Confirming and Reconfirming Architectural Decisions on Scalability: A Goal-Driven Simulation Approach

Tom Hill, Sam Supakkul, and Lawrence Chung

Department of Computer Science, The University of Texas at Dallas,
800 West Campbell Road Richardson, Texas 75080-3021, USA
tom.hill@eds.com, ssupakkul@ieee.org, chung@utdallas.edu

Abstract. Scalability, which refers to an ability to support increased loads with acceptable performance, is among the key issues in deciding on an architecture with its essential components, together with relationships between such components, as well as constraints on such components and relationships. As with just about any design, the architectural design space is potentially huge, if not infinite, while the quality of the final system to be implemented inevitably depends largely on various decisions made during the architectural design phase. Unfortunately, however, it often times seems difficult to analyze if an architectural design incorporates good decisions or even bad ones, since an architectural design is (supposed to stay) at a high-level of abstraction and not concrete enough on its performance and scalability behavior, before we commit to the time-consuming and costly lower level design, implementation and testing. In this paper, we propose an integration of goal-orientation, which is qualitative in nature, and simulation, which is quantitative in nature.

Keywords: performance, scalability, architecture, design decision, NFR framework, goal-oriented, simulation-based, softgoal interdependency graph.

1 Introduction

Scalability, which refers to an ability to support increased workloads with adequate service levels or performance [1], is among the key issues in deciding on an architecture. It is costly to retrofit an architecture after the fact if it is found not to provide the expected scalability.

The classic nineteen seventies German text 'Engineering Design', established the seven steps of the systems approach used today in the design activity in most design domains, including software design:(State Analysis, Goal Setting, Solution Variants Development, Analysis, Evaluation, Decision, and Implementation)[2]. The focus of this paper is on two of the architectural design activities: goal setting and solution variants development. The primary goals of the modern software architectural design activity are to create and communicate a lasting rational design. The design is a structured model of a future software implementation. Architecture, as a profession, is about demystifying the underlying structure of objects (buildings, bridges, spacecrafts, computers and software). A system's structure is defined by essential components, interactions between components and constraints attached to both.

R. Meersman, P. Herrero, and T. Dillon (Eds.): OTM 2009 Workshops, LNCS 5872, pp. 327–336, 2009.
© Springer-Verlag Berlin Heidelberg 2009

The primary contribution of this paper is a methodology which integrates goal-orientation and simulation to analyze the design decisions on scalability and other Non-Functional Requirements (NFRs) such as cost and reliability related architectural issues to qualitatively explore and confirm architectural decisions, and either re-confirm the validity of such decisions or make corrections to them. Additionally, we propose the use of simulation to reconfirm early architecture design decisions and build confidence in the designer's ability to predict the behavior of future system implementations. Confidence is reinforced by the observation of system behaviors, model consistency, model completeness, and the discovery of new system components [3]. "We believe what we see," and simulation affords us the opportunity to quickly see into the future of the system under study.

For this work, we adopt ideas from Goal-Oriented Requirements Engineering (GORE) (see, for example, [4] [5] [6] [7] for more details) and borrow general techniques from simulation (see, for example, [8] [9] [10] [11] for more details). While each of these two areas addresses either goal-orientation or simulation, our distinctives lie in the integration of the work from these two areas of research and practice.

Section 2 describes an overview of the Health Reimbursement System (HRS) we use for the purpose of illustration and also for our own case study. Section 3 describes the three main steps in our methodology. At the end, we summarize the paper, along with future directions.

2 Application Domain Overview: Health Reimbursement System

The paper focuses throughout on a case study detailing one of the world's largest health care systems to ensure the models are realistic and applicable to solve problems in the real world (please note that the actual numbers used in examples and models have been obfuscated to protect proprietary information). Figure 1 shows the context of the case study for a Healthcare Reimbursements System (HRS).

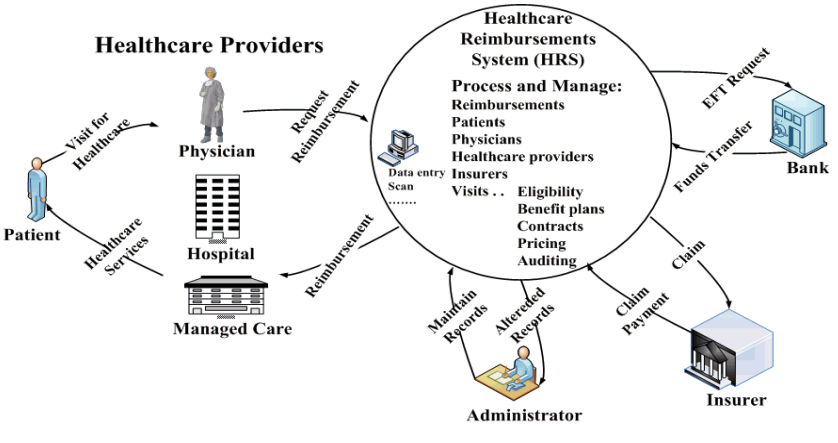

Fig. 1. The Healthcare Reimbursements System (HRS) case study context

Quickly tracing through the simplified context diagram above, the patient visits a healthcare provider (physician, hospital, or managed care). Care is given and the provider submits a request for reimbursement to the HRS. The HRS, via data entry/conversion, re-formats the data and verifies the validity of the following: coding; insured; provider; insurer; eligibility; plan; contract and; pricing. The HRS sends a claim to the insurer and receives an authorization to pay. In the final step, funds are transferred by a bank and the reimbursement is sent to the healthcare provider.

2.1 HRS Stakeholder Non-functional Requirements

Table 1 shows a set of informal non-functional requirements labeled by the primary stakeholder of a large health care system as system architecture principles. The architecture principles (NFRs) are presented in priority order. The HRS case study will examine the highest priority NFRs, performance and scalability, in detail.

Table 1. HRS stakeholder-provided initial architecture principles, non-functional requirements

System architecture principle	Architecture description (in priority order)
Performance and Scalability:	The ability for the solution to meet all service level and performance requirements in a cost-effective manner while providing the ability to expand to support future growth.
Other:	Reliability, Flexibility, Isolation, Data Independence, Business Process & IT Convergence, Open Architectural Foundation, Granularity

Given Table 1, the experienced healthcare-domain requirements engineer can begin to quickly understand user requirements and obtain agreement on a more formal description of the requirements. The production of a requirements scenario is the first step required to reduce the infinite design space to a more manageable performance and scalability subset.

2.2 HRS Architecture Design Decisions

The HRS architect is faced with a task to ensure that the HRS be able to support the initial performance requirements of 150,000 reimbursements per hour and be able to scale to support the estimated future performance of 300,000 per hour. In both cases, the corresponding batch processing must be completed within the batch cycle window of 8 hours. This scenario creates several looming architectural design decisions based on non-functional requirements: Which optimizing technique should be used to satisfy the response time constraints? Which method should be selected to guarantee that the batch cycle will complete within the time window requirements? How can the system perform and remain scalable?

3 A Goal-Driven Simulation Approach to Design Decisions

In this section, we present the three main steps of our methodology: Section 3.1 - Non-functional Requirements Goal Setting; Section 3.2 - Constructing a Key-decision Simulation Model; and, Section 3.3 - Executing and Analyzing Simulation Model Experiments.

3.1 Non-functional Requirements Goal Setting

In this step, informally elicited non-functional requirements that may impact on architectural decisions are analyzed and represented as softgoals to be achieved by the system. For instance, user defined requirements in Table 1 and the quantitative performance requirements described in Section 2 can be mapped to softgoals in a Softgoal Interdependency Graph (SIG) [5] as shown in Figure 2.

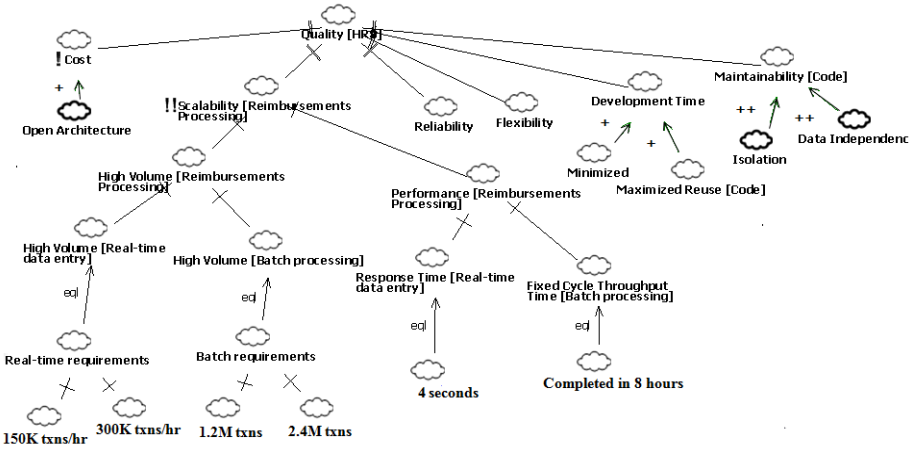

Fig. 2. A derived goal model constructed from the stakeholder requirements in Table 1

In Figure 2, quality of the system is refined using an AND-decomposition to cost, scalability, reliability, flexibility, development time, and maintainability softgoals (denoted by cloud icons), with scalability having the highest priority (very critical as denoted by !!) and cost having medium priority (critical as denoted by !). The scalability is further AND-decomposed to high volume of reimbursement processing and performance of reimbursement processing Softgoals, and subsequent other sub-goals. These softgoals may be refined to more concrete softgoals that specify quantitative requirements such as "4 seconds/transaction" for response time requirement and "completed in 8 hours" for fixed cycle throughput time. The “eql” symbol identifies an equal contribution from the two sub-goals real-time requirements and batch requirements.

During this step, goal-oriented analysis is used to explore and confirm architectural decisions by treating conceptual softgoals in a SIG as softgoals to be achieved. Figure 3 is a refined SIG based on the SIG in Figure 2, that focuses on the scalability softgoal.

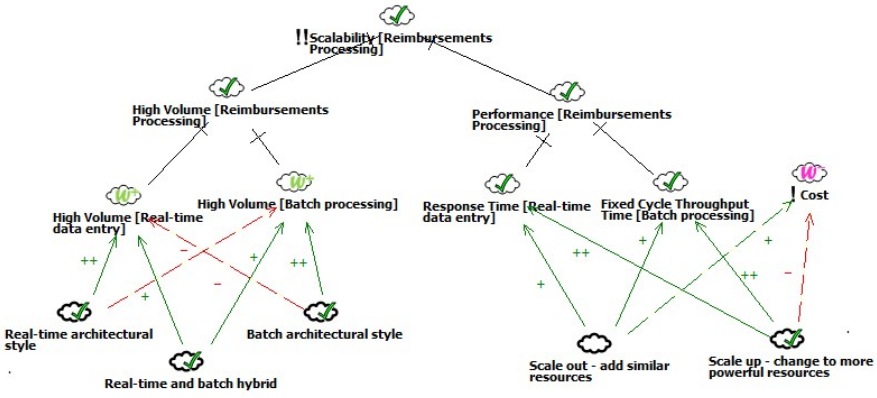

Fig. 3. Initial Softgoal Interdependency Graph (SIG) showing architectural decisions using goal-oriented analysis

To achieve the performance aspect of the scalability, two common techniques to implement scalability are explored, including scaling out or adding similar resources to the processing resources and scaling up or replacing resources with more powerful resources. Scale-up is known to provide better response time and fixed cycle throughput because it minimizes the effort required in design, implementation, and operation, therefore selected for the system (denoted by a check mark). It is also found to be more costly than the scale out alternative (denoted by a red-dash line labeled with Hurt/-- toward cost. On the other hand, scale out alternative is decided to be less desirable - Help/--.

In exploring alternatives for high volume real-time data entry, we find real-time architectural style is a highly desirable alternative (denoted by Make/++), however with a negative side-effect on high volume of batch processing (denoted by a dash-line labeled with Hurt/-). Conversely, batch architectural style is highly positive for achieving high volume of batch processing - Make/++, but has a negative side-effect on high volume of real-time data entry - Hurt/-. Since the HRS needs to support high volume workload in both real-time and batch processing both, the architects agree to use both, as well as real-time and batch hybrid solution that has a positive contribution toward both softgoals.

The solutions that are marked with a check mark represent initial architectural decisions based on a qualitative evaluation.

For the performance sub-goals of response time and fixed cycle throughput time, we examined the operationalization methods of real-time processing, batch processing, and a hybrid data collection to batch processing.

When we drew our first high-level softgoal interdependency graph, figure 3, we were quickly able to observe the major design dilemmas, their relationships and some tradeoffs.

3.2 Construct a Key-Decision Simulation Model

The architectural decision SIG, figure 3, is used to develop simulation models, such as the one shown in Figure 4.

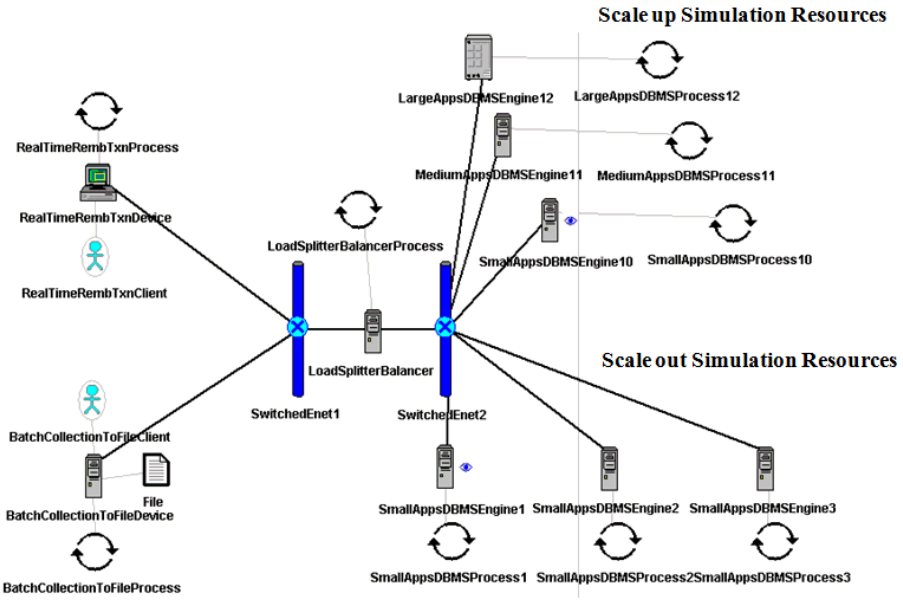

Fig. 4. HRS case study design decision simulation model topology diagram

Simulation modeling is designed to answer the questions posed in section 2: Which optimizing technique should be used to satisfy the response time constraints? Which method should be selected to guarantee that the batch cycle will complete within the time window requirements? How can the system perform and remain scalable? These questions can only be answered by understanding the behavior of a system over time.

The first step in constructing a simulation model is to understand the questions that the model must answer. The second step is to build a topological model to represent the infrastructure, resources, workload, and work flow.

Like the softgoal interdependency graph, the topological model is another observable abstract representation of the HRS. However the HRS simulation model abstraction is capable of representing the behavior of the system over time [8]. Figure 4, shows the topology of the model created for the HRS case study.

The creation of the HRS topology, figure 4, is basically a re-write activity, taking the softgoal interdependency graph, figure 3, as input and re-writing it into a simulation model configuration using the graphical editor provided by the HyPerformix Simulation System [8].

3.3 Execute and Analyze Simulation Experiments

The next step in building a simulation model, is to code experiments to answer the HRS case study questions by modifying the variables: workload (from 150,000 reimbursements per hour to 300,000); number of resources (servers); power of resources (server type); and the selection of reporting variables (elapsed time, number of transactions completed, response time average, throughput time, and utilization of

Table 2. HRS case study simulation experiments and results summary

Simulation experiment	Volume variable	Resource variable	Simulation result	Server utilization
1. Response time (4 secs)	150,000 Txns per hour	App DBMS Base(4 cpu) server*1	Average response -	100% - Fail
2.		App DBMS Base 2 servers	Average response - 5.18	100% - Fail
3.		App DBMS Base 4 servers	Average response - 3.58	Constant - Pass
4. Cycle thruput (8 hours)	1,200,000 input Txns	App DBMS Base 10 servers	Cycle hours < 8	Constant – Pass
5. Response time (4 secs)	300,000 Txns per hour	App DBMS Base 4 servers	Average response -	100% - Fail
6.		App DBMS Base 10 servers	Average response 3. 88	Constant - Pass
7. Cycle thruput (8 hours)	2,400,000 input Txns	App DBMS Base 24 servers	Cycle hours < 8	Constant – Pass
8. Response time (4secs)	150,000 Txns per hour	App DBMS Base (16 cpu) server	Average response - 4.18	Constant – Pass
9. Cycle thruput (8 hours)	1,200,000 input Txns	App DBMS Large server*2	Computed Cycle hours < 8	
10. Response time (4 secs)	300,000 Txns per hour	App DBMS Large server*2	Computed Average response -no	
11. Cycle thruput (8 hours)	2,400,000 input Txns	App DBMS Large server*2	Computed Cycle hours- no	

NOTES: *1 = App DBMS Base = Server Low CPU power, 4 Intel Itanium2 DC 9050 - 1.6 GHz , TPC-C (372K), cost ($ 673K)
*2 = App DBMS Large = Server High CPU power, 64 Intel Itanium2 - 1.6 GHz, TPC-C (4,092K), cost ($ 11,978K) performance characteristics unavailable in simulator - Transaction Processing Performance Council [TPC]computed calculations used to estimate – actual application benchmark required for 10 and 11 above [www.tpc.org]

resources). The functionality to report these variables is built-in to most commercial simulation systems, including the HyPerformix Simulation System.

The following serves as a detailed audit-trail describing each simulation experiment 1 through 11 in table 2:

1. Experiment to verify that a base-level server (low-CPU-power, 4 Intel Itanium2 DC 9050 - 1.6 GHz) can provide four second response time at 150,000 transactions per hour. The simulation failed when it determined that 100% of the server would be needed.
2. Scale-out experiment to find if two base-level servers would provide the necessary resources. The simulation gave 5.18 second response time for a short period of time, then failed after determining 100% of the server would be needed.
3. A scale-out experiment to discover if four servers would provide four second response time for 150,000 transactions per hour. The simulation settled down to a constant service with an average response time of 3.58 seconds.
4. An additional scale-out experiment to determine if ten base-level servers would process 1,200,000 batch transactions within the required eight-hour window (the batch processes carry out the bulk of HRS processing in the system [approximately twice the process load of the real-time processes]). The simulation settled to a constant processing rate to produce the desired effect.
5. The scale-out experiment to determine if four servers would provide enough resources to double the real time input volume to be processed per hour (3000,000 transactions). The simulation failed as expected.
6. The scale-out experiment to discover if four second response time can be maintained if ten servers are used. 3.88 seconds response time was reported and the simulation passed.
7. The final scale-out experiment was conducted using twenty-four servers to process twice the normal load (2,400,000). The simulation provided a constant successful result.

8. The first scale-up experiment was conducted to increase the power of a base-level single server to ten CPUs. The single server provided close the required response time (4.18 seconds) and passed the test.
9. The scale-up experiments 9, 10, and 11 were to use a high-CPU-power server, 64 Intel Itanium2 - 1.6 GHz to provide increased power. The simulation tool did not contain the performance characteristics for a high-CPU-power server. An analytical computation was used to determine if the high-CPU-power server with a TPC-C benchmark [12] of [4,092,000, or 10 times the TPC-C power of base-level server] would provide the necessary power to process 1,200,000 batch transactions in the eight-hour window. The calculation was positive. The last two calculations on the scale-up experiment (10 and 11) to double the load, were too close to be comfortable, using all of the 4 million TPC-C power at a cost of almost $ 12 million USD. An application-specific benchmark should be constructed to determine the feasibility of this alternative.

The true test of the simulation model and the coded experiments is "did it answer the design questions posed in section 2" in a timely manner: Which optimizing technique should be used to satisfy the response time constraints? Answer – Both optimizing techniques will satisfy the response time constraints. Which method should be selected to guarantee that the batch cycle will complete within the time window requirements? Answer – Only the scale-out technique will satisfy this constraint. How can the system perform and remain scalable? Answer - The system can perform and remain scalable, if the scale-out technique is used. The eight HRS simulation experiments, in table 2, executed in less than one elapsed-day.

Additionally, did the simulation model experiments reconfirm the earlier HRS architectural design decisions? Answer – It confirmed one design decision, and brought the scale-up decision into question. Has our confidence in decisions been raised? Answer – Our confidence has been elevated in the scale-out decision.

Did we gain any insight by discovering new architecture components? Answer – Yes, we learned the importance of the cost component (e.g. the cost of ten scale-out servers, required to satisfy the maximum response time constraint, is one-half the cost of a questionable large scale-up server). Can we now create a more comprehensive SIG and a more comprehensive simulation model? Answer - Yes, the key design decision is now, how do we approach the eight-hour batch cycle window in the scale-out implementation? The answers to these questions are summarized in table 2.

4 Related Work and Discussion

Goal-oriented analysis has been used to qualitatively explore and make architectural decisions [13], and has been extended to capture concrete quantitative performance requirements. However, integrating goal-oriented analysis with simulation appears to be novel and could be helpful in providing confidence in the architectural decisions and models, and perhaps could prevent potentially expensive architectural rework if the architecture is found not to provide adequate scalability. For example, in our running example, the initial analysis determined that scale-up solution was desirable based on past experience; however, the simulation showed that the approach would

not be sufficiently scalable to handle future expected workload within the 8 hours window constraint. Prior to our experimental study, Kim and Ellis constructed performance analytic models to observe the performance and scalability behavior of workflow systems and found that single server architectures are not very scalable [14]. The findings of this study encouraged us to take the next step into modeling and discrete event simulation, which was confirmed by our simulation. As the result, the goal-driven simulation could also provide an attractive alternative to the quantitative analysis, as simulation is based on the same architectural design model and also provides easy to understand and tangible results. A comprehensive view of the production-ready HRS Softgoal Interdependency Graph and the simulation model topology diagram can be found in earlier analysis working papers [15].

5 Conclusion and Research Direction

Two key issues during architectural design are performance and scalability. In this paper, we have presented a methodology which integrates goal-orientation and simulation to analyze the design decisions on performance- and scalability-related architectural issues, and either re-confirm the validity of such decisions or make corrections on them. At least through one case study, we have shown that goal-orientation and simulation modeling complement each other in exploring, and narrowing, the performance and scalability design space. Goal oriented design provides structure and formal input to simulation model creation - something that has been needed for a long time. Early simulation modeling and experimentation can provide a goal-oriented architectural design with a closed-loop feedback mechanism to reconfirm design decisions without committing to detailed/component design and implementation of the system. Lessons learned from our case study suggest that future research direction should involve providing simulation techniques for additional non-functional requirements. For example, how do we reconfirm our decisions on security-related NFRs, concerning operationalization methods such as a two-factor authentication? What repeatable experiments [attacks] can be devised to test this sub-goal?

Acknowledgment. Thanks for encouragement and recognition is given to Alfonso Lopez, EDS Distinguished SE, for building the initial system simulation model.

References

1. Menasce, D., Vigilio, A.: Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning. Prentice Hall PTR, Englewood Cliffs (2000)
2. Pahl, G., Beitz, W.: Engineering Design: A Systematic Approach. Springer, Berlin (German 1977, English 1995)
3. Giorgini, P., Mylopoulos, J., Nicchiaelli, E., Sebastiani, R.: Reasoning with Goal Models, pp. 167–181. Springer, Heidelberg (2002)
4. Dardenne, A., Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Science of computer Programming (1993)

5. Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, Dordrecht (2000)
6. Mylopoulos, J., Chung, L., Nixon, B.: Representing and using nonfunctional requirements: a process-oriented approach. IEEE Transactions on Software Engineering 18(6), 483–497 (1992)
7. Yu, E., Mylopoulos, J.: Understanding "why" in software process modeling, analysis, and design. In: Proc. 16th int'l conference on Software engineering, pp. 159–168 (1994)
8. Smith, K., Wescott, B.: Fundamentals of Performance Engineering. HyPerformix Press, Austin (2007)
9. Law, A., Kelton, D.: Simulation Modeling and Analysis. McGraw Hill, USA (1991)
10. Pritsker, A.: Introduction to Simulation and SLAM II. Wiley Systems, New York (1995)
11. Forrester, J.: Industrial Dynamics. Productivity Press, Cambridge (1961)
12. The Transaction Processing Performance Council, `http://www.tcp.org`
13. Hill, R., Wang, J., Narhstedt, K.: Towards a Framework for Qualifying Non-Functional Requirements. In: IEEE International Requirements Engineering Conference (RE 2004), pp. 1–6 (2004)
14. Kim, K., Ellis, C.: Narhstedt.: Workflow Performance and Scalability Analysis Using the Layered Queuing Modeling Methodology. In: GROUP 2001, pp. 135–143. ACM, New York (2001); 1-58113-294-8/01/0009
15. Hill, T., Supakkul, S., Chung, L.: Analyzing Architectural Decisions on Scalability: A Goal-Driven Simulation Approach (2009)