

# A Methodology for Predicting the Performance of Component-Based Applications

Nicolae Dumitrascu, Sean Murphy, Liam Murphy  
Department of Computer Science  
University College Dublin, Ireland

[nicolae@eeng.dcu.ie](mailto:nicolae@eeng.dcu.ie), [sean.murphy@iname.com](mailto:sean.murphy@iname.com), [liam.murphy@ucd.ie](mailto:liam.murphy@ucd.ie)

## Abstract

*One of the major problems in building large-scale enterprise applications is predicting the performance of the eventual solution before the application has been built. Middleware offered by component technologies such as Sun's Enterprise JavaBeans, Microsoft's .NET, or OMG's CORBA Component Model does not guarantee the fulfillment of performance requirements. These technologies support assembly of components and provide means to connect components together, but they do not provide support for predicting the quality of the assembly or an application built on assemblies. When systems are built using assemblies, an important characteristic in predicting the performance of the system is to predict the performance of a given assembly. We propose a methodology for reasoning about the performance of component based applications. Our methodology is based on creating performance profiles for each component, assembly and connection type, and groups them together in order to predict the performance of the applications. Development of a framework to implement this methodology is in progress, with the current focus on Microsoft .Net technology.*

**Keywords:** .Net assembly, COM+, performance prediction

## 1. Introduction

Component-based Software Engineering [1] is concerned with the development, deployment, and evolution of component-based software systems. Moreover, it focuses on the development of systems from software component assemblies, the development of assemblies, and system maintenance and improvement by means of component assembly customization.

Building software systems from assemblies and building assemblies from components requires established methodologies not only in relation to the development phase, but also to the entire assembly and system lifecycle including relevant quality attributes such as performance, dependability, security, safety, usability. There are many definitions of what a software component is. Most of them are close to the one given by C. Szyperski [2]: *a software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.* Nowadays, the most important software component technologies such as Microsoft Corporation's COM+, Sun Microsystems's Enterprise JavaBeans, and OMG's CORBA Component Model (CCM) support the building of systems from assemblies of components and provide services such as persistence, security, transaction, etc. However, they do not provide support for performance analysis of component configurations.

In this paper we outline a methodology for predicting the performance of component-based applications. Our goal is to achieve a good understanding of performance characteristics of component-assemblies. As a consequence, we can reason about performance attributes of components and assemblies in a uniform way. The development of the propose methodology is in progress, with the current focus on Microsoft .NET technology.

## 2. Related work

There has been a significant amount of research in the area of predicting software systems performance. Different performance models, such as queueing networks and their extensions called extended queueing networks and layered queueing networks, stochastic timed Petri nets, stochastic process algebra and simulation models have been applied more or less successfully to predict performance of new component technologies.

C. Szyperski [2] underlines there is a need for using assemblies and analyzing their properties for building software systems. He states that component assembly is the new word for programming and a good example today is .NET assembly from Microsoft.

M. Larsson et al. [3] describe a methodology for predicting the behavior of a product software system before it is built. They have proposed the use of a prediction-enabled component technology for developing and maintaining a component based product line architecture in the real-time system's domain. They illustrate predictability of assemblies for the specified component model and they take into account two concrete assembly's properties from a real-time product line's point of view: consistent and end-to-end deadline.

J. Stafford and K. Wallnau [4] discuss the affinities among software architecture, software component technology, compositional reasoning, component property measurement and component certification for the purpose of mastering component feature interaction and for developing component technologies that support compositional reasoning, and that guarantee that design-time reasoning assumptions are preserved in deployed component assemblies.

Yet, none of these research works make a clear distinction that component assemblies are relying on infrastructure level components to manage their lifecycle and execution. Moreover, they do not consider attributes specific to component model architecture.

### 3. Background

#### 3.1 .NET Framework

.NET framework [5] is the new environment proposed by Microsoft for building, deploying and running applications. The framework's software development technologies provide a powerful set of application runtime services that target the development of enterprise applications. Applications written for .NET run inside the run-time environment called the *Common Language Runtime* (CLR) and take advantage of the services that CLR offers. These services (i.e. garbage collector, memory management, security, versioning) simplify the process of development and deployment of applications.

#### 3.2 .NET assembly

.NET components are pieces of software, which are independently, deployable in a plug-in fashion, never deployed in isolation, and ready to interoperate with other programs. .NET assemblies are collections of components designed to work together, built into single or multiple files. Moreover, an assembly is the unit of reuse, versioning, security and deployment; it can be seen as logical .dll or .exe files that contains implementation of types (i.e. classes and interfaces), references to other assemblies, resource files, etc. As depicted in Figure 1, every assembly is completely described by Intermediate Language (IL) code, metadata, and manifest.

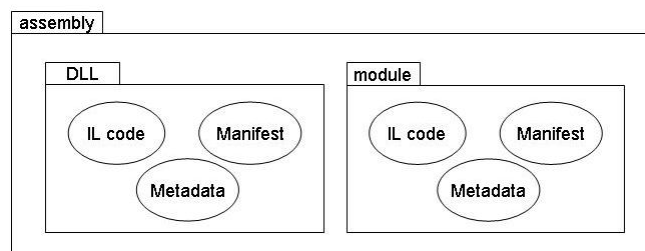


Figure 1 Assembly files

IL is an instruction set that the CLR interprets. *IL code* contains instructions for loading and initializing classes, for calling methods on objects, and for handling logical and arithmetic operations.

*Metadata* provides description of all types declared in the assembly. This information is used by CLR for registration, debugging, memory management, and security.

*Manifest* offers a description of the assembly (i.e. name, version, shared name, list of all files in the assembly) and all other assemblies this assembly references. Thus, the manifest contains the information the CLR needs to load the assembly and to access its types.

### 3.3 COM+ services

COM+ services [6] provide the runtime services needed by .NET components and assemblies to run. These services include:

- Just-In-Time Activation (JITA) – instantiates/discards components,
- object pooling – allows instances of frequently used resources, such as database connections, to be maintained in a pool for reuse by numerous clients,
- transaction – allows operations carried out by distributed components and resources to be treated as a single operation,
- synchronization – controls concurrent access to objects,
- queued components – permit components to communicate through asynchronous messaging and events – services that allow components to inform one another of significant events, such as changes in data or system state.

## 4. Performance prediction methodology

### 4.1 Methodology overview

Component-based applications currently built using .NET technologies are composed of interdependent assemblies. The assemblies and their interconnections vary for each design. We believe that performance profiles of assemblies and connections help in predicting the performance of such applications. Moreover, it aids selecting and adjusting the application software design to meet the performance goals in early phases of development. Furthermore, when the application is deployed and running it assists in tuning software attributes and in capacity planning. Our proposed methodology for predicting the performance of component-based applications is depicted in Figure 2.

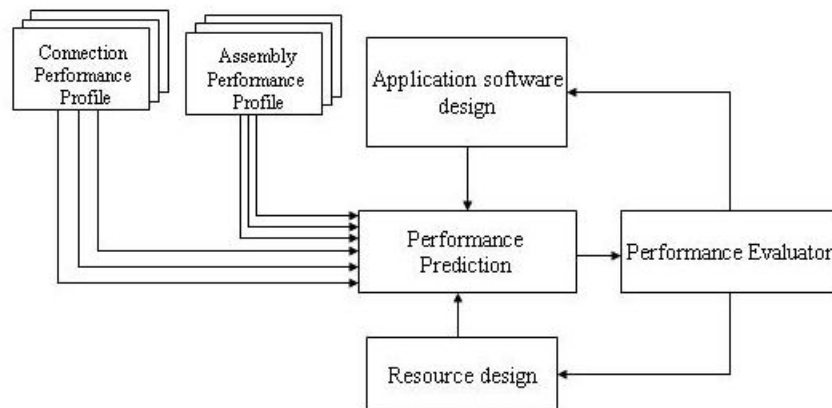


Figure 2 Application performance prediction

The *performance prediction* module predicts the performance of an application taking into account the application software design, the resource design (i.e. hardware configuration), the assembly performance profiles of all contained assemblies in the application, and the connection performance profiles of all connection types used.

The *application software design* comprises the design of the application, the use cases and scenarios that are important from performance point of view. In addition, it includes the workload estimation in terms of client requests, transaction mix, etc. The *resource design* defines the hardware resources (i.e. CPU, memory) used by the application.

*Performance Evaluator* analyses the predicted performance against the required performance. The feedback loop from the performance evaluator informs the application and system designer about the results of the design

evaluation. If the performance goals are not met, either the application design or the resource design has to be changed and a new re-evaluation of the performance could be driven.

#### 4.2 Connection performance profile

Connection performance profile specifies performance attributes for different types of connections between components. Component interaction is dependent on where components are deployed and running (Figure 3). We are investigating the influence of component location (i.e. physical machine, process, apartment, and context) on the component interactions and the delay induced.

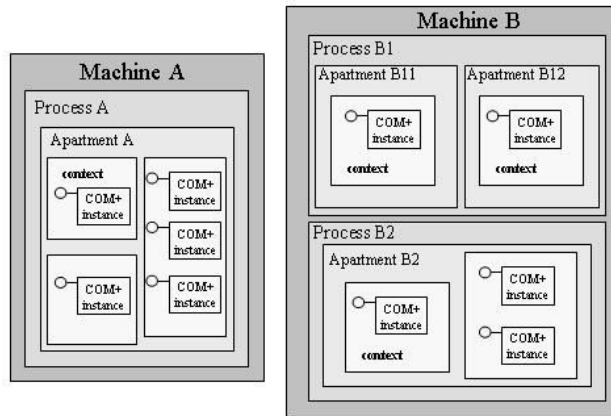


Figure 3 COM+ context

Some of the performance attributes when describing connections are: number of concurrent processes/threads, bandwidth - how much information can be carried in a given time between components, latency - how much time it takes for a packet of data to propagate across the network, etc. To build the connection performance profile we will consider these attributes. In addition we intend to discover new performance attributes for connection types.

#### 4.3 Assembly performance profile

In a divide-and-conquer approach, to be able to analyze the performance of an application, we must first be able to specify the performance attributes of an assembly. To determine the performance profile for an assembly we propose a methodology shown in Figure 4. First, we determine *component performance profiles* for each contained component in an assembly. This profile describes how various component attributes affect the performance of the component. Using this profile, we can analyze the behavior and performance of a component in a generic manner that is not related to any particular assembly requirements.

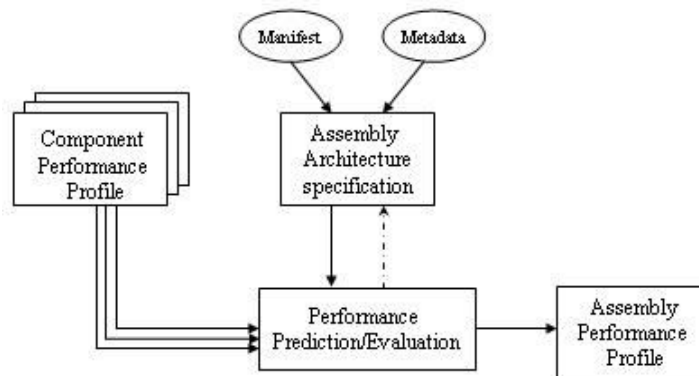


Figure 4 Assembly performance profile

Then we predict the performance of the assembly taking into account the defined component performance profiles and the assembly architecture. The *assembly performance profile* describes the assembly performance in terms of response time/request loads, throughput, memory consumption, etc. Our aim is to build an assembly performance profile that helps in predicting the performance of the assembly in different design configuration. Moreover, this profile provides the assembly architect with insights on how different components interact with each other, and gives him assistance on the effects of his architectural decisions. In addition, introducing these profiles into a generic performance model, it could be possible to predict the assembly configuration settings required to achieve a better overall performance.

#### 4.4 Component performance profile

For the performance profiles for .NET components we investigate the influence of different attributes defined by the component model on the performance of a component. We determine the impact on response time and throughput of these attributes.

There are many attributes defined by the .NET component model such as:

- transaction attributes : enable just in time activation, auto-done attribute used when a transaction is done
- object pooling attributes: MinPoolSize, MaxPoolSize, CreationTimeout
- security attributes
- event attributes: fire in parallel, allow in-process subscribers, publisher ID

The value of some of these attributes influence the performance of a component. We present as example the impact of object pooling attribute MaxPoolSize on the response time. Object pooling is intended to spread the cost of an object's initialization across multiple clients yet not all COM+ components use it. In Figure 5, we depict a scenario for object pooling. When a COM+ application starts up, COM+ instantiates as many components as the minimum pool size requires. As new requests come in, COM+ creates new instances up to the maximum pool size. Once the number of clients exceeds the maximum number of instances the clients are queued until an instance become available. They wait in the queue to acquire an instance until the creation time out value has been reached. Usually component object pooling is used if there is a need to acquire or restrict the number of expensive resources such as database connection or if the component methods perform a granular amount of work relative to the amount of work performed constructing the object. If it is decided the need of object pooling then the optimum level of object pooling attributes should be determined. The optimum level of the object pooling attributes have to take into account the memory consumption, the number of concurrent client requests, the number of database connections.

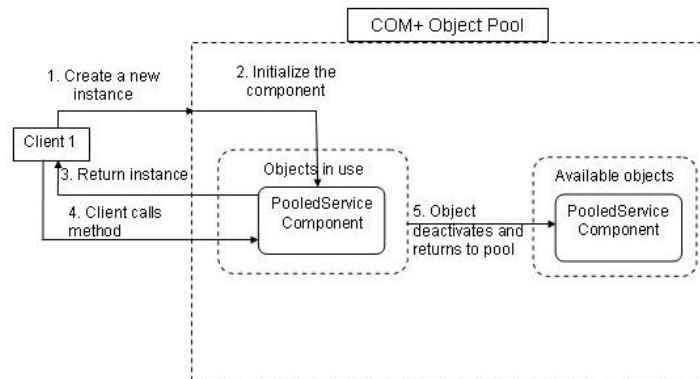


Figure 5 Object pooling scenario

When a client requests a service from a component, the response time is one of the performance parameters the user experience. From a client's perspective, it is the delay experienced between the point when a request is made and the server's response at the client is received. We provide an experiment which shows the influence of the value of component pool size on the response time. For this experiment, we opted for a component that performs a simple insert and select against three different tables of one database. Our experiment has had the following steps. We set the number of maximum value for component pool size to the following values: 30, 60, 90, and 120. For each number of maximum values, we sent client requests having the value between 100 requests and 500 requests. The results obtained are depicted in Figure 6.

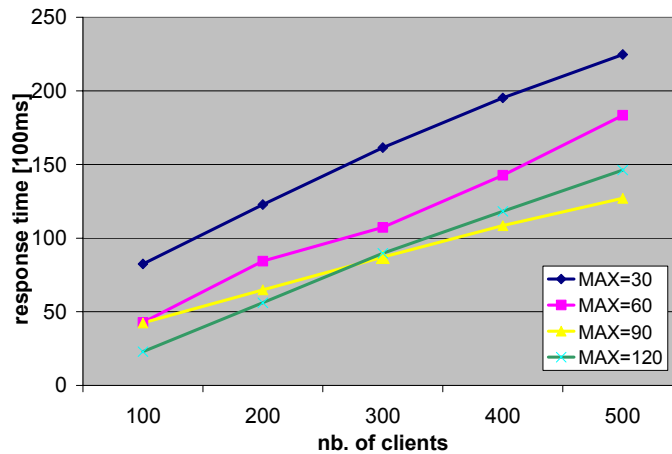


Figure 6 Experimental results

Choosing an optimal pool size is a tradeoff between performance and memory consumption. We can see from the graph that for a maximum pool size equal to 90 we have a good response time for a range of clients between 300 and 500. Using a maximum pool size equal to 120 we have a good response time for a range of clients between 100 and 300.

This simple experiment shows that the object pooling attributes have a great impact on the response time. We will further evaluate the influence of other component attributes on the component's performance and we will draw up an algorithm that determines the optimum range for the attributes values. In addition, we plan to obtain the assembly response time and assembly throughput as performance metrics generated by analytical performance models. We may use Markov models, queuing models, and stochastic process algebra to infer these metrics.

## 5. Conclusions and future work

This paper describes an approach for predicting the performance of component-based applications. We started with presenting a general overview of our methodology. The proposed methodology decouples the application performance prediction problem into a set of constituents (i.e. connection performance, assembly performance and component performance) in a "divide-and-conquer" approach. We introduced the connection performance profile and some performance attributes which should be considered for interconnections between components. Then we outlined the assembly performance profile with its inputs: component performance profile and the architectural specification. Later we presented attributes defined by the component model that influence the performance of the components in the runtime environment. We believe that our methodology allows us to reason about different design assembly configurations selecting the best one from the performance point of view.

Future work includes a rigorous reasoning on how different attributes influence the component and assembly performance. Moreover, a performance model that aids in predicting the performance of an assembly composed by components with known performance profiles is planned.

## Acknowledgement

The support of the Informatics Research Initiative of Enterprise Ireland is gratefully acknowledged.

## References

- [1] I. Crnkovic, *Component-based Software engineering: building systems from software components*, ACM SIGSOFT, Software Engineering Notes, v.27(3), May 2002.
- [2] C. Szyperski, *Component Software – beyond object oriented programming*, second edition, Addison-Wesley Publishers Ltd., 2002.
- [3] M. Larson, A. Wall, C. Norstrom, I. Crnkovic, *Using prediction-enabled technologies for embedded product lines architectures*, Proceedings of the 5<sup>th</sup> ICSE Workshop on Component-Based Software Engineering, Orlando, USA, 2002 .
- [4] J. Stafford and K. Wallnau, *Predictable assembly from certifiable components*, ECOOP 2001.
- [5] Microsoft Corporation, MSDN library: <http://www.msdn.microsoft.com/>
- [6] Juval Lowy, *COM and .NET component services*, O'Reilly & Associates, 2001.