A Framework for Performance Management of Component Based Distributed Applications

Adrian Mos
Performance Engineering Laboratory
Dublin City University, Ireland
+353-1-7007644
mosa@eeng.dcu.ie

ABSTRACT

We propose a framework that can be used during as well as after development, to identify performance problems, suggest corrections and predict performance in large-scale component-based distributed enterprise systems.

1. INTRODUCTION AND PROBLEM

Component Based Middleware platforms such as Enterprise Java Beans (EJB), Microsoft .NET or Corba Component Model (CCM) address the needs of large enterprise projects by providing reusable standardized services and reliable runtime environments which developers can effortlessly integrate and use in their applications. To reduce development costs, developers often use Commercial-Off-The-Shelf (COTS) components or outsource parts of the development effort to third parties. The downturn is that when the resulting application is large, it is difficult for architects and developers to profoundly understand the implications of different design options over the overall performance of the running system. They often make functional assumptions (total system workload or workload distribution), and technological assumptions (best practices for a particular platform, operating characteristics of an application server), which may lead to design decisions that cause severe performance problems after the system has been deployed.

2. PROPOSED SOLUTION

The Component Performance Assurance Solutions (COMPAS) Framework comprises three modules: monitoring, modelling and prediction; they can be used in conjunction or separately, depending on the amount of information available to developers. Developers would use COMPAS to instrument, model and predict the performance of a Target Application (TA), which can be a full, completed enterprise application, or just a functional running subsystem of the enterprise application. Our approach integrates well in development environments that adhere to iterative development processes such as Rational Unified Process or Extreme Programming. Such processes demand that a running version of the application exists at the end of every iteration, making monitoring possible.

The **monitoring module** extracts run-time information from the TA without changing the TA's source code or the run-time infrastructure (i.e. application server's code). Proxy components are automatically deployed in the run-time environment upon introspection of the TA's static deployment structure (e.g. the EAR file for J2EE applications). One proxy component (PXC) is created for each original component (ORC) from the TA. In this way, the TA is completely mirrored by a Proxy Application (PA). Each PXC in the PA will assume the identity (i.e. the name bound to a component in the server's naming directory) of its corresponding ORC from the TA, and at the same time, the ORCs will be given different identities. This process ensures that any client (external client or another ORC) call to an ORC will be intercepted by a corresponding PXC. In addition, container-managed lifecycle events (e.g. creation, destruction, activation) can be intercepted. Upon any interception, the PXCs produce time-stamps, compute different metrics (e.g. method execution response time), notify the monitoring framework and forward the request to the corresponding ORC. A history of the performance parameters associated with an ORC is maintained, which allows the detection of performance degradation (e.g. a method exhibits 10 times increase in its execution time). When performance degradation is detected, an alert is attached to the corresponding PXC, and the monitoring infrastructure is notified.

The information extracted by the monitoring infrastructure is used in the **modelling module** to create UML execution models of the TA. These models are augmented with performance indicators as specified by the *UML Profile for Schedulability, Performance, and Time*. The model extraction process is based on statistical methods and uses dynamic information such as time-stamps and method execution times together with static information from component deployment descriptors such as inter-component dependencies (depending on platform specifications). To facilitate the understanding of the system, the generated models are traversable both horizontally between transactions at the same abstraction level, and vertically between different layers of abstraction using the concepts defined by the Model Driven Architecture [1] (MDA). The MDA approach is useful for managing the complexity of the generated models, and allows a faster identification of performance problems in the TA design.

A logical **feedback loop** exists between the monitoring module and the modelling module and its main purpose is to control and focus the monitoring process by automatically activating and deactivating PXCs in a way that ensures that only the relevant ORCs are monitored at any moment in time. This approach reduces the total overhead of the monitoring infrastructure. After the execution scenarios (logical paths consisting of chains of components) have been identified during the modelling phase, the only active PXCs will be those corresponding to top-level ORCs (the first components in each scenario). If a performance alert is issued by any of the top-level PXCs, COMPAS will activate the remaining PXCs in that

OOPSLA '02, November 4-8, 2002, Seattle, Washington, USA.
ACM 1-58113-626-9/02/0011.

particular scenario, and the alert can be narrowed down to the ORC in the logical path that is responsible for the performance loss perceived in the top-level PXC.

The **performance prediction module** simulates the generated models and developers can specify different workload characteristics such as the number of simultaneous users and their inter-arrival rate. In addition developers can specify expected performance attributes for particular ORCs, which are transformed into conditions for generating alerts during the simulation. We envisage that in the simulation process, developers will be able to change the generated models and observe the effects such changes can have on the overall performance of the application.

In the modelling phase as well as in the prediction phase, developers will visually browse the generated models using a top-down approach and will be able to easily switch between Platform Independent Models (PIMs) and Platform Specific Models (PSMs) as defined by the MDA specification. PIMs can be represented using the Enterprise Collaboration Architecture subset from the Enterprise Distributed Object Computing (EDOC) UML Profile, in order to benefit from a standardized form of representation for business modelling concepts. PSMs can be represented using specialized profiles such as the UML Profile for EJB, which provide means to illustrate technology specific details. When a performance alert is detected in a model, developers can control the level of detail that is presented, and can see if that particular problem is caused by poor design of the business workflow (i.e. tight coupling between what should have been loosely coupled components) or by a wrong technological decision (i.e. in some cases in EJB, implementing a component as a stateful session bean can lead to lower performance than if implemented as a stateless session bean).

COMPAS will use different technological profiles corresponding to particular platforms such as EJB or .NET. Such profiles will contain known performance issues and patterns such as [2] for the platforms they represent and will facilitate the detection of wrong technological decisions or anti-patterns in that context. For example, an EJB PSM can show a performance alert when an entity bean finder method returns a large result set. In such a situation, the COMPAS may suggest a pattern such as Value List Handler [2] to alleviate the performance problem.

One of the most significant methods for performance modelling and prediction is presented in [3] reporting significant results in the improvement of the software development process, specifically the use of Software Performance Engineering methods aided by related tools such as SPE-ED. These tools assume that developers can map application entities such as objects or methods to run-time entities such as I/O utilization, CPU cycles or network characteristics. It has been proved that such techniques and tools like SPE-ED help in achieving performance goals and reducing performance related risks for general object-oriented systems and even for distributed systems. However, we argue that middleware such as EJB or other component-oriented platforms, exhibit an inherent complexity which developers find hard if not impossible to quantify even in simple models. Automated services such as caching, pooling, replication, clustering, persistence or Java Virtual Machine optimisations, provided by EJB application servers, for example, contribute to an improved and at the same time highly unpredictable run-time environment. In contrast, COMPAS extracts simplified performance information such as method execution times from running versions of the target application, and creates UML performance models automatically. This approach eliminates the need for assumptions and can offer more accurate models and predictions.

The Form framework [4] automatically generates execution profiles from Java applications, being partially similar in intent to the COMPAS Modelling module. Form uses JVM instrumentation to intercept object level events used to build UML sequence diagrams showing the captured interactions and is not particularly performance oriented. COMPAS however, uses a non-intrusive approach, deploying a parallel PA, and is strongly focused on the *performance* of *component-based* systems, taking into consideration specific factors such as component types and lifecycle events in representing the MDA models.

3. STATUS AND FUTURE WORK

A proof-of-concept monitoring module for the EJB technology has been implemented which can automatically generate proxy components for any TA. It uses basic graphical consoles that can show real-time graphs of performance metrics for such events as method invocations and creation and destruction of EJBs. Work is under way to implement the model generator and different UML model representation alternatives are being considered. The separation of abstraction levels and the classification of models in PIMs and PSMs are of particular interest. In addition, we are considering different alternatives for realising the adaptive monitoring, such as introducing more logic in the PXCs to facilitate a certain degree of self-control and automatic organisation. Finally, we are evaluating visual modelling tools such as ArgoUML, and simulation packages, in order to be integrated in the framework. The COMPAS framework is described in more detail and examples of use are presented in [5].

4. ACKNOWLEDGMENTS

This work is funded by Enterprise Ireland Informatics Research Initiative 2001 under grant ATRP/01/220, and supported by Iona Technologies and Sun Microsystems Ireland.

5. REFERENCES

[1] Object Management Group, *Model Driven Architecture*, OMG document number ormsc/2001-07-01, OMG, 2001

[2] J. Crupi, D. Alur, D. Malks. *Core J2EE Patterns*, Prentice Hall, 30 September 2001.

[3] C.U. Smith, L.G. Williams. Performance and Scalability of Distributed Software Architectures: An SPE Approach, *Parallel and Distributed Computing Practices*, 2002

[4] T. Souder, S. Mancoridis, M. Salahm. Form: A Framework for Creating Views of Program Executions. In *Proceedings of IEEE International Conference on Software Maintenance ICSM'01*, Florence, Italy, November 2001

[5] A. Mos, J. Murphy. Performance Management in Component-Oriented Systems Using a Model Driven Architecture™ Approach. In *Proceedings of IEEE International Enterprise Distributed Object Computing Conference EDOC*, Lausanne, Switzerland, September 2002