A Case Study in Eliciting Scalability Requirements

Leticia Duboc[1] Emmanuel Letier[1] David Rosenblum[1] Tony Wicks[2]

[1]Department of Computer Science
University College London
Gower Street
London WC1E 6BT
United Kingdom
{l.duboc, d.rosenblum, e.letier}@cs.ucl.ac.uk

[2]Fortent Ltd
80-110 New Oxford Street
WC1A 1HB
United Kingdom
t.wicks@fortent.com

Abstract

Scalability is widely recognized as an important software quality, but it is a quality that historically has lacked a consistent and systematic treatment. To address this problem, we recently presented a framework for the characterization and analysis of software systems scalability. A key weakness of that initial work was that its quantitative analysis was based on arbitrarily defined variables and functions, which could compromise its results. This risk can be mitigated through a systematic exploration of system scalability goals and application domain during requirements engineering.

This paper describes our application of goal-oriented requirements engineering (GORE) for eliciting the scalability requirements of a large, real-world financial fraud detection system. The case study reveals both the suitability and the limitations of GORE as a technique for eliciting the information needed by stakeholders to specify scalability goals of a system. In the paper, we describe these findings in detail and chart a course for future research in extending goal-oriented techniques to scalability requirements.

1 Introduction

Scalability is a software quality largely overlooked during requirements engineering, in part because of the lack of techniques for eliciting quantifiable and testable scalability requirements. In previous work, we have proposed a framework to characterize and analyze the scalability of software systems [3]. A problem with that initial work is that it did not provide sufficient means to instantiate precisely the elements it uses in its scalability analysis, which could compromise its results. To address this problem, it is necessary to undertake a clear, consistent and systematic exploration of the system's scalability goals and application domain. After an investigation of available techniques, we settled on *goal-oriented requirements engineering* (GORE) as a good candidate and investigated its suitability by applying it to elicit the scalability requirements of a real-world financial fraud detection system, the Intelligent Enterprise Framework (IEF).

In this paper, we describe our experience with the application of GORE to IEF and discuss its advantages and shortcomings as a technique to elicit scalability requirements. The paper is structured as follows: Section 2 presents a brief discussion of how scalability requirements are viewed in the software industry, our scalability framework and GORE. Section 3 introduces IEF and section 4 demonstrates our application of GORE to elicit IEF's requirements. Section 5 presents a critical evaluation of both our experience with IEF and of GORE as a technique for eliciting scalability requirements. Finally, Section 6 concludes the paper.

2 Background

2.1 Scalability Requirements in Industry

To the best of our knowledge, there is no established work specifically on scalability and requirements. Traditional requirements elicitation techniques, such as use cases, focus on functional requirements. For non-functional requirements, approaches such as the NFR framework [2] allow one to specify non-functional concerns as *soft goals* at a high level of abstraction only, in a way that is too vague for a precise scalability analysis of the system.

In industry, the lack of techniques to elicit and articulate quantifiable and testable scalability requirements leads

to a number of problems. Our discussions with developers of 16 companies of different sizes and industries reveals that they generally paid no systematic attention to scalability during requirements engineering. Many companies concentrate on functional requirements, never considering load, growth characteristics, or technical and physical boundaries when designing and building their systems. Although time-to-market concerns are arguably justifiable for some systems, ignoring non-functional requirements often leads to scalability problems that cannot be fixed quickly, and the repeated use of workarounds can make the system unmanageable.

When scalability is not completely ignored, requirements are often described only in terms of application domain boundaries. These boundaries are based on assumptions that often change, either because they are found to be incorrect or because the application domain has changed. Furthermore, focusing on limiting cases of relevant domain or design variables may lead to a solution that does not perform at its best for the sub-ranges of greatest interest to the stakeholder (such as the sub-ranges corresponding to the market segment that generates most of its revenue). Sometimes, scalability requirements can be drawn naturally from quality of service agreements with respect to different values of the scaling ranges. Other times, however, functions describing scalability requirements seem to be derived from folklore or the willingness to comply with generally accepted good practices in software development, which could impose unjustifiable demands on the system design (such as the vague belief in the need for linear scalability [1]). Furthermore, management often pushes for a quick solution, even when scalability problems have been predicted. As a result, people tend not to take responsibility for scalability, leading to non-functional and workload tests being largely overlooked. Scalability problems are, as a consequence, often discovered during production.

2.2 A Framework for Scalability

In previous work, we defined scalability as *a quality of software systems characterized by the operational impact that scaling aspects of the system environment and design have on certain measured software qualities, as these aspects are varied over expected operational ranges* [3]. If the system can accommodate this variation in a way that is acceptable to appropriate stakeholders, then it is a scalable system. We also defined a technique to reveal through modelling or testing the causal relationships underlying system scalability in terms of variables representing the application domain (such as the number of simultaneous users), the system design (such as the thread pool size), and relevant software qualities (such as average or peak throughput). The technique is built around the use of *preference functions* and *utility functions*.

As in other software analysis techniques, the correctness and usefulness of the results are highly dependent on the selection and measurement of variables, their scaling bounds, and functions to be used during testing and analysis. However, in case studies we conducted in our previous work, we discovered that it is all too easy to make these selections arbitrarily. Consequently, it is apparent that it is necessary to derive the *variables*, *scaling ranges*, *bounds on the ranges* and *characteristic functions* in a clear, consistent and systematic manner from system goals in order to increase the precision and usefulness of scalability analysis results. We therefore have begun to use GORE for such purposes.

2.3 GORE

Goal-oriented requirements engineering (GORE) supports a natural elicitation of software requirements in the context of high-level goals [2, 5]. Goals may be formulated at different levels of abstraction, ranging from high-level strategic concerns to low-level, technical concerns. Goals also cover different types of concerns: functional concerns associated with the services to be provided, and non-functional concerns associated with quality of service—such as safety, accuracy, performance and so forth. A requirement is a goal under the responsibility of a single agent in the system-to-be becomes, while an assumption is a goal under the responsibility of an agent in the environment.

Often, non-functional goals do not need to be satisfied in an absolute sense. In the exploration of alternative system proposals, each alternative may have a different impact on the degree of satisfaction of higher-level goals. The partial degree of satisfaction of a goal can be modeled by annotating goals with measurable *quality attributes* and *objective functions* defined over these quality variables [4].

3 The Intelligent Enterprise Framework

Over the last 10 years, the banking and finance sector has seen a considerable consolidation. The rate of electronic transactions has increased, analytical methods have become more complex, and system performance expectations are higher. Fortent's Intelligent Enterprise Framework (IEF) is a platform designed to process large volumes of data, builds adaptive profiles of *business entities* (such as accounts and account holders) that are represented within the data and, through the applications of analysis methods, generate automated alerts to notify users of behavior that appears fraudulent. In order to maintain the system's performance as data volumes have grown, in a lifespan of seven years, the system has been subjected to significant enhancements in anticipation of scalability barriers; these extensions involved significant time and effort on the part of developers.

The first stages of the data analysis are performed by a critical sub-system, the *Data Manager*, which is responsible for the validation, preprocessing and migration of the transactional data to the database. In IEF's first design (in 2000), the Data Manager had known scalability limits. Its implementation incurred a high storage overhead, increasing both memory footprint and processor activity. The Data Manager was then re-designed in 2003. The new design decreased the demand for memory by using a combination of memory and disk storage during data manipulation. Over the subsequent few years, the number of distinct business entities continued to grow further, requiring time-consuming I/O. It then became apparent that the second design would eventually incur unacceptable processing times in the system. Therefore, in 2007, the Data Manager entered its third generation of design, to avoid predicted scalability problems. The implementation of this third design is written in Java and contains 1,556 classes and 326,293 lines of code.

4 Scalability Requirements of IEF

Our requirements engineering activity took place in parallel with the third major enhancement of the Data Manager. Therefore, the requirements for this new version, arguably, had been defined. As originally stated, scalability requirements were more a statement of idealized goals rather than rational requirements with clear understood objectives and, in particular, they relied on implicit assumptions. However, without a clear statement of these assumptions it would not be possible to perform a reliable analysis of IEF's scalability. For example, the initial requirements document did not include adequate quantitative figures about the expected growth in data volumes, about the operating environment on which the new version of the software would have been and about the expected behaviour of the system if data volume and machine capacity were scaled beyond the stated limits. This figures were needed in order to adequately test the design of the new system.

Our main objective was, therefore, to derive precise and correct measurable requirements in order to test the scalability of the Data Manager. Scalability requirements are highly dependent on the application domain and should be explored in the context of high-level business goals. We therefore focused our analysis by looking at the interests of Fortent's customers and the nature of their operating environments that most affected the Data Manager[1]. Fortent's customers are primarily retail, investment banks and other financial institutions, but for the reminder of this paper, we will use the word *banks* to consider this group.

We applied the KAOS goal-oriented method [5] in order to elicit and specify the scalability requirements of IEF. Figure 1 presents a portion of the goal tree we developed. A bank is concerned with recognizing unusual transactions, as they can represent illicit activities, such as fraud or money laundering. It is in the interest of the bank that *fraudulent transactions are detected and acted upon in a timely manner*, as represented by Node a.1 in Figure 1. For this goal to be achieved, it requires that *possible fraudulent transactions are signaled quickly* (Figure 1, Node b.1), *signaled transactions are investigated* (Figure 1, Node b.2), and *fraudulent transactions are acted upon* (Figure 1, Node b.3), which may mean not authorizing the completion of the transaction. While the last two goals are solely the responsibility of the bank staff, the signaling of suspicious transactions should be performed with the support of IEF.

Suspicious fraudulent transactions can be signaled in real-time or before the next working day. In the former, *transactions are provided and processes continuously*, while in the latter, *transactions are provided in daily batches and processed overnight*. The selection depends on the type of fraud being addressed and the upstream banking processes. Both alternatives are captured by Nodes c.1 to c.4 in Figure 1. We have only considered the refinement of the "batch-based" solution'. In order to guarantee that the bank staff can start to investigate suspicious transactions as soon as possible on the next working day, the stakeholders established an acceptable processing time window of 4 hours, with a tolerance of 20%, thereby establishing a maximum threshold of 4.8 hours. However, ideally, they would like the new version of the system to maintain the processing time of the previous version, which was around 3 hours. These objectives are modelled as functions in the goal "*Batch Processed Within Time Window*" (Figure 1, Node c.3). Furthermore, as the goal refers to the ability of processing the data batch within a specified time, *processing time* was chosen as a quality variable. The description of this goal is as follows:

Goal Achieve[Data Batch Processed Within Time Window]

Instance of *performance goal*

Def *System should be able to process the data batch within an specified time window. Processing encompasses all the required tasks from reading the data files until all suspicious transactions are signaled for investigation by the bank.*

Objective Functions

Name	Definition	Mode	Target % of Batches
3hrProcTime	p_time within 3 hrs	Max	50%
4hrProcTime	p_time within 4 hrs	Max	80%
4.8hrProcTime	p_time within 4.8 hrs	Max	100%

Quality Variable

p_time : time

Def: *Total time required from reading the data batch files until the signaling of all suspicious transactions.*

Sample Space: *Set of batches submitted.*

In order for the data batch to be processed successfully, IEF must be able to handle the bank's transactional volume.

[1] Please note that the actual numbers used in the requirements have been changed in order to protect Fortent's proprietary information.

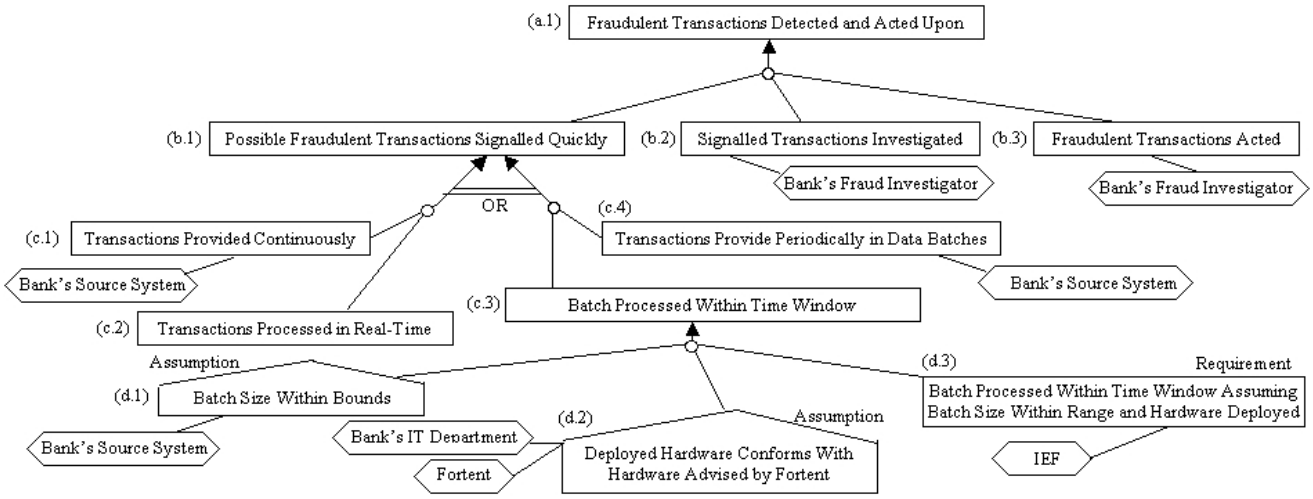

Figure 1. IEF Goal Tree

Therefore, Fortent has to determine the *range* of the data volume IEF must support and assumes that *the data batch provided by the bank will never exceed the assumed maximum data batch size* (Figure 1, Node d.1). This assumption is defined as follows:

Assumption Maintain[Batch Size Within Bounds]

Def *For every bank, the size of the data batch submitted to the system should be less than the expected maximum size for the bank. Data batch size corresponds to the number of records in the batch, since every record can be roughly considered of the same size. The number of records can be estimated from the number of accounts in a bank. This varies between 0.5 and 3 transactions per account per day. The number of accounts varies from 200 transactions to 100 million transactions, with majority of banks in range 2 million to 10 million transactions*

Quality Variable

batch_size : size

Def *Number of records in a batch.*

Sample Space: *Set of batches submitted.*

Responsibility: *Bank source system.*

The precise specification of the assumptions required much elicitation effort and allowed us to uncover inconsistencies and disagreements about projected values. These inconsistencies could be resolved through negotiation with all parties and was one of the main benefits of the requirements activity.

Fortent handles banks of different sizes by varying the hardware infrastructure on which IEF is executed. Satisfaction of the goal "*Batch Processed Within Time Window*" therefore also relies on the assumption that the *hardware deployed in the bank's site conforms with Fortent's advice for the expected maximum batch size* (Figure 1, Node d.2). Although Fortent has the means to estimate the hardware required for different data volumes, these estimates are based on the existing implementation of the system. The new design means that the relationship between hardware and data volume will have to be revisited, resulting in an implicit requirement: *the behaviour of the system for a varying data volume and hardware configurations should be predictable*. This assumption is defined as follows:

Assumption Achieve[Deployed Hardware Conforms With Hardware Advised for Batch Size]

Def *For every bank, a hardware configuration is advised based on the expected maximum batch size for the bank. The bank is expected to deploy a hardware that is at least as powerful as the one advised by Fortent to that particular bank. Hardware configuration is defined in terms of the number of CPUs, the size of the RAM and the size of the disk.*

Responsibility: *Bank & Fortent.*

Finally, the goal stating that *batches should be processed within a time window for varying assumptions on the banks' maximum batch size and deployed hardware* (Figure 1, Node d.3) defines the scalability requirement for IEF. The definition of this requirement is as follows:

Goal Achieve[Data Batch Processed Within Time Window on Varying Batch Size and Hardware]

Instance of *performance goal | scalability goal*

Def *For every bank, if the bank hardware satisfies Fortent's hardware requirements and if the submitted batch is smaller than the assumed maximum batch size, then the batch processing time should conform with the objective functions defined in the goal "Data Processed Within Time Window".*

Responsibility: *IEF System*

Nodes d.1 and d.2 are realized by the combination of two application domain agents, *bank* and *Fortent*, making these assumptions according to GORE. Node d.3 is assigned to *IEF*, turning the goal into a requirement. This requirement can be used to derive scalability test cases for the system.

5 Critical Evaluation

Fortent can use the goal models resulted from the application of GORE to support a number of activities, such as scalability analysis, derivation of test cases and hardware sizing. As with any requirements engineering exercise, human issues were a key contributor to the difficulties we experienced in applying GORE to elicit the scalability requirements of IEF. Knowledge is spread across individuals, and their assumptions are a mix of "common sense" plus peoples' experiences on different projects. Some data characterization had been performed, but these were isolated attempts, whose results had not been widely circulated. We found the elicitation of measurable requirements particularly difficult. The varying characteristics of the customer base led to different assumptions to be considered and people found it difficult to provide definite estimates, forcing us on occasion to perform our own data characterization.

As part of the GORE exercise, we explored high-level business goals that provided a rationale for low-level requirements. With respect to the scalability framework, GORE allowed a more precise characterization of scaling ranges and objective functions. Some scaling bounds happened to be more flexible than originally stated, while others were found to be more rigid. On a previous scalability analysis of IEF [3], one of the preference functions stated that the higher the throughput the better. This preference, however, does not represent the true goals of the business—in fact, the use of such a function might result in an over-engineered system. Stakeholders ultimately are not directly interested in achieving the highest throughput possible. Instead, what they really want is to ensure that a data batch is processed within a given time window, as represented by the objective functions in the resulting goal graph. One of the tools that enabled us to elicit goals and estimate bounds more precisely was to get an explicit statement of all underlying assumptions about the application domain and deployment environments.

With respect to GORE, we found some of its well recognized advantages particularly useful for scalability:

Uncovers assumptions about the application domain: The scalability of a system is highly dependent not only on the assumptions made about the current system environment, but also on the estimation of this environment in the future. Making such assumptions explicit is crucial for scalability.

Provides rationale for requirements: Without an explicit statement of their rationale, scalability requirements may impose unjustifiable demands on the system design. Goal refinement trees provide traceability from high-level strategic objectives to low-level technical requirements, so that correctness and sufficiency of scalability requirements can be validated.

Provides traceability: Traceability is particularly useful in the context of scalability. If assumptions on the ranges of the application domain are, in the future, found to be incorrect or no longer valid, they can be easily traced into the system requirements and design.

Assignment of Responsibilities: By assigning goals to agents, GORE can distribute the responsibility for scalability by, for example, making human agents responsible for characterizing the expected workload.

Measurable Quality variables and objective functions: The ability to express objective functions in terms of measurable quality variables was a strong advantage over our previous use of preferences and utility functions that relied entirely on untestable values.

However, we found a number of difficulties in applying GORE for eliciting scalability requirements. Most importantly, there is currently a lack of sound techniques for elaborating scalability goals. In particular:

Handling ranges in the application domain: Although GORE provides a means of identifying and specifying measurable quality goals as objective functions, it overlooks the existence of ranges (and its probability distribution of values) on the sample spaces of quality variables and thresholds on the value of these variables. As a result, techniques are needed for exploring and articulating such ranges in a goal tree.

Taxonomy of application domain assumptions: While goals have a clear taxonomy (functional, performance, reliability, security, etc.), we found no taxonomy for assumptions. Should an assumption on the range of a variable of the application domain be treated in the same way as an assumption on the correctness of the input format? Which elements should each category of assumptions contain? Such taxonomy and associated heuristics would help future modelling efforts.

Handling disagreements on range values: In IEF, some of the "conflicts" between assumptions were not actually conflicts or diversions in the usual sense[5], but rather disagreements about the range of values for application domain variables. We were lacking an appropriate way to model these disagreements in the goal model.

Handling time-varying assumptions: How to deal with assumptions that vary over time? For example, the increase of electronic transactions and advances on hardware technologies means that the assumptions on

IEF's application domain and required infrastructure are bound to vary over time. Can this evolution be represented in goal models?

Modelling Hardware Characteristics in Goal Trees: Scalability is frequently dealt with by means of a hardware-based scaling strategy, where the system's capacity is increased to deal with the variation on the application domain [6]. The boundary between requirements and design is vague, so when should a distinction between hardware and software be made in a goal model? For IEF, we explicitly chose to model assumptions on hardware infrastructure because such strategy imposes a requirement on the predictability of system performance on various infrastructures, and because the responsibility for software and hardware were spread over two organizations.

6 Conclusion

In this paper we described our experience in applying goal-oriented requirements engineering to elicit and articulate the scalability requirements of a large real-world system. We found that a number of well-recognized advantages of GORE are particularly useful for scalability. However, there is a lack of techniques for elaborating goal models with respect to scalability goals.

We intend to use our experience to devise systematic techniques for the application of GORE to elicit scalability requirements. In particular, we intend to create refinement patterns for scalability goals, to devise a taxonomy and describe elements for scalability assumptions, and to investigate conflicts between and evolution of range assumptions.

7 Acknowledgments

This work was partially supported by the European IST FET programme in project SENSORIA (IST-2005-016004). Leticia Duboc is funded under a studentship from UCL. David Rosenblum holds a Wolfson Research Merit Award from the Royal Society. The authors gratefully thank Steve Brewin, Iain Mclaren, Andrew Towers and other seasoned developers for discussions of this work.

References

[1] G. Brataas and P. Hughes. Exploring architectural scalability. In *Proc. 4^{th} WOSP*, pages 125–129. ACM Press, 2004.

[2] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Springer, October 1999.

[3] L. Duboc, D. Rosenblum, and T. Wicks. A framework for characterization and analysis of software system scalability. In *Proceedings of ESEC-FSE'07*, pages 375–384, New York, NY, USA, 2007. ACM.

[4] E. Letier and A. van Lamsweerde. Reasoning about partial goal satisfaction for requirements and design engineering. In *Proceedings of FSE'04*, pages 53–62, New York, NY, USA, 2004. ACM.

[5] A. van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proceedings of RE '01*, page 249, Washington, DC, USA, 2001. IEEE Computer Society.

[6] C. B. Weinstock and J. B. Goodenough. On system scalability, 2006. SEI, Carnegie Mellon University CMU/SEI-2006-TN-012.