Automated performance modelling of distributed software architectures: middleware and networks

Tom Verdickt
Supervisor(s): Bart Dhoedt, Frank Gielen

Abstract—One of the most critical aspects of the quality of a software system is its performance. However, system designers usually only take the functional aspects of the system into account during system design, resulting in the need for expensive new hardware, partial redesigns, etc. While performance modelling formalisms and quantitative solution methods were designed to integrate performance considerations into the development cycle, they remain hard to use, because system designers need to be familiar with the modelling formalisms and all the aspects of the system and all its components.

This research proposes a model transformation framework to automatically include the architectural impact and the overhead of using middleware, reducing the burden on the system designers. Also, a tool has been designed to allow general-purpose software performance models (LQN) to interact with specialised network models (NS), allowing very accurate modelling of the network and its impact on the overall system performance.

Keywords—performance engineering, MDA, UML, LQN, NS

I. INTRODUCTION

One of the most critical aspects of the quality of a software system is its performance. However, software engineering methodologies strongly focus on the functionality of the system, while applying a “fix-it-later” approach to performance aspects. The system is designed to meet its functional requirements, postponing considerations about the nonfunctional requirements (such as performance) to the later development stages. As a result, lengthy fine-tunings, expensive extra hardware, or even redesigns are necessary for the system to meet the performance requirements. And even with fine-tuning, there is no guarantee that the system performance will be appropriate.

A. Software Performance Engineering (SPE)

A better approach is to use software engineering techniques that allow to integrate performance considerations into the design process [1]: (predicted) system performance should be evaluated throughout the entire development cycle, by using performance modelling and quantitative solution methods. Several performance modelling formalisms have been designed (e.g. queueing networks, Petri Nets, LQN [2]), together with automatic solvers and simulators to extract performance estimates from the models.

This methodology for performance engineering has an important drawback: new models need to be created in a performance modelling language unfamiliar to the designers. Therefore, much recent research is aimed at automating the performance modelling process by automatically transforming general-purpose system models (such as UML) into performance models (e.g. queueing networks) [3]. This allows designers to model the system using the formalisms they are familiar with (e.g. UML) and to obtain the performance models automatically.

Another important problem in software performance engineering is the required level of detail in the models. In order to obtain accurate performance estimates, the model should be rather detailed concerning the most important, performance-limiting parts. This means that the designers need a thorough knowledge of the internals of all the system components. This is where Model Driven Architecture may help to model the system.

B. Model Driven Architecture (MDA)

The Model Driven Architecture (MDA) [4] is a recent effort to improve the use of modelling in system design, by prescribing how a system should be modelled. MDA describes what types of models should be used, how those models should be used, and how the model types relate to each other.

An important aspect of MDA is the definition of different categories of models. The most important model types are the platform-independent models (PIM) and the platform-specific models (PSM). A PIM addresses the operation of the system, but does not include anything that is specific for a certain technology or language. PSMs give a more detailed, lower-level view of the system, taking (part of) the underlying platform into account. MDA also focuses on transformations between system models (most importantly from platform independent models to platform specific models).

C. Distributed systems

System modelling, and specifically performance modelling, becomes even more complex when considering distributed systems. Distributed systems are a response to the growing demands for processing power and the geographical spreading and heterogeneity of processing power, data sources, and storage. They consist of several collaborating components (both hardware and software) connected by a network.

Often, middleware is used to enhance the interoperability between the various system components. Middleware offers the advantage of location transparency, platform and programming language independence, event handling, etc. Important middleware standards include the Common Object Request Broker Architecture (CORBA), Java Remote Method Invocation (Java RMI), Web Services, etc.

The growing interest in distributed systems has resulted in a growing interest in performance engineering techniques for those systems. Several efforts to model and predict the performance of middleware-based systems have already been under-
taken. Using these models still requires a detailed knowledge of the internals of the middleware (and of the modelling language itself) in order to be able to adjust the model to the specific characteristics of the system and to integrate the middleware model into the overall model of the system.

Another part of the difficulty of modelling distributed systems is the network. Many performance engineering formalisms (e.g. queueing networks) are targeted at modelling software, running on an abstract processing device. There is little direct support for modelling the network details of the system, such as bandwidth or packet loss. Similarly, tools dedicated to network modelling and network simulation exist (e.g. NS, OPNET Modeler), but those are not really suited for general software modelling. However, the network is a critical and possibly performance-limiting component of a distributed system. Therefore, it should be an integral part of the system model.

II. MODULAR MODELLING PLATFORM

The goal of this research is to develop a modelling platform that allows a modular and semi-automatic modelling of the system (Fig. 1). The system designers need to provide a network model and a high-level model of the software architecture. The platform will automatically include the middleware impact into the model (following some middleware usage information provided by the system designers). Existing tools can then be used to transform the obtained middleware-aware software model into a performance model [3]. Finally, the platform allows this performance model to interact with the network model in order to obtain an estimate for the performance of the entire system.

The framework follows the MDA idea in that it includes the middleware impact by semi-automatically transforming a middleware-independent UML model into a middleware-aware model (effectively performing an MDA PIM-to-PSM transformation). The architects do not need to know the full details of the middleware, since those details will be inserted by the transformation tool. This way of modelling would also allow rapid evaluation of the performance of the system with several different middleware technologies, in order to find the one with the best results.

The input to the transformation consists of a high-level, middleware-independent UML model, constructed by the system designers, together with some middleware usage information (deployment information for additional middleware services, execution times of middleware components, etc.). The UML model can be seen as a PIM (where the middleware is considered the “platform”), while the middleware information describes the platform specifics.

The transformation output is a more detailed UML model (a PSM) of the system, containing all the necessary details of the middleware, both architectural and performance-related.

The framework consists of a transformation algorithm and a library of middleware descriptions, each containing the middleware-specific part of the transformation for that type of middleware. The middleware library gives designers the opportunity to rapidly model and compare the system using different types of middleware, without having to delve into the internals of all those different middleware types.

IV. INCORPORATING NETWORK PERFORMANCE

As mentioned earlier, most modelling languages do not allow accurate modelling of both the network and the software architecture. Therefore, a hybrid solution was developed, allowing LQNS (an analytic layered queuing network solver) and NS (a network simulator) to co-operate. The LQN model is a general-purpose software performance model of the system, in which the network is represented as a simple delay, or rather a delay per message type sent over the network. The NS model, on the other hand, contains a detailed network model, including the network layout, routing algorithms, communication protocols, etc.

A framework has been developed to implement this hybrid model solving. The framework solves both models iteratively, using the results obtained from one model to refine the other, until a stable and accurate model is reached, from which performance estimates for the entire system can be obtained.

V. CONCLUSION

The framework presented in this paper allows modular and semi-automatic modelling of distributed systems. It automatically includes the impact of using middleware in the system, and allows designers to model the network and the software architecture using the modelling languages best suited for that purpose. Finally, it solves these models using existing tools to obtain performance estimates for the complete system.

REFERENCES


