. Moreover, requests originated by the user or by 3rd Party Applications may require service from an external remote subsystem.

In Figure 2 we describe the Simplicity system behavior when it receives a request from the users. Such a request can involve either the subsystem A, B or an external (remote) subsystem with probability set to 0.55, 0.25 and

0.20, respectively (Note that such probabilities are annotated in the sequence diagram by means of the `PAstep` stereotype of the UML Profile for Schedulability, Performance and Time [7]).

For what concern the requests coming from 3rd Party Applications (Figure 3), they can involve either the subsystem A, B or a remote subsystem with probability 0.25, 0.45 and 0.30, respectively. Finally, the requests coming from remote subsystems (Figure 4) involve A, B or C with the probability 0.50, 0.25 and 0.25, respectively.

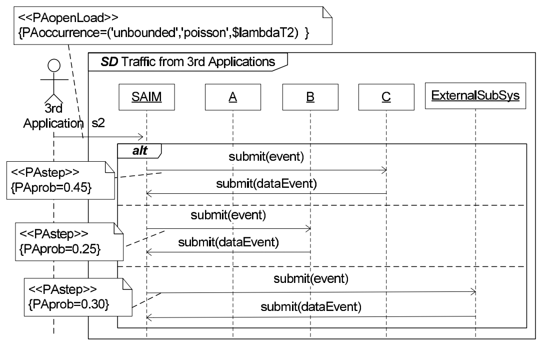

Fig. 3. Traffic from the 3rd Party Applications

In all the three diagrams we use the UML profile for Schedulability, Time and Performance [7] to annotate additional information describing performance related aspects of the system. More precisely, such information is the workload intensity for each type of incoming traffic (by means of the SPT `<<PAopenLoad>>` stereotype), and the probabilities an alternative behavior can occur (by means of the SPT `<<PAstep>>` stereotype). The notations here are parametric since they are expressed in the diagram by variables as the symbol `$` indicates. We discuss later the setting of values for such variables considered in the evaluation step.

Information about the service demand of each component interaction is reported in table 1. In the table the service demands indicated represent the assumed number of instructions the interaction requires to the subsystem receiving it. We also include in the table a column for the `LAN` resource that is involved every time a remote communication occurs (e.g., when a local subsystem requires a service to a remote one). We assume here that a remote interaction requires the sending of two messages through the `LAN`, which we assume requiring the same amount of time of 50 software instructions each.

The service demand can also be annotated in the interactions in the sequence diagram by using an opportune tag value of the `PAstep` stereotype. We decided to report this additional information in a table since the service demand must

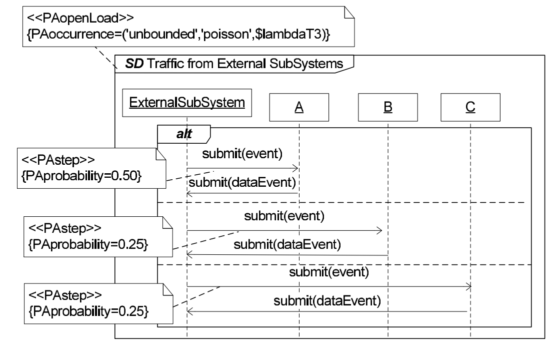

Fig. 4. Traffic from a Remote Subsystem

	SPA	**SAIM**	**Mediator**	**A**	**B**	**C**	**SBC**	**Ext.SubSyst.**	**LAN**
submit(event)	5 istr.	3 istr.	4 istr.	8 istr.	100 istr.	12 istr.	–	20 istr.	50
submit(dataEvent)	5 istr.	3 istr.	4 istr.	–	–	–	–	–	50
deliveryEvent	–	–	–	–	–	–	8 istr.	–	50
backData	–	–	–	–	–	–	8 istr.	–	50

Table 1. Service Demand

be reported for all the component interactions making the diagram difficult to read.

4 Performance Analysis Approaches

In this section we briefly report the used approaches for the early performance analysis. The first, based on Queuing Networks [5], does not require description of hardware capabilities and analyzes the system only from the software aspects perspectives, the second instead combine software and hardware aspects by allowing the reasoning on hardware platform too. Both the approaches can be applied at the design level and for them we show the modeling step, the analysis results and the feedback at the software life cycle level the interpretation of the analysis results suggests.

Before to go through the methodologies and the evaluation steps, we discuss the performance analysis goals and the values setting of the reported experiment.

Analysis Goals - Through the performance analysis we want to identify the software bottlenecks and the hardware limitations in order to suggest (i) design/implementation alternatives to overcome the software bottlenecks, (ii) a better deployment strategy of the software component onto the hardware platform that optimizes the usage of the hardware facilities, (iii) and to determine which resource should be powered to improve the system performance.

The performance indices we are interested on are: the response time of the system, that is the end-to-end time the system spends to satisfy a request, and the utilization of the Terminal Broker subsystems and of the hardware resources (only in the second approach) in order to identify the software bottlenecks and the hardware limitations.

Value Settings - In our experiments we evaluated the generated performance models by considering several workloads and behavior probabilities. For the sake of space, we cannot report all the values settings we considered and analysis results we obtained. Among them, we selected the set of values we described in the previous section for their representativeness. For them, we conducted a scalability study by varying the rates λ_1, λ_2, λ_3 of user, 3rd party and remote requests, respectively. For simplicity, here we consider a fixed ratio among the three workloads by assuming $\lambda_1 = \lambda_2 = 10\lambda_3 = \lambda$, *i.e.*, we assume the external requests rate to be one tenth of the others which are assumed to be equal. Hence, we study the system performance as function of λ.

4.1 QN-based Approach

The first approach we used (defined in [3, 4]) allows the quantitative analysis of the Software Architectures (SA) by generating a multi-chain Queueing Network model (QN) [5], from the SA specification described by UML 2.0 Diagrams [8]. The used diagrams are the Use Case Diagrams, to specify the services provided by the software system, the Component Diagram to model the SA structure in terms of the software components and connectors, and the Sequence Diagrams,to describe the behavior of the software system, in terms of component interactions, when a request of a given type enters it.

The approach requires that such diagrams are annotated with additional information by means of the UML profile for Schedulability, Performance and Time (SPT) [7]. The needed information is the workloads characterization and the execution probabilities (annotated in the Sequence Diagrams), and the service demand of each component together with the scheduling policies of their waiting queues (annotated in the Component Diagram).

The approach identifies the QN customer types from the Use Case diagram: each use case in the diagrams represents a customer type in the QN. The behavior of such customer in the QN is described by the Sequence Diagrams that are, hence, used to generate the QN topology and the chains definition. A Sequence Diagram gives also information on the workload intensity of the class of customer it describes. Finally, the information for the parameterization of the service centers of the QN, such as the type and the service rates of the service centers and the scheduling policies they use to extract jobs from their waiting queues, are extracted from the Component Diagram.

In the approach, a QN model represents the software architecture and a service center corresponds to an architectural component. Differently from other QN-based approaches, the service centers do not represent hardware devices (such as disk and processor), but they represent software components. The QN

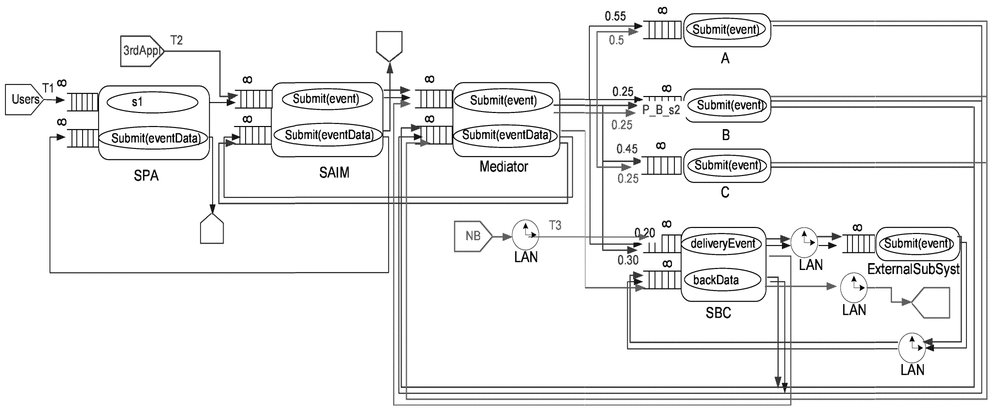

Fig. 5. QN Model for the User Terminal System.

topology describes how the software components are combined to form the software system and how a system service request is accomplished through the software components interactions.

User Terminal System Modeling By applying the methodology on the User Terminal System we had obtained the QN model in Figure 5. As required by the approach, we modeled the system though UML 2.0 diagrams. For the sake of space, we do not report the Use Case and the Component Diagrams, whereas the annotated Sequence Diagrams describing the incoming traffic (QN customers behavior) are reported in Figure 2, 3, an 4.

Since we have three type of arrivals that specify three open workloads, in the QN model we have three sources that generates the relative customers with the frequency annotated in the Sequence Diagrams. The QN model also contains 8 service centers, one for each Simplicity subsystem composing the User Terminal System and one for the remote subsystem. These service center are infinite FIFO queuing centers providing one (for A, B, C, ExternalSubSyst subsystems) or two services (for SPA, SAIM, Mediator and SBC subsystems). The service time for each service is obtained by dividing the correspondent service demand in the Table 1 per the speed factor of the logical device we suppose executes the actions needed to provide the services (in the modeling of the Simplicity system we assumed that such logical devices can all process 1000 instructions per second). The information of the service demands is extracted from the UML Component Diagram. The approach introduces in the QN three delay centers (namely the LAN centers) to model the extra delay that characterizes the remote communications.

Finally the interconnection of the service centers in each chain is extracted by the Sequence Diagram. In the QN model we represent each chain with a different color, blue for the traffics generates by the users (T1 traffic type), red for the one generates by the 3rd party applications (T2 traffic type) and green for the one generates form the Network Brokers (T3 traffic type). In each chain there is a branching point modeling the `alt` interaction operator in the sequence diagrams. The probability a QN customer follows a particular route in a branching point is extracted from the annotations in the sequence diagram.

Performance Analysis Results Here we report the QN-model performance results for the selected experiment (values of Table 1).

In Figure 6(left) we plot the utilization as function of λ for selected subsystems. As expected, from the software architecture point of view, subsystem B, with its 100 instructions per service request, represents the system bottleneck and saturates when the number of events approaches 190. All other subsystems have much lower utilization levels.

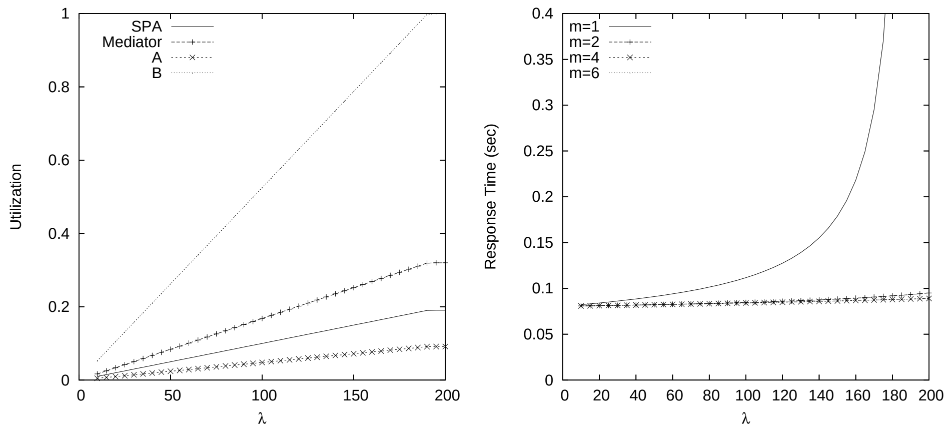

Fig. 6. QN-Model Performance: (left) Subsystem Utilization; (right) User Events Response Time as Function of the Number of Events per Second for Different Level of Replication m of Subsystem B.

For this architecture, a possible alternative is represented by a replication of subsystem B (or a multi-threaded implementation). Replication results in the removal of the software bottleneck (for $m > 3$ the bottleneck now becomes the Mediator, which saturates for about 500 events per second).

In Figure 6 (right) we plot the response time for the User events (T1 requests) as function the number of events per second for different levels of replication m of subsystem B. It is worth observing that for less than 200 messages a replication level of 2 is sufficient to improve performance.

4.2 LQN-based Approach

The second approach is based on LQN performance models. LQN was developed as an extension of the well-known Queueing Network (QN) model [9]. With respect to traditional QN, in LQN a server, to which customer requests arrives and queue for service, may become a client to other servers from which it requires nested services while serving its own clients. An LQN model is represented as an acyclic graph whose nodes (named also tasks) are software entities and hardware devices, and whose arcs denote service requests. The software entities are drawn as parallelograms, and the hardware devices as circles.

A software or hardware server node can be either a single-server or a multi-server. A LQN task may offer more than one kind of service. An entry has its own execution time and demands for other services (given as model parameters). Each server has an implicit message queue, where the incoming requests are waiting their turn to be served. Servers with more then one entry still have a single input queue, where requests for different entries wait together.

The approach we follows is derived from [6], where a methodology for the derivation of LQN models from UML specifications is defined, here adapted to use the same UML 2.0 Diagrams considered for the QN-based approach. We additionally use a Deployment Diagram to describe the mapping of the software component on to hardware platform the software system will run. Again, this diagram is annotated with the SPT profile to bring information about the hardware devices.

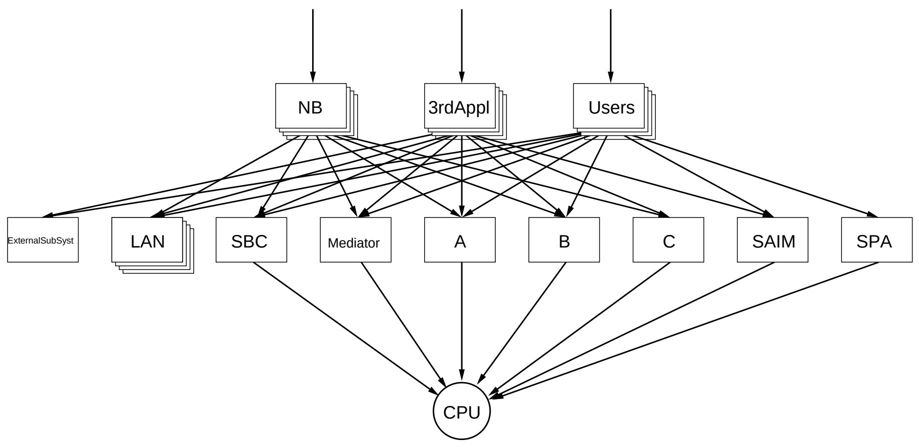

Fig. 7. LQN model for the User Terminal System.

The approach extracts the external arrival and/or reference tasks from the Use Case diagram with each use case identifying an external arrival/reference tasks. Each software component in the Component Diagram is mapped to a

LQN software task, and each node (representing processor, disks, communication networks, etc.) in the Deployment Diagram is mapped to a hardware devise. Task details, i.e., type of service, entries, phases, scheduling policies service demands and visit ratios are derived from the sequence diagrams (in the QN approach this very same information is used to generate the QN topology and chains definition), whereas device details are extracted from the Deployment Diagram. The obtained LQN model thus represents both the software architecture we want to analyze and the underlying hardware infrastructure.

User Terminal System Modeling Fig. 7 represents our LQN model of the Simplicity Terminal Broker derived from the UML 2.0 diagrams.

We have three external arrivals representing the external workload whose sources, generating the respectively traffics, are modelled, in the LQN framework, by the three (infinite-threaded) tasks on the top of Figure 7 (namely NB, 3rdAppl and Users).

The model contains also 8 software tasks (corresponding to A, B, C, ExternalSubSyst, subsystems SPA, SAIM, Mediator and SBC subsystems) all with one entry (we could have diversified the entries for the A, B, C and ExternalSubSyst which offer two services, but since we assume the same service demand it was not necessary). All these tasks, except ExternalSubSyst, are executed on the local CPU, since the Simplicity Terminal Broker has to be deployed on the user terminal that in general has a single CPU. The CPU is modelled in the LQN model by the device at the bottom of the Figure 7. We assume the CPU can execute 1000 instruction per second. Task CPU demand is obtained by Table 1. Tasks visit ratio parameters are extracted by the Sequence Diagrams and their annotation.

An additional infinite-threaded task is added to model the LAN delay.

Performance Analysis Results For the experiment under study, the QN-based analysis has shown us that subsystem B is a software bottleneck, which can be removed by replicating the subsystem.

The LQN model now allows us to also account for the hardware resources under different alternatives (we remark that information about hardware devices and deployment may not be available at the early design stage). Figure 8 (left) shows again the response time for the User events as function the number of events per second for different level of replication m of subsystem B. In this scenario, with all terminal subsystems sharing the same CPU, we observe that the system saturates for only 80 events per second. This is no surprise given the speed of the CPU. More importantly, the figure shows that replication of subsystem B further degrades performance. Indeed, because of the single shared CPU, replication, while reducing waiting time to access subsystem B services, increases the execution time of all subsystems. Overall, we observe degraded performance.

To overcome the hardware bottleneck, we can consider a faster CPU or multiple CPU. In Figure 8 (right) we show the same curves for a scenario with 4

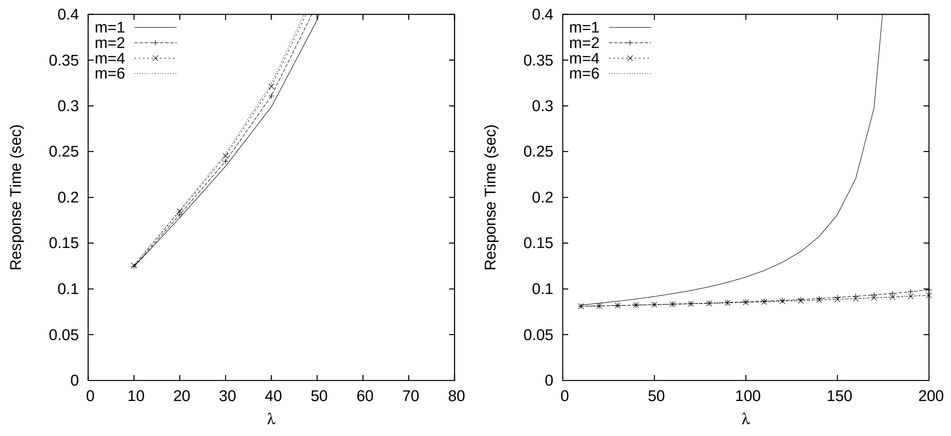

Fig. 8. LQN-Model Performance. User Events Response Time as Function of the Number of Events per Second for Different Level of Replication m of Subsystem B: 1 CPU (left); 4 CPU (right).

CPU. We observe that the use of multiple CPU moves the bottleneck back to the subsystem B. As also shown by the QN analysis, this bottleneck can be now removed by replicating the subsystem B itself.

5 Summing-up

We have shown the application of two software performance analysis approaches to the Simplicity User Terminal System. Both of them have specific aspects and characteristics. The first, based on QN model, does not require description of hardware capabilities and analyzes the system only from the software perspective. The second, based on LQN model, instead, combines software and hardware aspects by allowing the reasoning on hardware platform too. Both approaches are usable at the design level even if the first approach can be applied earlier than the second that requires additional information on the hardware platform, in general available later in the design phase.

The QN-based approach is useful to improve the software architecture of the system by pointing out wrong design decisions (such as software components selection, communication protocol, software system structuring), and software bottlenecks of the proposed architecture, as we experienced in the analysis we reported above. Moreover, it provides a mean to easily evaluate design alternatives that can overcome the highlighted problems (as, in the case we showed, the multi-threading for the B subsystem). Finally it can suggest action that should be avoided later in the software development process.

The second approach is more expressive than the first one, since it combines software and hardware aspects. To be applied, it is necessary a detailed description of the hardware platform and of how the software components are deployed over it. In general, such information is available later in the software life cycle, and, sometimes, when the software system has been implemented. The LQN-based analysis can be useful to identify hardware limits and the minimum hardware upgrade needed to overcome the performance problems pointed out.

In case the platform hardware is, instead, a project constrain, i.e., the system will run on a given platform, the second approach helps to find the better software components deployment in order to make the most of the hardware capabilities available. In this scenario, where we cannot strengthen the hardware facilities to overcome performance problems, it becomes of extremely importance to make right decisions at the software design level and hence the analysis provided by the QN-based. This, as we said, points out the most weak parts of the software system that should be carefully treated, and can suggest design alternatives that improve the performance of the whole system.

The QN model, albeit it abstracts system hardware, allowed us to clearly identify an alternative which correct a potential design mistake. Indeed, while it is always possible (or easier) to upgrade the system hardware it is not so for the software after the design stage.

6 Conclusion and Future Works

In this paper we have presented an early performance analysis of the Simplicity User Terminal System. We have proposed and analyzed two models of it and the used approaches provided an easy and quick validation techniques of the software design against the performance requirements.

As expected the used approaches had two different level of details. The QN-based allows to study the performance of the software architecture by pointing out the software bottlenecks and design decisions that could degrade the performance of the whole system. Whereas the second approach is useful to identify hardware limits and wrong decisions in the system deployment.

Future works include to study the potential of the modeling approaches at the early design stage and assess their relative effectiveness. At the moment, we are refining the entire system model to study several architectural alternatives. We also plan to complement this work by measurements of the system prototype which will provide the means to validate this performance study.

More interesting and challenging study will be the performance analysis of the Network Brokers and of the whole brokerage framework. For what concern the analysis at the software architecture level, of the peculiar importance it is the identification of how many NB the brokerage framework should be composed of, and how their interconnection structure should be, in order to obtain a scalable system. Whereas, at the hardware level, it should be necessary to understand how much processing power and bandwidth are necessary to guarantee a good performance level. Note that, in these analyses of the brokerage framework, we can mostly reuse the two models of the Terminal Broker we have here reported since its architecture is similar to the one of the Network Broker.

References

1. S. Balsamo, A. Di Marco, P. Inverardi and M. Simeoni, “Model-based Performance Prediction in Software Development: A Survey”, *in IEEE Transactions of Software Engineering*, pp. 295–310, vol. 30 (5), 2004.
2. N. Blefari Melazzi and al., “The Simplicity Project: easing the burden of using complex and heterogeneous ICT devices and services. Part I: Overall Architecture”, *in Proc. of IST Mobile and Wireless Summit 2004*, Lyon, France, June 2004.
3. A. Di Marco, P. Inverardi, ”Compositional Generation of Software Architecture Performance QN Models”, *Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pp. 37-46, June, 2004, Oslo, Norway.
4. A. Di Marco, ”Model-based Performance Analysis of Software Architecture”, Ph.D. Thesis, 2005.
5. E. D. Lazowska, J. Zahorjan, G. Scott Graham, K. C. Sevcik, ”Quantitative System Performance: Computer System Analysis Using Queueing Network Models”, *Prentice-Hall*, 1984.
6. D.C.Petriu, H. Sen, “Applying the UML Performance Profile: Graph Grammar-based Derivation of LQN Models from UML Specifications”, *Lecture Notes in Computer Science 2324*, pp.159-177, Springer Verlag, 2002.
7. Object Management Group, ”UML Profile, for Schedulability, Performance, and Time”, *OMG document ptc/2002-03-02*, `http://www.omg.org/cgi-bin/doc?ptc/2002-03-02`.
8. Object Management Group, ”Unified Modeling Language 2.0”, 2003, `http://www.omg.org/uml/`.
9. C.M. Woodside, J.E. Nelson, D.C. Petriu and S.Majumdar, “The Stochastic Rendevous Network Model for Performance of Synchronous Client-Server-like Distributed Software”, *in IEEE Transaction on Computers*, Vol. 44, pp. 20-34, January 1995.