Software Performance Engineering for Object-Oriented Systems: A Use Case Approach

Sanket Jain (12BCE081)
Shubham Rathi (12BCE097)

Abstract— ***Many object-oriented systems fail to meet performance objectives when they are initially constructed. These performance failures result in damaged customer relations, lost productivity for users, lost revenue, cost overruns due to tuning or redesign, and missed market windows. Most performance failures are due to a lack of consideration of performance issues early in the development process. However, early consideration of performance in object-oriented systems is straightforward. The Use Case scenarios produced by developers during analysis and design serve as a starting point for performance analysis. This paper describes a systematic approach to the performance engineering of object-oriented systems based on Use Case scenarios. This approach is cost-effective and has a low impact on the software development process. A simple case study illustrates the process.***

Keywords- *Software performance engineering (SPE), object-oriented, software, use case, performance, analysis, design.*

I. INTRODUCTION

Object-oriented techniques have become widely accepted for designing and implementing software systems in application areas ranging from client-server to real-time, embedded systems. Object-oriented software systems are typically easier to understand, easier to adapt to new requirements, and have a higher potential for reuse than those developed with procedural approaches.

Our experience is that most performance failures are due to a lack of consideration of performance issues early in the development process, in the architectural design phase. Poor performance is more often the result of problems in the design rather than the implementation.

Performance myth: "Ignore efficiency through most of the development cycle. Tune performance once the program is running correctly and the design reflects your best understanding of how the code should be structured. The needed changes will be limited in scope or will illuminate opportunities for better design."

In this paper we focus on Use Cases since they provide the basis for a bridge between object-oriented methods and SPE (Smith and Williams, 1997). An instance of a Use Case represents a particular execution of the system. Use Case instances are described using Scenarios. Scenarios from Use Cases are translated into SPE performance scenarios. Performance scenarios are, in turn, used to construct and evaluate a variety of performance models.

II. RELATED WORK

In this paper we focus on Use Cases since they provide the basis for a bridge between object-oriented methods and SPE (Smith and Williams, 1997). An instance of a Use Case represents a particular execution of the system. Use Case instances are described using Scenarios. Scenarios from Use Cases are translated into SPE performance scenarios. Performance scenarios are, in turn, used to construct and evaluate a variety of performance models. Even then, the approach tends to be very general and ad hoc. There is no attempt to integrate performance engineering into the development process.

Some work specifically targeted at object-oriented systems has emerged from the performance community. Smith and Williams (Smith and Williams, 1993) describe performance engineering of an object-oriented design for a real-time system. However, this approach applies general SPE techniques and only addresses the specific problems of object-oriented systems in an ad hoc way.

Baldassari et. al. describe an integrated object-oriented CASE tool for software design that includes a simulation capability for performance assessment (Baldassari, et al., 1989), (Baldassari and Bruno, 1988). The CASE tool uses petri nets for the design description language rather than the general methods described above, thus the design specification and the performance model are equivalent and no translation is necessary.

III. OBJECT -ORIENTED DEVELOPMENT

Object-oriented development is an approach to software specification, design and construction that is based on identification of the objects that occur naturally in the application and implementation domains.

A number of approaches to object-oriented analysis and/or design have appeared over the past several years.

Analysis

- Investigating the Boundaries of a Problem
- What are the Scope and Requirements?
- How is the System Accessed?
- Who needs Access to What When?
- Determining WHAT needs to be done!

OO Analysis

- Identification of Critical Concepts in the Problem Domain that Correspond
- Emphasis on Finding Objects and Components
- What is Available to Facilitate OO Analysis?

Design

- Development of a Logical Solution
- Represents One Way to Solve Problem
- Defining HOW System Fulfills WHAT!

OO Design

- Emphasis on Defining Logical Software Objects and Components
- Evaluate Alternative OO Designs
- Leads to Implementation of a Feasible Solution

A number of approaches to object-oriented analysis and/or design have appeared over the past several years. Static models (Class and Object Diagrams) describe the classes and objects that are relevant to the problem and the relationships among them. Dynamic models (State Diagrams) describe the patterns of behavior that apply to objects belonging to a given class. A number of methods have also adopted Use Case Diagrams to describe interactions between the system and its environment or between objects within the system.

Use Case instances are described using Scenarios. A Scenario is a sequence of actions describing the interactions between the system and its environment (including the user) or between the internal objects involved in a particular execution of the system. In object-oriented methods, scenarios are used to:

- describe the externally visible behavior of the system,
- involve users in the requirements analysis process,
- support prototyping,
- help validate the requirements specification,
- understand interactions between objects, and
- Support requirements-based testing.

Figure 1 illustrates a high-level MSC for a simple automated teller machine (ATM).
Each object that participates in the scenario is represented by a vertical line or axis.

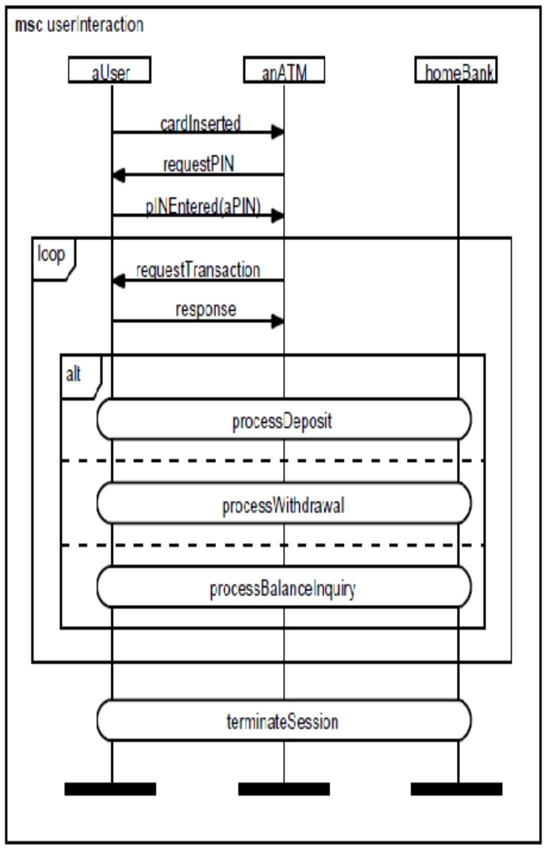


Figure 1. Message Sequence Chart for a user interaction with the ATM

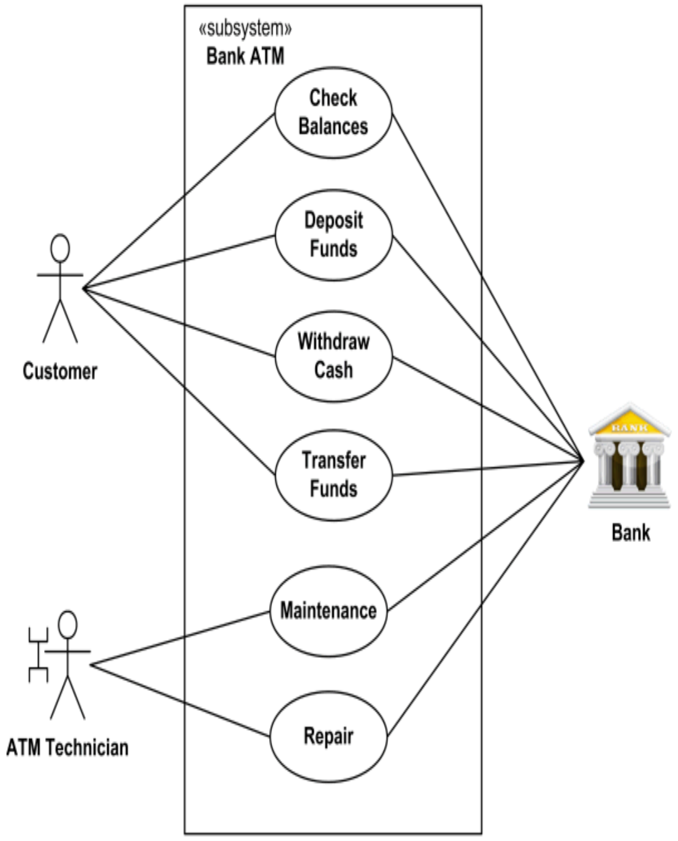
IV. USE CASE APPROACH

Software Use cases are scenarios for understanding system requirements .A use case is an interaction between users and a system. The use case model captures the goal of the user and the responsibility of the system to its users.
In the requirement analysis, the use case is described as one of the following:

- Non-formal text with no clear flow of events.
- Text, easy to read but with a clear flow of events to follow (this is a recommended style).
- Formal style using pseudo code.

The use case description must contain:

- How and when the use case begins and ends.
- The interaction between the use case and its actors, including when the interaction occurs and what is exchanged.
- How and when the use case will need data stored in the system or will store data in the system.
- Exceptions to the flow of events.
- How and when concepts of the problem domain are handled.



Every single use case should describe one main flow of events. An exceptional or additional flow of events could be added. The exceptional or additional flow of events could be added. The exceptional use case extends another use case to include the additional one. The use case model employs extends and uses relationships.

- The extends relationship is used when you have one use case that is similar to another use case but does a bit more .It extends the functionality of the original use case.
- The uses relationship senses common behavior in different use cases.

Use cases could be viewed as concrete or abstract.

- An abstract use case is not complete and has no actors that initiate it but is used by another use case .This inheritance could be in several levels.
- Abstract use cases are the ones that have uses or extend relationships.

V. SOFTWARE PERFORMANCE ENGINEERING

Software performance engineering is a quantitative approach to constructing software systems that meet performance objectives. SPE prescribes principles for creating responsive software, the data required for evaluation, procedures for obtaining performance specifications, and guidelines for the types of evaluation to be conducted - 7 - at each development stage. It incorporates models for representing and predicting performance as well as a set of analysis methods.

Two types of models provide the design assessment:

A. *Software Execution Model:*

The software execution model represents key aspects of the software execution behavior. It is constructed using execution graphs (Smith, 1990) to represent workload scenarios.

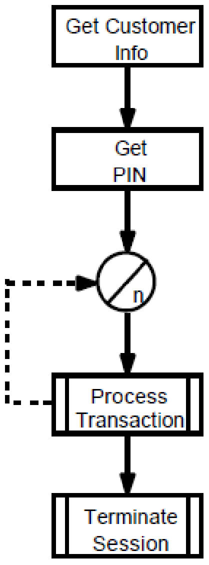
Solving the software model provides:

- A static analysis of the mean, best- and worst case response times.
- It characterizes the resource requirements of the proposed software alone, in the absence of other workloads, multiple users or delays due to contention for resources.

B. *System Execution Model:*

If the software execution model indicates that there are no problems, analysts proceed to construct and solve the system execution model. This model is a dynamic model that characterizes the software performance in the presence of factors, such as other workloads or multiple users that could cause contention for resources. The results obtained by solving the software execution model provide input parameters for the system execution model. Solving the system execution model provides the following additional information:

- More precise metrics that account for resource contention.
- Sensitivity of performance metrics to variations in workload composition.
- Effect of new software on service level objectives of other systems.
- Identification of bottleneck resources.
- Comparative data on options for improving performance via: workload changes, software changes, hardware upgrades, and various combinations of each.



Execution Graph for user interaction with the ATM

VI. SPE for OOD

1. *Establish performance objectives*:
   Performance objectives specify the quantitative criteria for evaluating the performance characteristics of the system under development. These objectives may be expressed in several different ways, including: response time, throughput, or constraints on resource usage. For information systems, response time is typically described from a user perspective, i.e., the number of seconds required to respond to a user request. For real-time systems, response time is given as the amount of time required to respond to a given external event. Throughput requirements are specified as the number of transactions or events to be processed per unit time.

2. *Identify important Use Cases*:
   The important Use Cases are those that are critical to the operation of the system or which are important to responsiveness as seen by the user. Typically, this is only a subset of the Use Cases that are identified during object-oriented analysis.

3. *Select key performance scenarios*:
   It is unlikely that all of the scenarios for each critical Use Case will be important from a performance perspective. For each important Use Case, the key scenarios are those which are executed frequently or those which are critical to the perceived performance of the system.

4. *Translate scenarios to execution graphs*:
   Once the key performance scenarios have been identified, the MSC representation is translated to an execution graph. Currently, this is a manual process. However, the close - 10 - correspondence between scenarios as expressed in MSCs and execution graphs suggests that an automated translation may be possible.
   Estimates of the amount of processing required for each step in the execution graph are obtained from the class definition for each object involved. This information is contained in the class diagram, or the logical view of the system architecture (Kruchten, 1995). As described above, early in the development process, these may be simply best/worst case estimates. Later, as each class is elaborated, the estimates become more precise.

5. *Add resource requirements*:

The processing steps in an execution graph are typically described in terms of the software resources (e.g., operating systems calls or database accesses) used. Resource requirements map these software resource requirements onto the amount of service they require from key devices in the hardware configuration.

Resource requirements depend on the environment in which the software executes. Information about the environment is obtained from the physical view of the architecture (Kruchten, 1995). In the UML, this corresponds to the Deployment Diagram.

6. *Solve the models*:

As noted above, solving the execution graph characterizes the resource requirements of the proposed software alone. If this solution indicates problems, analysts consider design alternatives to address the problems. If not, then analysts proceed to solve the system execution model.

VII. SUMMARY & CONCLUSION

This paper has described a systematic approach to the performance engineering of object-oriented software that is critical to preventing performance failures. Performance failures may result in damaged customer relations, lost productivity for users, lost revenue, cost overruns due to tuning or redesign, missed market windows, and, in the worst case, the need to completely re-design the product or even cancel the project.

Object-oriented systems offer unique challenges for performance engineering due to the complexity of interactions between objects and the trend toward distribution of objects over a network. However, our experience has shown that it is possible to *cost-effectively* engineer object-oriented systems that meet performance goals. This paper has described the process of software performance engineering for object-oriented systems and illustrated that process with a simple case study.

The key to this process is the employment of Use Case scenarios as a link between object-oriented analysis and design models and performance models. Use case scenarios can be translated to execution graphs which serve as a starting point for performance modeling. This connection is a key step in enabling the performance evaluation of new object-oriented software systems. We illustrated the connection with a simple best-case *software* execution model. The software execution model is sufficient for many architecture and design evaluations.

This work is part of a larger project to make it easier for developers to perform initial performance assessments. One of the principal barriers to the widespread acceptance of SPE is the gap between the software developers who need performance assessments and the performance specialists who have the skill to conduct comprehensive performance engineering studies with today's modeling tools. Thus, extra time and effort are required to coordinate the design formulation and the design analysis. This limits the ability of developers to explore design alternatives. Our other future work will address models of distributed object-oriented systems, the specification of performance requirements, and additional tool features to automate SPE evaluations.

REFERENCES

[1] Auer, K. and K. Beck, Lazy Optimization: Patterns for Efficient Smalltalk Programming, in *Pattern Languages of Program Design, Volume 2*, (J. Vlissides, et al., ed.), Addison-Wesley, 1996.

[2] Baldassari, M., et al., PROTOB: A Hierarchical Object-Oriented CASE Tool for Distributed Systems, *Proceedings of the European Software Engineering Conference, 1989*, Coventry, England, (1989).

[3] Baldassari, M. and G. Bruno, An Environment for Object-Oriented Conceptual Programming Based on PROT Nets, in *Advances in Petri Nets, Lectures in Computer Science No. 340*, ed.), Springer-Verlag, Berlin, 1988.

[4]. Beilner, H., et al., Towards a Performance Modeling Environment: News on HIT, *Proceedings of the 4th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*, Plenum Publishing, (1988).

[5]. Beilner, H., et al., The Hierarchical Evaluation Tool HIT, in *Performance Tools and Model Interchange Formats*, (F. Bause and H. Beilner, ed.), Universität Dortmund, Fachbereich Informatik, Dortmund, Germany, 1995.

[6]. Booch, G., *Object-Oriented Analysis and Design with Applications*, Benjamin/Cummings, Redwood City, CA, 1994. Booch, G., *Object-Oriented Design with Applications*, Benjamin/Cummings, Redwood City, CA, 1991.

[7]. Buhr, R. J. A. and R. S. Casselman, *Use Case Maps for Object-Oriented Systems*, Prentice Hall, Upper Saddle River, NJ, 1996.

[8]. Buhr, R. J. A. and R. S. Casselman, Timethread-Role Maps for Object-Oriented Design of Real-Time and Distributed Systems, *Proceedings of OOPSLA '94: Object-Oriented Programming Systems, Languages and Applications*, Portland, OR, 301-316 (1994).

[9]. Buhr, R. J. A. and R. S. Casselman, Architectures with Pictures, *Proceedings of OOPSLA '92: Object-Oriented Programming Systems, Languages and Applications*, Vancouver, BC, 466-483 (1992). Gamma, E., et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, MA, 1995.

[10]. Goettge, R. T., An Expert System for Performance Engineering of Time-Critical Software, *Porceedings of the Computer Measurement Group Conference*, Orlando, FL, 313- 320 (1990).

[11]. Grummitt, A., A Performance Engineer's View of Systems Development and Trials, *Proceedings of the Computer Measurement Group Conference*, Nashville, TN, 455-463 (1991). - 24 - Hrischuk, C., et al., Automatic Generation of a Software Performance Model Using an Object-Oriented Prototype, *Proceedings of the Third International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, Durham, NC, 399- 409 (1995).

[12]. ITU, Criteria for the Use and Applicability of Formal Description Techniques, Message Sequence Chart (MSC), International Telecommunication Union, 1996.

[13]. Jacobson, I., et al., *Object-Oriented Software Engineering*, Addison-Wesley, Reading, MA, 1992.

[14]. Rumbaugh, J., et al., *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[15]. Selic, B., et al., *Real-Time Object-Oriented Modeling*, John Wiley and Sons, New York, 1994.

[16]. Shlaer, S. and S. J. Mellor, *Object Lifecycles: Modeling the World in States*, Yourdon Press, Englewood Cliffs, NJ, 1992.

[17]. Shlaer, S. and S. J. Mellor, *Object-Oriented Systems Analysis: Modeling the World in Data*, Yourdon Press, Englewood Cliffs, NJ, 1988.