Systematic Elaboration of Scalability Requirements through Goal-Obstacle Analysis

Leticia Duboc, Emmanuel Letier, and David S. Rosenblum, *Fellow*, *IEEE*

Abstract—Scalability is a critical concern for many software systems. Despite the recognized importance of considering scalability from the earliest stages of development, there is currently little support for reasoning about scalability at the requirements level. This paper presents a goal-oriented approach for eliciting, modeling, and reasoning about scalability requirements. The approach consists of systematically identifying scalability-related obstacles to the satisfaction of goals, assessing the likelihood and severity of these obstacles, and generating new goals to deal with them. The result is a consolidated set of requirements in which important scalability concerns are anticipated through the precise, quantified specification of scaling assumptions and scalability goals. The paper presents results from applying the approach to a complex, large-scale financial fraud detection system.

Index Terms—Requirements/specifications, analysis, performance measures, quality analysis and evaluation, goal-oriented requirements engineering, KAOS, scalability

1 Introduction

Scalability is widely regarded as one of the most important qualities of software-intensive systems [1], yet fundamental principles for developing scalable systems are still poorly understood. As with other nonfunctional system qualities, a system's ability to scale is strongly dependent on decisions taken at the requirements and architecture levels [2], [3]. Failure to consider scalability adequately in these stages may result in systems that are impossible or too costly to change if scalability problems become apparent during testing or system usage.

Current technologies, such as virtualization and cloud computing, may help to achieve scalability in software systems. However, a system whose architecture fails to support some of its scalability goals cannot be made scalable simply by employing virtualization and clouds for its implementation. In order to take advantage of these technologies, one must understand and define the system's scalability goals precisely. An a priori analysis of scalability goals prevents system analysts from being caught prematurely into discussions about technology and gives them more freedom to explore, negotiate, and fulfill these goals. There is therefore a strong need for systematic techniques to reason about scalability during the early stages of development.

- *L. Duboc is with the Department of Computer Science, State University of Rio de Janeiro (UERJ), R. São Francisco Xavier, 534-Maracanã, Rio de Janeiro, RJ-CEP 20550-900, Brazil. E-mail: leticia@ime.uerj.br.*
- *E. Letier is with the Department of Computer Science, University College London, Gower Street, London WC1E 6BT, United Kingdom. E-mail: e.letier@cs.ucl.ac.uk.*
- *D.S. Rosenblum is with the School of Computing, National University of Singapore, Computing 1, 13 Computing Drive, Singapore 117417, Republic of Singapore. E-mail: david@comp.nus.edu.sg.*

Manuscript received 30 Oct. 2010; revised 11 Jan. 2012; accepted 21 Jan. 2012; published online 13 Feb. 2012.
Recommended for acceptance by M. Jackson.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2010-10-0322.
Digital Object Identifier no. 10.1109/TSE.2012.12.

In our previous studies of scalability, we defined scalability as *the ability of a system to maintain the satisfaction of its quality goals to levels that are acceptable to its stakeholders when characteristics of the application domain and the system design vary over expected operational ranges* [4]. A system's scalability is therefore relative to other primary quality goals for the system. For example, the scalability of a Web search engine may be characterized by its ability to return search results in a time that remains almost instantaneous (a performance goal) when the number of simultaneous queries and the number of pages indexed increase. Scalability, however, is not limited to performance goals. The scalability of an air traffic control system may be characterized by its ability to keep safe separation distances between airplanes (a safety goal) when the number of airplanes in the airspace managed by the system increases. One should note that, as illustrated by the latter example, scalability is not limited to performance goals. Performance is only one of the many possible indicators of scalability; there are others, such as are reliability, availability, dependability, and security [4].

Defining the scalability requirements of a complex system is not trivial. Many issues arise during their elaboration [5]:

- how to specify scalability requirements that are precise and testable so that they provide adequate inputs for architectural design and analysis;
- how to identify and manage scalability-related risks early in the development process, and in particular how to make sure one has identified all application domain quantities whose variations could have an impact on the system's scalability;
- how to deal with the inevitable unknowns, uncertainties, and disagreements about expected ranges of variations for these domain quantities;
- how to ensure that the specified scalability requirements are neither too strong nor too weak with

respect to business goals, to expected ranges of variations in the application domain, and to the capabilities of current technologies; and
- how to find adequate tradeoffs between scalability requirements and other qualities, in particular those related to cost and development time.

Existing requirements engineering techniques provide little concrete support to address these specific issues. User stories and use-case-based approaches to requirements engineering overlook scalability concerns and other nonfunctional requirements altogether. A few industry white papers provide high-level guidelines for writing quantifiable scalability requirements, but no precise foundations nor any guidance to deal with the above issues [6], [7]. Goal-oriented approaches such as KAOS [8], i* [8], [9], and Problem Frames [10] provide generic support for elaborating, structuring, and analyzing software requirements, including both functional and nonfunctional requirements, but no specific support to deal with scalability concerns. A rudimentary application of a goal-oriented approach that would consist of, say, specifying a high-level goal named Scalable System and gradually refining it into more precise requirements is not satisfying because of the lack of a precise definition for the scalability goal and the lack of support for refining it into testable scalability requirements.

Dedicated requirements engineering techniques exist for other important categories of nonfunctional requirements such as safety [11], [12] and security [13], [14], [15]; however, up to now no specific attention has been given to scalability requirements. Surprisingly, scalability is not even included in some taxonomies of nonfunctional requirements in industrial standards or in the software engineering literature [8], [16], [17], [18], [19]. As a consequence, scalability requirements are often overlooked or defined in vague terms at best [5].

This paper presents a systematic method for elaborating and analyzing scalability requirements, and it presents the results of a case study in which we applied the method to a complex, large-scale financial fraud detection system. The method has been formulated in the context of the KAOS goal-oriented requirements engineering method [8], and it relies on KAOS for the elaboration of goal models, for managing conflicts between goals, and for evaluating and selecting among alternative system designs. We extend the conceptual framework of KAOS with precise characterizations of the concepts of *scalability goals*, *scalability requirements*, and *scaling assumptions* that are essential for reasoning about scalability at the requirements level. We also extend the KAOS model elaboration process with concrete support for elaborating and analyzing such concepts.

Our approach deals with scalability concerns during the goal-obstacle analysis steps of the goal-oriented elaboration process [20]. The basic idea consists of systematically identifying and resolving potential variations in domain quantities that will cause goals to fail; we will define such conditions as *scalability obstacles*. The aim of a scalability obstacle analysis is to anticipate scalability related risks at requirements elaboration time, when there is more freedom for finding adequate ways to deal with them, rather than discovering them during system development or, worse, experiencing unanticipated scalability failures at runtime. Existing obstacle analysis techniques [20] are not sufficient for handling scalability obstacles because they support only generic analyses in which the specifics of scalability obstacles are not considered. As a result, scalability obstacles are likely to be overlooked during obstacle identification (as they were, for example, in a previous obstacle analysis of the London Ambulance Service case study [20]). We present new, more specific *heuristics*, *formal patterns* and *model transformation tactics* for identifying and resolving scalability obstacles.

2 Background

2.1 The Intelligent Enterprise Framework

Our presentation relies on examples of a real-world system, Searchspace's Intelligent Enterprise Framework (IEF), that we used to develop and validate our techniques.

IEF is a platform used by banks and other financial institutions for processing large volumes of financial transaction data, and notifying users (such as bank employees) of potentially fraudulent behavior within the data. IEF has important scalability requirements due to the recent widespread consolidation in the banking and finance sector. The rate of electronic transactions has increased dramatically, analytical methods have become more complex, and system performance expectations have become higher. Over the last 10 years, the number of installations of the company's software went from a handful of deployments in a small number of countries to many hundreds of deployments, spread worldwide. The volume of data processed increased from a single deployment peak of about 3 million transactions per day to over 90 million transactions for a recent deployment, coupled with an increase in the complexity of the processing performed.

The system uses a number of analytical methods for recognizing fraudulent transactions. For simplicity, we consider here only the comparison of transactions delivered by a bank against known fraudulent patterns, which may be done continuously or overnight. In the former, transactions are streamed by the bank system into the IEF and processed as soon as they arrive. In the latter, transactions are provided periodically and processed in data batches. The choice of approach depends on the type of fraud being addressed and the upstream banking processes. Alerts about potentially fraudulent transactions are generated by the IEF and then investigated by the bank, which will dismiss or confirm the frauds and then take appropriate measures.

2.2 Goal-Oriented Requirements Engineering

Goal orientation is a paradigm for the elicitation, evaluation, elaboration, structuring, documentation, and analysis of software requirements. The KAOS approach is composed of a modeling language and a constructive elaboration method supported by informal heuristics and formal techniques [8]. The modeling language combines multiple system views for modeling goals, domain objects, agents and their responsibilities, agent operations, and system behaviors through scenarios and state machines. The method consists of a series of steps for elaborating and analyzing the different views. In this section, we focus on presenting the key concepts that are used in the paper. Further details about the KAOS modeling language and elaboration method can be found in Lamsweerde's book [8].

A *goal* is a prescriptive statement of intent that a system should satisfy through the cooperation of agents. *Agents* are active system components, such as humans, hardware devices, and software components, that are capable of performing operations that modify the system state in order to satisfy goals. The word *system* refers to the composition of the software under development and its environment. Goals range from high-level business objectives whose satisfaction involves multiple agents (such as minimizing loss due to financial fraud, and maintaining the bank's reputation, etc.), to fine-grained technical properties involving fewer agents (such as processing all transaction batches within 8 hours). Unlike goals that are prescriptive, *domain properties* are descriptive statements that are always true in the application domain, irrespective of agents' behaviors. Physical laws are typical examples of domain properties. *Domain hypotheses* are also descriptive statements about the application domain that, but unlike domain properties, are considered to be more uncertain and subject to change.

In a goal model, goals are organized in AND/OR refinement structures. An *AND-refinement* relates a goal to a set of subgoals; this means that satisfying all subgoals in the refinement is a sufficient condition in the domain for satisfying the goal. *OR-refinement* links relate a goal to a set of *alternative* AND-refinements; this means that each AND-refinement represents an alternative way to satisfy the parent goal.

The KAOS elaboration process consists of constructing a system's goal model both top-down (by asking HOW questions to refine goals into subgoals) and bottom-up (by asking WHY questions to identify parent goals). During the goal refinement process, goals are decomposed into subgoals so that each subgoal requires the cooperation of fewer agents for its satisfaction; the refinement process stops when it reaches subgoals that can be assigned as the responsibility of single agents. In order to be assignable to an agent, a goal must be *realizable* by the agent. Realizability here is a formal condition requiring that the goal definition corresponds to a transition relation between the agent's *monitored* variables (its input) and controlled variables (its internal and output variables) [21]. A goal assigned to the software under development is called a *software requirement*. A goal assigned to an agent in the environment is called a *domain expectation*. The term *domain assumption* is used to refer to both domain expectations and domain hypotheses.

As an example, Fig. 1 shows a portion of the goal model we have elaborated for a previous version of the IEF system that supported batch processing only [5]. Graphically, goals are depicted as parallelograms and AND-refinement links are represented as arrows from subgoals to a parent goal. Agents are depicted as hexagons, and a link between an agent and a goal means that the agent is responsible for the goal. The top of the figure shows three stakeholders' goals for this system, namely, Avoid [Loss Due to Fraud], Maintain [Bank Reputation], and Avoid [Inconvenience to Account Holder]. The goal Achieve [Fraud Detected and Resolved] contributes to these three goals and is refined into the goal Achieve [Alerts Generated] requiring that fraud alerts be generated for potentially fraudulent bank transactions, and the expectation Achieve [Alerts Investigated and Acted Upon Quickly], under the responsibility of a human Fraud Investigator, requiring that generated alerts either be confirmed as fraudulent and resolved or cleared as false alerts. The goal Achieve [Alerts Generated] is itself refined into an expectation on the Bank System to provide the batches of transactions data and a requirement on the Alert Generator software component to process the batches of transactions overnight.

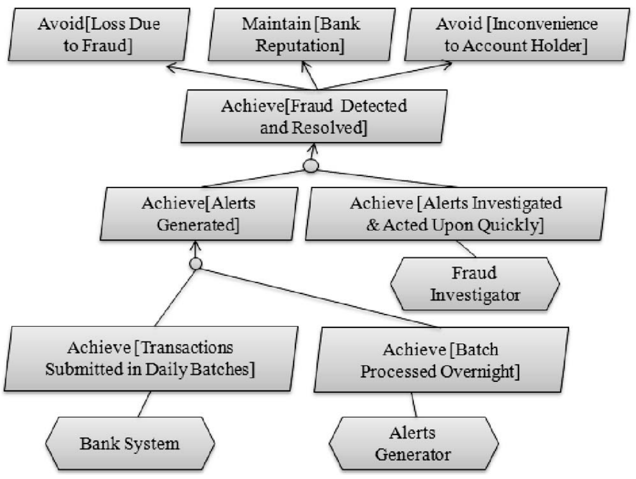
Avoid[Loss Due to Fraud]
Maintain [Bank Reputation]
Avoid [Inconvenience to Account Holder]
Achieve[Fraud Detected and Resolved]
Achieve[Alerts Generated]
Achieve [Alerts Investigated & Acted Upon Quickly]
Fraud Investigator
Achieve [Transactions Submitted in Daily Batches]
Achieve [Batch Processed Overnight]
Bank System
Alerts Generator

Fig. 1. Partial model for a previous version of the IEF [5].

2.3 Specifying Goals

Each goal has a specification including its *pattern* (e.g., Achieve, Maintain, Avoid), *name*, and natural language *definition*. A goal may also be specified as belonging to one or more categories (e.g., functional, performance, safety) and be given an optional *formal definition* expressed in Metric Linear Temporal Logic [22], which is used to disambiguate natural language definitions and to allow automated reasoning on goal models. In this paper, we will use the following standard temporal logic operators:

$\square P$	(P is true in all future states)
$\diamond P$	(P is true in some future state)
$\diamond_{\leq d} P$	(P holds in some future state that is less than d time units from the current state)

The natural language and formal definitions of a goal define what it means for the goal to be satisfied in an absolute sense. Partial levels of goal satisfaction are specified by extending goal specifications with domain-specific *quality variables* and *objective functions* [23]. Mathematically, quality variables correspond to probabilistic random variables. The specification of objective functions and quality variables extends the goal model with a probabilistic layer that can be used to compute the impact of alternative goal refinements and responsibility assignments on the degrees of goal satisfaction.

As an example, the goal Achieve [Batch Processed Overnight] has the following specification:

Goal Achieve [Batch Processed Overnight]
Category Performance goal
Definition Batches of bank transactions provided by the bank system should be processed in less than 8 hours. Processing consists in generating alerts about transactions in the batch that match known fraudulent patterns stored in the system.
Formal Spec ($\forall$ b:Batch)
$\Box$ (EndOfDay $\wedge$ b.Submitted $\rightarrow \Diamond_{\leq 8hours}$ b.Processed)
Quality Variable: procTime: Batch $\rightarrow$ Time
Definition The time to process the full batch of transactions after the end of the day.
Sample Space The set of daily batches of bank transactions submitted by the bank system.
Objective Functions At least 9 out of 10 batches should be processed within 8 hours, and all batches should be processed within 9.6 hours.

Name	Definition	Target
% of Batches Processed in 8 hours	Pr(procTime $\leq$ 8h)	90%
% of Batches Processed in 9.6 hours	Pr(procTime $\leq$ 9.6h)	100%

Responsibility Alert Generator

The natural language and formal definitions state that this goal is satisfied in an absolute sense if every batch of transactions submitted to the IEF is processed in less than 8 hours. The quality variable procTime is a random variable whose sample space is the set of daily batches submitted to the IEF and whose value denotes the time to process a batch. The first objective function states that the system should be designed to maximize the probability that a batch is processed in less than 8 hours. The final column specifies the target to be achieved for each objective function; at least 90 percent of the batches should be processed within 8 hours, and there is a tolerance of 20 percent in the processing time (i.e., 9.6 hours) for the remaining batches. The goal is assigned to the Alert Generator agent, a subsystem of the IEF.

2.4 Goal-Obstacle Analysis

Goals, requirements, and assumptions elaborated during a requirements elaboration process are often too ideal because they fail to take into consideration exceptional conditions in the application domain that may violate them. The principle of obstacle analysis is to take a pessimistic view of the model elaborated so far by systematically considering how the actual system might deviate from the model.

An *obstacle* to some goal is an exceptional condition that prevents the goal from being satisfied [20], [24]. An obstacle O is said to *obstruct* a goal G in some domain characterized by a set of domain properties *Dom* if, and only if,

- the obstacle entails the negation of the goal in the domain (i.e., $O, \mathrm{Dom} \models \neg G$); and
- the obstacle is not inconsistent in the domain (i.e., $\mathrm{Dom} \not\models \neg O$).

Graphically, obstacles are depicted as "reverse" parallelograms and related to the goals they obstruct by "negated"

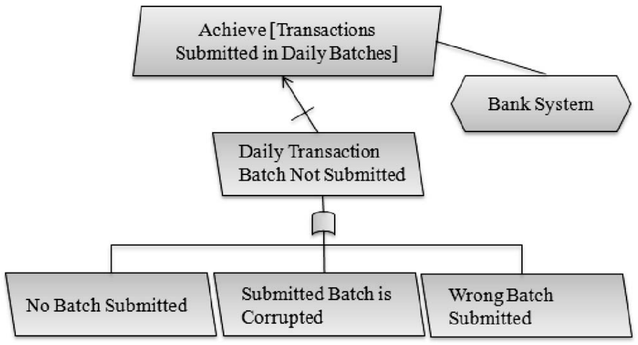
Achieve [Transactions Submitted in Daily Batches]
Bank System
Daily Transaction Batch Not Submitted
No Batch Submitted
Submitted Batch is Corrupted
Wrong Batch Submitted

Fig. 2. Obstacles to the expectation achieve [transactions submitted in daily batches].

arrows. Obstacles can be refined into subobstacles. The resulting model corresponds to a goal-anchored form of risk model. For example, Fig. 2 shows the obstacle Daily Transactions Batch Not Submitted obstructing the goal Achieve [Transactions Submitted in Daily Batches] and its OR-refinement into subobstacles No Batch Submitted, Submitted Batch is Corrupted, and Wrong Batch Submitted.

Starting from an idealized goal model, a goal-obstacle analysis consists of

1. *identifying* as many obstacles as possible by systematically considering all leaf goals and assumptions in the goal graph;
2. *assessing* the relative importance of the identified obstacles in terms of their likelihood and impacts on top-level goals;
3. *resolving* the more critical obstacles by modifying existing goals, requirements, and assumptions, or by introducing new ones so as to prevent, reduce, or tolerate the obstacles.

For example, having identified the obstacle that the Bank System could provide a corrupted batch of transactions will generate additional requirements on the Alert Generator. This identify-assess-resolve loop is repeated until all remaining obstacles are considered acceptable without further resolution.

3 Specifying Scalability Requirements

According to our definition of scalability, the elements required to define what it means for a system to be scalable are 1) the quality goals of the system, 2) the characteristics of the application domain and system design that are expected to vary, and their ranges, and 3) the acceptable levels of quality goal satisfaction under these variations. In the KAOS framework, these elements can be represented as follows:

1. Quality goals correspond to goals whose specification includes domain-dependent objective functions.
2. Expected variations of characteristics in the application domain correspond to a new kind of domain assumption that we call a *scaling assumption*.
3. The acceptable levels of quality goal satisfaction under these variations are specified as target values for the quality goal objective functions. We refer to quality goals that are constrained by scaling assumptions as *scalability goals*.

Note that, even though scalability is highly influenced by the system design and implementation, goals and assumptions related to scalability should be described in terms of domain phenomena only and should not refer to design or implementation concerns. Goals will impose constraints on the system design by making reference to assumptions about variations in the domain. Stating the constraints in terms of domain phenomena not only avoids overspecification, but also gives system designers more freedom in choosing the design that will satisfy these goals best. We describe these elements and their roles in more detail in the following sections.

3.1 Specifying Scaling Assumptions

We define a *scaling assumption* as a domain assumption specifying how certain characteristics in the application domain are expected to vary over time and across deployment environments. The latter dimension is especially important in the context of product families. The specification of scaling assumptions should therefore be defined in terms of the following items:

1. one or more *domain quantities* whose expected variations are defined in the assumption;
2. the *periods of time* and *classes of system instances* over which the assumption is defined; and
3. the *range of values* each quantity is expected to assume for each system class over each period of time.

For example, the scaling assumption Expected Batch Size Evolution presented below specifies how the number of transactions in daily batches submitted to the IEF is expected to vary over time and for banks categories.

Assumption Expected Batch Size Evolution
Category Scaling assumption
Definition From 2011 to 2015, daily batches are expected to contain up to the following numbers of transactions for different bank categories:[1]

Bank	2011	until 2013	until 2015
small	10,000	15,000	20,000
medium	1 million	1.2 million	1.8 million
large	50 million	55 million	60 million
merger	80 million	85 million	95 million

Our definition of scaling assumption covers the case where variations of domain quantities refer to a single period of time and a unique class of system instances. For example, an alternative, simpler scaling assumption on the batch size could be the following:

Assumption Expected Batch Size Evolution
Category Scaling assumption
Definition Over the next five years, daily batches of transactions for all banks are expected to vary between 10,000 and 95 million transactions.

Following the KAOS classification of properties (Section 2.2), a scaling assumption could either be a *descriptive* scaling hypothesis, in which case the assumption is assumed to hold without being prescribed on any system agent, or a *prescriptive* scaling expectation, in which case some agent would be expected to restrict its behavior so as to ensure that the scaling assumption is satisfied. Scaling assumptions are not, however, software requirements, as they describe properties that are assumed to be true in the application domain rather than properties that must be enforced by the software-to-be.

Scaling assumptions can be specified with varying levels of precision. For some systems, specifying orders of magnitude for some of the domain quantities may be sufficient to inform the early design decisions adequately at the requirements and software architecture levels. For others, one may need more precise figures, such as the ones in the above examples. In some cases, it may be useful to distinguish between the average and peak values of some of the domain quantities. The specification of a scaling assumption might even refer to the full probability distribution of a domain quantity under consideration (e.g., the distribution of the number of transactions in daily batches and the expected evolution in the distribution's parameters or moments over time).

The ranges of values referenced in scaling assumptions must be elicited and negotiated with project stakeholders. These may be obtained by analyzing characteristics of the existing system, extrapolating characteristics from related systems, making predictions about future system evolutions, and eliciting and negotiating the ranges with stakeholders. A variety of requirements elicitation techniques is available to support such activities [8]. Applying such techniques to elicit these numbers of course remains difficult due to uncertainties about future behaviors and divergences of opinion among system stakeholders. But surfacing the uncertainties and divergences explicitly during requirements elaboration is essential to an adequate analysis of a system's scalability requirements. A systematic approach for identifying what scaling assumptions need to be elicited, such as the scalability obstacle analysis technique to be described in Section 4, greatly facilitates this process.

A key feature of our approach is that the values to be estimated in scaling assumptions denote measurable, domain-specific quantities whose validity eventually can be checked in the running system. This contrasts with other quantitative techniques for specifying nonfunctional requirements that rely on estimating subjective quantities that have no domain specific meaning [25], [26], [27], [28], [29] and can therefore never be validated empirically.

3.2 The Role of Scaling Assumptions

Like other kinds of domain assumptions, scaling assumptions support the refinement of goals toward requirements and expectations that can be satisfied by a single agent [8], [30]. Semantically, scaling assumptions express constraints on the ranges of values that domain quantities are expected to assume over specified time periods and classes of system instances. Logically, the *absence* of a scaling assumption for a domain quantity means that there is *no assumed constraint* on its possible values, that is, its value at any point in time potentially could be infinite.

1. These numbers are fictitious and used for illustration only.

To illustrate this, consider again the goal Achieve [Batch Processed Overnight] in Section 2.3. In Fig. 1, we had assigned that goal as the responsibility of the Alert Generator agent. The goal is indeed realizable by the agent as it is defined entirely in terms of quantities that are monitored and controlled by that agent. Realizability implicitly assumes that agents have infinite resource capacities to perform the operations required to satisfy their goals. In practice, however, all agents have finite capacities that may be insufficient for satisfying the goal—a form of *scalability obstacle*. In this example, it will be impossible for the Alert Generator to guarantee the satisfaction of the goal for *any* size of daily batches to be processed. To obtain a goal whose satisfaction can be guaranteed by the Alert Generator alone, the goal Achieve [Batch Processed Overnight] needs to be refined into a scaling assumption on the size of daily batches, such as one of those given in the previous section, and a goal requiring daily batches to be processed within 8 hours *only for batches satisfying the scaling assumption*. Using the scaling assumption Expected Batch Size Evolution in Section 3.1, we obtain the following subgoal:

Goal Achieve [Batch Processed Overnight Under Expected Batch Size Evolution]
Category Performance goal, scalability goal
Definition At the end of each day, the batch of transactions submitted by the bank should be processed in less than 8 hours, provided that the batch size does not exceed the bounds stated in the scaling assumption "Expected Batch Size Evolution."

This goal can now be satisfied by an Alert Generator with sufficiently large but nevertheless finite capacities (i.e., finite processing speed, memory, and storage capacities).

3.3 Specifying Scalability Goals

We define a *scalability goal* as a goal whose definition and required levels of goal satisfaction (as specified by its objective functions) make explicit reference to one or more scaling assumptions. The goal Achieve [Batch Processed Overnight Under Expected Batch Size Evolution] is a scalability goal because its definition refers to the scaling assumption Expected Batch Size Evolution.

Our definition is consciously in opposition with a common intuition about scalability that consists of viewing a system as scalable if it can handle *any* increased workload in a cost-effective way [31]. In this more common view, the satisfaction of a scalability goal should not be restricted to bounds explicitly stated in scaling assumptions. Our definition, in contrast, takes the view that expected variations of relevant domain characteristics should always be stated explicitly in order to adequately inform a *cost-effective* design strategy.

The specification of a scalability goal involves describing how the required level of goal satisfaction is allowed to vary when domain characteristics vary within the bounds specified in scaling assumptions. This goal specification takes different forms, depending on whether the goal satisfaction level is required to stay the same or is allowed to vary for different values of the domain characteristics. For example, for an air traffic control system, the required safety level would remain the same when the number of airplanes simultaneously present in a region increases, whereas for an online store, one may accept system performance to slow as the number of simultaneous requests increases.

- A *scalability goal with fixed objectives* is one in which the same level of goal satisfaction must be maintained under the full range of variations specified in the scaling assumptions. These are goals for which the goal definition, objective functions, and target values are the same across the whole range of values considered in the scaling assumptions.
- A *scalability goal with varying objectives* is a scalability goal whose required level of goal satisfaction is not the same under the whole range of variations specified in the scaling assumptions. Their goal definition, objective functions definitions, or target values are different across the range of values considered in the scaling assumptions.

Scalability goals also can be extended to specify graceful degradation of goal satisfaction when the system operates outside its nominal scaling range. This can be done by specifying additional scaling assumptions or additional scaling ranges within scaling assumptions, characterizing possible values for the domain characteristics outside of the initial scaling assumptions that the system must still be able to react to. More than one level of degradation could be specified if needed. Such concerns for handling such cases are typically handled during the obstacle analysis steps of the goal-oriented requirements elaboration process that will be presented in Section 4.

Finally, scalability goals can be specified at different levels in the goal refinement graph. Following the KAOS definitions, a *scalability requirement* is a scalability goal assigned to an agent in the software-to-be.

4 Scalability Obstacle Analysis

This section describes the process of elaborating requirements models that include specifications of scalability goals and scaling assumptions.

As mentioned earlier, what it means for a system to be scalable is relative to other quality goals for the system. Identifying the primary quality goals of a system is therefore a prerequisite to the precise specification of its scalability goals. Our process for analyzing a system's scalability and elaborating scalability goals assumes that the other functional and quality goals of the system have been elaborated following a systematic goal-oriented requirements elaboration process [8].

As was illustrated in Section 3.2, during incremental elaboration of requirements models, initial specification of goals and assignments of goals to agents are typically *idealized*, in the sense that they are generally elaborated without explicit considerations for the scaling characteristics in the application domain and the limited capacities of agents for fulfilling the goals assigned to them.

Starting from idealized goals and goal assignments is beneficial as it avoids premature compromises based on

TABLE 1
Scalability Obstruction Patterns

Goal	Domain Property	Obstacle
$G : \Box P$	$\Box (P \rightarrow \mathit{Load}_G \leq \mathit{Capacity}_{Ag})$	$\Diamond (\mathit{Load}_G > \mathit{Capacity}_{Ag})$
$G : \Box (SA \rightarrow P)$	$\Box (P \rightarrow \mathit{Load}_G \leq \mathit{Capacity}_{Ag})$	$\Diamond (SA \wedge \mathit{Load}_G > \mathit{Capacity}_{Ag})$

implicit and possibly incorrect perceptions of what might be possible to achieve [32]. However, later on in the requirements elaboration process it is necessary to take into account what can be achieved realistically by the system agents and, if needed, modify the specifications of goals, requirements, and expectations accordingly.

Therefore, a natural way to deal with scalability during requirements elaboration consists of handling scalability concerns during the goal-obstacle analysis steps of the goal-oriented method. Starting from an initial goal model, our process for elaborating scalability requirements consists of the following activities:

1. systematically *identifying scalability obstacles* that may obstruct the satisfaction of the goals, requirements, and expectations elaborated so far;
2. *assessing* the likelihood and criticality of those obstacles; and
3. *resolving* the obstacles by modifying existing goals, requirements, and expectations, or generating new ones so as to prevent, reduce, or mitigate the obstacles.

The resolution of scalability obstacles leads to the identification of scaling assumptions and scalability goals introduced in the previous section.

4.1 Identifying Scalability Obstacles

We define a *scalability obstacle* to some goal G as a condition that prevents the goal from being satisfied when the load imposed by the goal on agents involved in its satisfaction exceeds the capacity of the agents.

In this definition, we define the concepts of *goal load* and *agent capacity* to denote domain-specific measures intended to characterize the amount of work needed to satisfy the goal and the amount of resources available to the agent to satisfy the goal, respectively. To identify concrete scalability obstacles, one must instantiate these concepts to domain-specific measures appropriate to the goals and agents under consideration.

This definition suggests the following heuristic for identifying a scalability obstacle from a goal: *For each goal, identify what defines the goal load, what agent resources are involved in its satisfaction, and consider an obstacle of the form* Goal Load Exceeds Agent Capacity. This heuristic extends the existing catalog of obstacle generation heuristics in [8] and [20].

For example, consider again the goal Achieve [Batch Processed Overnight] assigned to the Alert Generator agent. The goal load in this case is the number of transactions in the batches to be processed, and the agent capacity is the Alert Generator's throughput measured as the number of transactions it can process per second. A scalability obstacle to this goal is therefore the condition Batch Size Exceeds Alert Generator Processing Speed. This obstacle's name, which refers to a goal load and agent capacity expressed in different units, is extended with a more precise obstacle definition, namely, that the batch size exceeds the number of transactions that can be processed overnight given the alert generator's throughput, thus comparing terms expressed in the same unit.

As another example, consider in Fig. 1 the goal Achieve [Alerts Investigated and Acted Upon Quickly] assigned to human Fraud Investigators. This goal requires every generated alert either to be confirmed and resolved or to be cleared by a fraud investigator within a day of the alert being generated. The goal load in this case is the number of alerts generated per day, while the agent capacity is the number of alerts the team of fraud investigators is able to investigate per day. The scalability obstacle to this goal is Number of Alerts Exceeds Fraud Investigators Capacity.

In general, a scalability obstacle can be an obstacle to a goal residing at any level in the goal refinement structure. For example, for an air traffic control system, a scalability obstacle to the high-level goal Maintain [Safe Airplane Separation] would be the condition Number of Airplanes in Region Exceeds System Capacity. However, as for standard goal-obstacle analysis, we found it preferable to generate scalability obstacles from terminal goals because this generates more specific obstacles whose likelihoods are easier to assess and for which more specific obstacle resolutions can be envisaged.

4.1.1 Formal Patterns of Scalability Obstruction

In addition to informal heuristics, the existing obstacle analysis method relies on a catalog of formal obstruction patterns for identifying obstacles from goals [20]. These patterns provide a formal and more precise characterization of the informal heuristics. Two formal patterns corresponding to the above heuristics are shown in Table 1. The first pattern considers goals of the form $\Box P$, which do not refer to a scaling assumption; the second pattern is defined for scalability goals of the form $\Box (SA \rightarrow P)$, where SA is the condition defined in a scaling assumption on the goal load. In both cases, the identification of the scalability obstacle relies on a domain property $\Box (P \rightarrow Load_G \leq Capacity_{Ag})$ stating that P can be true only if the goal load does not exceed the agent capacity. The identified scalability obstacles have the form shown in the last column of the table. The first obstacle states that eventually the goal load exceeds the agent capacity. The second one states that despite the values of domain quantities being within the ranges defined in the scaling assumption, eventually the goal load exceeds the agent capacity. Note that these patterns require that the goal load and agent capacity be expressed in or converted to the same units.

4.1.2 Scalability Obstacles to Multiple Goals

When identifying scalability obstacles, it is not sufficient to consider obstacles to each goal in isolation. There might be

TABLE 2
Pattern of Scalability Obstruction to Multiple Goals

Goals	Domain Property	Obstacle
$G_1, \ldots, G_n$	$G_1 \wedge \ldots \wedge G_n \rightarrow \Box\, (Load_{G_1} + \ldots + Load_{G_n} \leq Capacity_{Ag})$	$\Diamond\, (Load_{G_1} + \ldots + Load_{G_n}) > Capacity_{Ag}$

situations where an agent's capacity is sufficient to satisfy each of its goals in isolation, but insufficient when having to satisfy all goals together.

This can be characterized formally by the obstruction pattern shown in Table 2. Consider a set of goals $G_1, \ldots, G_n$ assigned to an agent Ag and involving for their satisfaction the same agent resource Ag_{Res}. Assuming a domain property saying that the agent's capacity must always be bigger than the sum of the goals' loads in order for the goals to be satisfied, that is, $G_1 \wedge \cdots \wedge G_n \rightarrow \Box\, (Load_{G_1} + \cdots + Load_{G_n} \leq M)$, then a scalability obstacle to the combined set of goals $G_1, \ldots, G_n$ is the obstacle $\diamond\, (Load_{G_1} + \cdots + Load_{G_n} > M)$. That is, the scalability obstacle is a condition O such that $O, Dom \vdash \neg\, (G_1 \wedge \cdots \wedge G_n)$.

This pattern relies on the simplifying assumption that all goal load measures are expressed in the same unit and that the total load imposed by the goals is the sum of the load of the individual goals, but it can be generalized easily to a situation where the domain property involves any monotonically increasing function $F(Load_{G_1}, \ldots, Load_{G_n})$ instead of simple addition. When $n = 1$, this pattern is equivalent to the first obstruction pattern to a single goal in Table 1.

To illustrate this pattern, consider the Data Store agent and the two goals Maintain [Transactions Data Stored] and Maintain [Alerts Data Stored] under its responsibility in Fig. 3. Note that Fig. 3 is a *responsibility diagram*—a diagram showing an agent and a set of goals it is responsible for [8]—that has been extended with a scalability obstacle obtained by the application of this pattern.

In this example, the load for each goal is the size of the transaction data and alert data to be stored, respectively; both goals involve the Data Store's storage resource for their satisfaction, and a necessary condition for both goals to be satisfied is that the sum of the transaction data and alert data does not exceed the Data Store's storage capacity. An application of the pattern generates the scalability obstacle Transactions and Alerts Data Exceeds Storage Capacity. Note that the strict application of the pattern for

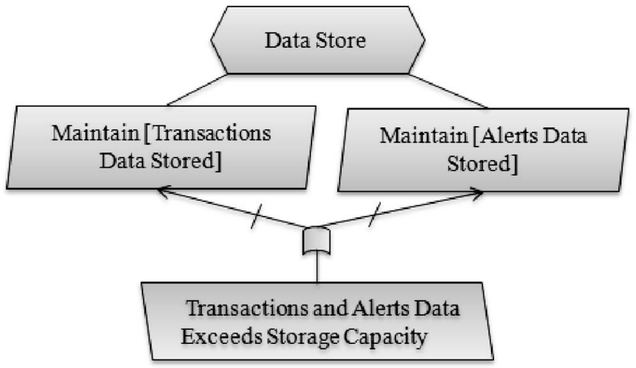
Data Store
Maintain [Transactions Data Stored]
Maintain [Alerts Data Stored]
Transactions and Alerts Data Exceeds Storage Capacity

Fig. 3. Scalability obstacle to multiple goals.

single goals described in the beginning of this section would have generated two obstacles, Transactions Data Exceeds Storage Capacity and Alerts Data Exceeds Storage Capacity. However, the logical conjunction of these two obstacles is *not* equivalent to the joined obstacle Transactions and Alerts Data Exceeds Storage Capacity.

The pattern of scalability obstruction to multiple goals suggests the following heuristic procedure for the systematic identification of scalability obstacles from goals.

For each agent,

1. identify the set of all goals assigned to the agent;
2. for each goal, identify what defines the goal load and what agent resources are involved in its satisfaction;
3. for each agent resource Res_i identified in step 2, take all goals $G_1, \ldots, G_n$ requiring this resource and consider a scalability obstacle of the form $\diamond\, (Load_{G_1} + \cdots + Load_{G_n} > M)$ obstructing these goals, where M is the minimum value of the agent resource needed to satisfy all goals together.

4.2 Assessing Scalability Obstacles

Once potential scalability obstacles have been identified, their criticality and likelihood should be assessed. As for other kinds of obstacles, the criticality of a scalability obstacle depends on its impact on higher-level goals and the importance of those goals.

A lightweight technique to support this process consists of using a standard *qualitative risk analysis matrix* in which the likelihood of a scalability obstacle is estimated on a qualitative scale from *Very Unlikely* to *Almost Certain* and its criticality estimated on a scale from *Insignificant* to *Catastrophic* [8]. The goal refinement graph helps in estimating obstacle criticality by allowing one to follow goal-refinement links upward to identify all high-level goals affected by an obstacle.

If there exists a quantitative goal model relating quality variables on software requirements and domain expectations to higher level goals [23], then this model can be used to perform a more detailed quantitative analysis of the impact of obstacles on higher level goals. However, developing such quantitative models requires more effort, and a detailed quantitative analysis is not always needed as the main objective of the obstacle assessment step is to separate scalability obstacles for which resolutions need to be sought from those whose risk is so low that they can safely be ignored.

4.3 Resolving Scalability Obstacles

Scalability obstacles whose combined likelihood and criticality are considered serious enough must be resolved. The obstacle resolution process comprises two activities: the *generation* of alternative resolutions and the *selection* of resolutions among generated alternatives. Only the generation step is considered here; the selection among alternatives

TABLE 3
Scalability Obstacle Resolution Tactics

Strategy	Short Description	Scalability Obstacle Resolution Tactic
Goal Substitution	Resolve the obstacle by finding alternative goals in which the obstructed goal is no longer needed	Can be applied without specialization
Agent Substitution	Resolve the obstacle by changing the responsibility assignment for the obstructed goal	• Transfer goal to non-overloaded agent ∘ Automate responsibility of overloaded human agent • Split goal load among multiple agents ∘ Split goal load into subtasks ∘ Split goal load by case
Obstacle Prevention	Introduce new goals and assumptions to prevent the obstacle from occurring	• Introduce scaling assumption • Introduce scalability obstacle prevention goal ∘ Set agent capacity according to load - Set agent capacity upfront to worst-case load - Adapt agent capacity at runtime according to load - Increase number of agent instances - Increase the capacity of the agent instance ∘ Limit goal load according to agent capacity - Limit goal load according to fixed agent capacity - Distribute goal load over time - Limit goal load according to varying agent capacity
Obstacle Reduction	Introduce new goals to reduce the obstacle's likelihood	• Influence goal load distribution
Goal Weakening	Change the definition of an obstructed goal to make it more liberal so that the obstruction disappears	• Weaken goal definition with scaling assumption • Weaken goal definition by strengthening scaling assumption • Weaken goal objective function ∘ Relax real-time requirement ∘ Relax required level of satisfaction
Goal Restoration	Introduce a new goal to restore the satisfaction of the obstructed goal when violated	Can be applied without specialization
Obstacle Mitigation	Introduce a new goal to mitigate the consequences of an obstacle when it occurs	Can be applied without specialization
Do-Nothing	Leave the obstacle unresolved	No specialization needed

can be performed using qualitative or quantitative techniques covered elsewhere [8], [18], [23].

Ideally, obstacle resolutions should be defined independently of any technology adopted to develop the system. Doing so allows the requirements analyst to explore a broader range of resolutions and to negotiate goals with the stakeholders. For example, the resolution *adapt agent capacity at runtime according to load*, described in Section 4.3.3, predicts the future value of the goal load at runtime and adjusts the agent capacity accordingly. Later in the development lifecycle this resolution could be implemented using, say, a cloud, in which the agents would be deployed as cloud-based services that can be replicated to support a growing load. It is important to note, however, that the decision of which agents would be cloud based and which requirements they would be responsible for is a subsequent activity to our method and should not be considered in the generation of alternative resolutions described in this section.

In Kaos, obstacle resolution tactics are model transformation operators that resolve an obstacle by introducing new goals, assumptions, and responsibility assignments in the model, or by modifying existing ones. We have extended the existing catalog of obstacle resolution tactics [20] with new, specialized tactics to deal with scalability obstacles. Our tactics have been developed by systematically considering specializations of the eight general obstacles resolutions strategies in [20]: goal substitution, agent substitution, obstacle prevention, goal weakening, obstacle reduction, goal restoration, obstacle mitigation, and do-nothing.

The benefit of these tactics is to encode expert knowledge about how to deal with scalability risks in a form that allows requirements analysts to explore systematically during requirements elaboration the range of alternative ways to deal with such risks.

Table 3 provides an overview of the scalability obstacle resolution tactics. The first and second columns show the general obstacle resolution categories and descriptions, as defined by Lamsweerde and Letier [20]. The third column shows the specialized tactics and subtactics for resolving scalability obstacles.

A complete, formal description of these tactics can be found in Duboc [33]. This section presents the main idea of each tactic. For each of the general obstacle resolution strategies, we briefly discuss its relevance to resolving scalability obstacles and, where appropriate, define new model transformation tactics that are specific to the resolution of scalability obstacles. The Do-Nothing strategy consists simply of leaving the obstacle unresolved and will not be considered further. This tactic should be applied only when the likelihood and criticality of the obstacle are considered low.

4.3.1 Goal Substitution

A first effective way to resolve an obstacle to some goal is to eliminate the obstacle completely by finding an alternative to the obstructed goal. For scalability obstacles, the principle is to try to find an alternative way to satisfy the system's higher-level goals in which the scalability risk no longer exists.

Example. Consider the scalability obstacle Batch Size Exceeds Alert Generator's Processing Speed obstructing the goal Achieve [Batch Processed Overnight]. An application of the goal substitution strategy in this case is to design a system in which transactions are processed continuously, 24 hours per day, rather than in overnight batches. This is in fact a strategy that has been developed for a later version of the IEF, which now supports both modes of operation.

4.3.2 Agent Substitution

This tactic consists of eliminating the possibility of an obstacle occurring by assigning the responsibility for the obstructed goal to another agent. The following specializations of this tactic are useful for resolving scalability obstacles:

- **Transfer goal to nonoverloaded agent.** This tactic consists of transferring responsibility for one or more of the goals G_i assigned to an overloaded agent Ag to an alternative agent Ag' with bigger capacity or smaller load. Possible choices for the alternative agent can be explored systematically by considering all agents already present in the model. This tactic may lead also to the introduction of a new agent role. A particular case is the commonly used tactic **automate responsibility of overloaded human agent**.
- **Split goal load among multiple agents.** This tactic consists of refining a goal G assigned to an overloaded agent Ag into subgoals G_i with smaller loads such that each subgoal can be assigned to a different agent with enough capacity to satisfy it. An application of this tactic may lead to the introduction of new agent roles taking responsibilities for the subgoals G_i. Two variants are the specialized tactics **split goal load into subtasks** and **split goal load by case** that consist in generating the subgoals for G using a milestone-driven or a case-driven formal refinement pattern, respectively [34]. In the first case, a goal of the form Achieve [Target Condition From Current Condition] is refined into two subgoals Achieve [Milestone From Current Condition] and Achieve [Target Condition From Milestone], each being assigned to separate agents with sufficient capacity to perform the tasks required for the satisfaction of each subgoal. In the second case, a goal G is refined into subgoals of the form G When Case_i, where the set of conditions Case_i covers an exhaustive set of cases in which G must be satisfied. Each subgoal is then assigned to a separate agent with sufficient capacity to support it.

Example. To illustrate these tactics, consider the goal Achieve [Alerts Investigated and Acted Upon Quickly] assigned to the class of agent Fraud Investigator, and the scalability obstacle Number of generated alerts exceeds fraud investigators' capacity. The tactic **split goal load into subtasks** suggests identifying subgoals corresponding to subtasks that can be assigned to different agents in the system, such as refining the goal Achieve [Alerts Investigated and Acted Upon Quickly] into two separate goals Achieve [Alerts Investigated Quickly] and Achieve [Fraudulent Transaction Acted Upon Quickly], each assigned to a different agent with a specialized role. The former goal is assigned to the agent Fraud Investigator, while the latter is assigned to the agent Collector. This division is, in fact, adopted by some of Searchspace's customers. The tactic **split goal load by case** suggests identifying a set of alternative cases of alert investigation that could be assigned to fraud investigators who specialize in such cases and are therefore more efficient to deal with them (e.g., based on the type of alert and account holder).

4.3.3 Obstacle Prevention

Obstacle prevention consists of introducing to the model a new assertion $\neg O$ that prevents the obstacle from occurring. Alternative obstacle prevention tactics make this new assertion either a domain assumption to be satisfied by some agent in the environment, or a goal whose satisfaction involves the software-to-be [20].

For a scalability obstacle of the form Goal Load Exceeds Agent Capacity, the following obstacle prevention tactic can be used:

- **Introduce scaling assumption.** This model transformation tactic has already been defined in Section 3.2. It prevents a scalability obstacle from obstructing a goal G by 1) introducing a scaling assumption SA such that its definition implies the negation of the obstacle, and 2) using this scaling assumption SA to refine the obstructed goal G into G When SA.
- **Introduce scalability obstacle prevention goal.** This tactic consists in preventing the scalability obstacle from occurring by generating a new goal of the form:

Goal Avoid [Goal Load Exceeds Agent Capacity]
Formal Spec $\Box \neg (Load_G > Capacity_{Ag})$

Alternative refinements for this goal can then be generated using the refinement tactics described below. These tactics differ according to whether the obstacle prevention goal is achieved through constraining the agent capacity or the goal load. In both cases, one can consider a static tactic or a dynamic one.

- **Set agent capacity according to load.** This tactic consists of refining the above goal into a scaling assumption defining a bound X on the goal load (i.e., $\Box\,(Load_G \leq X)$), and a goal requiring the agent capacity to have a value above X (i.e., $\Box\,(Capacity_{Ag} > X)$. It has the following static and dynamic specializations:

- **Set agent capacity upfront to worst-case load.** For this specialization, X defines a fixed worst-case bound on the goal load, that is, $\Box\,(Load_G \leq WorstCaseLoad)$, and the goal requires the agent capacity to have a constant value above this worst-case load, that is, $\Box\,(Capacity_{Ag} > WorstCaseLoad)$.
- **Adapt agent capacity at runtime according to load.** This tactic is the dynamic counterpart of the previous one. In this tactic, X defines a time-varying variable representing the predicted future value of the goal load at runtime (i.e., $\Box\,(Load_G \leq ExpectedLoad)$), and the goal dynamically extends the agent capacities based on the predicted load (i.e., $\Box\,(Capacity_{Ag} > ExpectedLoad)$).

Two typical tactics to satisfy the subgoal in both cases are to **increase the number of agent instances** so that their increased capacity is greater than X, or **increase the capacity of the agent instances** so that it becomes greater than X. In the IEF, for example, this could be done by increasing the number of Fraud Investigators processing alerts or by providing training to investigators so that alerts can be processed more quickly.

- **Limit goal load according to agent capacity.** This tactic consists of refining the obstacle prevention goal into a requirement or expectation of the form $\Box\,(Capacity_{Ag} \geq K)$ requiring the agent capacity to be greater than some value K, and a subgoal of the form $\Box\,(Load_G \leq K)$ limiting the goal load value at any given time below K. This tactic also has two specializations:
  - **Limit goal load according to fixed agent capacity.** For this tactic, K is a constant value representing a fixed agent capacity. One typical tactic to satisfy this subgoal is to **distribute the goal load over time**; for example, for a student enrollment system, different registration dates can be set for different student groups.
  - **Limit goal load according to varying agent capacity.** This tactic is the dynamic counterpart of the previous one, where K is a time-varying variable. The principle here is to monitor or predict variations in the agent capacity at runtime and dynamically change the limit on the maximal goal load based on these variations.

Example. To illustrate the use of these tactics for the exploration of alternative system designs, consider again the obstacle Batch Size Exceeds Alert Generator Processing Speed and its prevention through the goal Avoid [Batch Size Exceeds Alert Generator Processing Speed]. We consider each of the above goal refinement tactics in turn.

1. The tactic **set agent capacity upfront to worst-case load** refines the goal into a scaling assumption Batch Size Below Max (where Max is a constant value) and a subgoal Maintain [Alert Generator Processing Speed Above Max] such that the alert generator would have a fixed capacity designed to deal with the largest possible batch size.
2. The tactic **adapt agent capacity at runtime according to load** refines this goal into the subgoals Maintain [Accurate Batch Size Prediction] and Maintain [Alert Generator Processing Speed above Predicted Batch Size]. Satisfaction of these adaptation goals can be assigned as the responsibility of human agents who have to manually upgrade the system, but they also could be partly automated by developing a self-managed system that would monitor and predict batch sizes and automatically allocate more resources as needed.
3. The tactic **limit goal load according to fixed agent capacity** refines the goal into a requirement Maintain [Alert Generator Processing Speed Above K] (where K is a constant value)—this can be translated to a hardware requirement on machines running the IEF—and a subgoal Maintain [Submitted Batch Size Below K] requiring the size of the submitted batch to remain within the alert generator capacity. There are, in turn, different ways in which this goal could be satisfied. It could, for example, be assigned as the responsibility of the Bank System, or we could introduce an additional agent between the Bank System and Alert Generator responsible for truncating batches of transactions that are too large and for rescheduling the remaining transactions to a later time or redirecting them to another system.
4. The tactic limit goal load according to varying agent capacity is the dynamic counterpart of the above refinement in which the limit on the batch size submitted to the Alert Generator would vary at runtime based on observations of the actual processing speed of the Alert Generator (as opposed to a fixed processing speed that was estimated and defined once when the system was deployed).

These alternative refinements would be modeled as OR-refinement links in the goal graph structure. Each alternative would have to be evaluated in terms of cost, risks, and its contribution to high-level goals. Such an evaluation would then be used to decide which alternative or combination of alternatives to select for implementation.

4.3.4 Obstacle Reduction

Obstacle reduction consists of introducing a new goal aiming at reducing the likelihood of the obstacle occurring. Typically, this technique plays on factors that encourage or dissuade human agents to behave in certain ways. For scalability obstacles, one common obstacle reduction tactic is to **influence the distribution of goal load**. Consider, for

TABLE 4
Goal Deidealization Patterns for Scalability Obstacles

Goal	Scalability Obstacle	Deidealized Goal
$G : \Box P$	$\Diamond\,(Load_G > Capacity_{Ag})$	$\Box\,(Load_G \leq Capacity_{Ag} \rightarrow P)$
$G : \Box\,(SA \rightarrow P)$	$\Diamond\,(SA \wedge Load_G > Capacity_{Ag})$	$\Box\,(SA \wedge Load_G \leq Capacity_{Ag} \rightarrow P)$

example, an e-commerce website wishing to announce a sale. It can send notification e-mails to registered customers on different dates rather than all on the same date, assuming that customers are more likely to shop as soon as they receive the e-mail. Another typical example is the tactic used in some large office buildings to reduce morning congestion in their elevators by spreading out employees' start times on different floors.

4.3.5 Goal Weakening

Sometimes a goal is found to be obstructed by an obstacle simply because the initial goal definition is too strong. In this case, one way to resolve the obstruction is to *deidealize* the goal, that is, to change the goal definition to make it more liberal so that the obstruction is no longer present. This process is supported by a set of formal and informal goal transformation tactics [20]. We have defined the following new transformation tactics for resolving scalability obstacles through goal weakening.

- **Weaken goal definition with scaling assumption.** This tactic transforms an obstructed goal into a more liberal goal, requiring the original goal to be satisfied only when its load does not exceed the agent capacity. It should be applied when a goal G that does not refer to any scaling assumption is obstructed by a scalability obstacle Goal Load Exceeds Agent Capacity. In this tactic, the original goal is transformed into the goal named G When Goal Load Does Not Exceed Agent Capacity, where the name corresponds to a scaling assumption. The formal transformation pattern associated with this tactic is given by the first row in Table 4. It states that a goal of the form $\Box\, P$ obstructed by a scalability obstacle $\diamond\,(\mathrm{Load}_G > \mathrm{Capacity}_{Ag})$ is transformed into a goal of the form $\Box\,(\mathrm{Load}_G \leq \mathrm{Capacity}_{Ag} \rightarrow P)$.

Note that the definition of the resulting goal is the same as the definition of the subgoal that would be obtained by applying the tactic **introduce scaling assumption**. The difference is in terms of how the weakening is propagated, as we illustrate on the example below.

- **Weaken goal definition by strengthening the scaling assumption.** This tactic has the same objective as the one above, but it should be applied when the obstructed goal already refers to a scaling assumption—that is, when a scalability goal G When SA referring to a scaling assumption SA is obstructed by an additional scalability obstacle. Applying this tactic transforms the original goal into the goal G When Goal Load Is Within SA and Does Not Exceed Agent Capacity. The formal transformation pattern associated with this tactic is given by the second row in Table 4.

Once a goal definition has been weakened with one of the two tactics above, the change needs to be propagated along the refinement links in the goal graph following the general procedure described by Lamsweerde and Letier [20]. In the case of these tactics, the propagation typically is performed by weakening the definitions of parent goals using the same patterns in Table 4.

- **Weaken goal objective function.** This tactic involves weakening the goal's objective function or required levels of satisfaction so that it requires less agent capacity for its satisfaction. For Achieve goals of the form $\Box(C \rightarrow \diamond_{\leq d} T)$, a specialized tactic for goal weakening is to **relax real-time requirement** by increasing the time d for achieving the condition T. The **relax required level of satisfaction** specialization involves relaxing the minimum value to be achieved in order for the goal objective function to be maximized (i.e., the target values of the goal's objective function).

Example. Applying the tactic **weaken goal definition with scaling assumption** to the goal Achieve [Batch Processed Overnight] generates the weaker goal Achieve [Batch Processed Overnight When Batch Size Does Not Exceed Alert Generator's Speed]. This change would be propagated upward in the goal model of Fig. 1 to yield the goal Achieve [Alert Generated When Batch Size Does Not Exceed Alert Generator's Speed]. The purpose of considering such a tactic is to systematically explore the space of possible resolutions. In this example, it is likely that such resolutions will not be acceptable to system stakeholders. Consider also the goal Achieve [Batch Processed Overnight Under Expected Batch Size Evolution] (specified in Section 3.2). This goal is specified with the following fixed objective function:

Objective Functions At least 9 out of 10 batches should be processed within 8 hours, and all batches should be processed within 9.6 hours.

Name	Definition	Target
% of Batches Processed in 8 hours	Pr(processingTime $\leq$ 8h | ExpectedBatchSizeEvolution)	90%
% of Batches Processed in 9.6 hours	Pr(processingTime $\leq$ 9.6h | ExpectedBatchSizeEvolution)	100%

Assume that the scalability obstacle Expected Batch Size Evolution Exceeds Alert Generator Processing Speed obstructing this goal is considered likely to occur. In order to resolve the obstacle, the tactic **relax real-time requirement** would weaken the goal by increasing the expected processing time from 8 to 10 hours. Alternatively, the tactic **relax required level of satisfaction** would weaken the goal by allowing for less than 90 percent of the batches to be processed in 8 hours.

4.3.6 Goal Restoration and Obstacle Mitigation

In some cases, it may be impossible or too costly to guarantee that all scalability obstacles will be avoided. One should therefore consider how to tolerate a scalability obstacle or mitigate its consequences when it occurs. Two general tactics here are goal restoration and obstacle mitigation.

In goal restoration, a new goal of the form $\Box\,(\neg P \rightarrow \diamond P)$ is added to the model to eventually restore the condition P of the obstructed goal. In obstacle mitigation, a new goal is added to attenuate the consequences of an obstacle occurrence. This typically involves ensuring the satisfaction of a weaker version of the obstructed goal or of an ancestor of the obstructed goal. Consider, for example, a call center with the goal Achieve [Calls Answered Quickly] stating that calls should be answered in less than 3 minutes. If, in a given day, the call center receives an unusually high number of calls, this goal is obstructed by the scalability obstacle Number of Calls Exceeds Customer Service Team Capacity. A tactic to mitigate this obstacle might be to create a new goal Achieve [Return Call] which ensures a weaker version of the obstructed goal stating that a call must be taken in less than 3 minutes or else the details of the customer should be recorded and the call returned in less than half an hour.

5 Case Study

We have applied the scalability obstacle analysis techniques described in the previous section to the elaboration of the scalability requirements of a major redesign of the IEF system. One of the main objectives of this redesign was to extend the existing batch-based system with functionality to process transactions in a continuous stream.

This case study follows a previous case study reported by Duboc et al. [5] in which we elaborated a KAOS goal model for the batch-based version of IEF. The first case study occurred before our development of the scalability obstacle analysis techniques. The elaboration of the goal model for the new system and its subsequent scalability obstacle analysis were performed by the first author by invitation of the IEF project manager after the success of the first case study. Having worked at Searchspace for three and a half years, she had good knowledge of the application domain and previous system versions.

Unlike a retrospective study of a fully implemented system with well-understood requirements, this study involved a system under development, with all the complexities normally present during the initial elaboration of a system's requirements. Stakeholders were busy with their own daily tasks and had limited time for requirements elicitation and validation; knowledge about the system was spread across individuals; many assumptions originated from experience with previous projects that may not have been valid for the new system; documents were contradictory and incomplete; and many requirements had no clear rationale other than being present in older versions of the IEF.

Our case study lasted for a period of approximately 30 working days, including interviews, brainstorming sessions, modeling, data characterization, and document writing and reviewing. During the first two days, the analyst built an initial high-level model based on her own domain knowledge to serve as a starting point for discussions. The next 25 days were spent elaborating the initial, idealized goal model and applying the scalability obstacle identification and assessment activities described in the previous sections. The last 5 days were spent exploring obstacle resolutions and producing the final documentation.

The initial goal model was elaborated from a number of sources: the goal model for the previous batch-based system, a business requirements specification that had already been written for the new system, and information gathered from interviews with project stakeholders during the goal model elaboration. The initial business requirements specification document contained mostly functional requirements specified through use cases, and a single unjustified estimation of the number of transactions the system should be able to process per day. The IEF developers judged this document insufficient to inform the software design adequately.

The full goal model that we elaborated cannot be shown here due to its size and the sensitivity of the information it contains. Sanitized extracts from the model can be found in Duboc [33]. A portion of the goal model on which we illustrate the scalability obstacle analysis techniques is shown in Figs. 5 and 7 in the Appendix.

5.1 Identifying Scalability Obstacles

We followed the process defined in Section 4.1 to identify scalability obstacles systematically from goals assigned to agents. Table 5 illustrates the result of this process by showing the scalability obstacles generated from some of the responsibility assignments included in the goal model of Appendix A.

In this table, the Alert Generator agent is shown to be responsible for three goals: Achieve [Batch Processed Overnight], Achieve [Real-time Transactions Processed Instantaneously], and Achieve [Fraud Patterns Tested on Past Transactions]. The first goal is similar to the one defined in Section 2.2, the second goal bounds the processing time for transactions that require immediate clearing, and the third goal requires that every new fraud detection rule should be tested against past transactions.

For these three goals, we have defined the measure of the goal load to be the number of *transaction checks* (txn check), where a transaction check corresponds to the application of a single fraud detection rule to a single transaction. The load imposed by these goals on the Alert Generator is therefore proportional to the number of transactions to be checked and the number of fraud detection rules to be applied on each transaction. We thereby obtained the scalability obstacle Number of Txn Checks Exceeds Alert Generator Processing Speed obstructing these three goals. In defining this

TABLE 5
Scalability Obstacles to IEF Goals

Agent	Goal	Scalability Obstacle
Alert Generator	Achieve [Batch Processed Overnight] Achieve [Real-time Transactions Processed Instantaneously] Achieve [Fraud Patterns Tested on Past Transactions]	Number of Txn Checks Exceeds Alert Generator Processing Speed
Data Store	Achieve [Incoming Transactions Stored] Achieve [Alerts Stored] Achieve [Fraud Detection Rules Stored]	Amount of Data Exceeds Data Storage Space
Fraud Investigator	Achieve [Alerts Investigated and and Acted Upon Quickly]	Number of Alerts Exceeds Fraud Investigator Speed

scalability obstacle, we further took into consideration the fact that not all fraud detection rules take the same time to compute. For example, fraud detection rules that consider historical information about a set of related accounts take much longer to process than simple comparisons of transactions against blacklisted account numbers.

Other domain characteristics, such as the percentage of new accounts in a daily batch or stream, also influence the goal load and therefore were included in the definition of this scalability obstacle. Thus, unlike our simple illustrative example in Section 4.1 where the load of the goal Achieve [Batch Processed Overnight] was a simple scalar value (the number of transactions), in our full model this load is defined as a vector composed of several domain characteristics. Significant knowledge about the application domain and fraud detection rules was necessary to identify such additional characteristics.

As shown in Table 5, the obstacle identification process also led us to identify scalability obstacles to agents other than the Alert Generator. In particular, the number of generated alerts and the time needed to handle an alert are important factors whose variations could prevent Fraud Investigators from satisfying their goals.

In our model, the Data Storage agent is separate from the Alert Generator because it corresponds to a given domain component the Alert Generator must interact with. From the goals shown in Table 5, we generated the scalability obstacle Amount of Data Exceeds Data Storage Space whose definition refers to domain quantities such as the number and size of transactions and generated alerts.

Our systematic identification of scalability obstacles led us to uncover 13 important domain characteristics that have an impact on the system's scalability. All these characteristics except one had been overlooked in the initial business requirements specification, and their impact on scalability had never been made fully explicit to the main system stakeholders. The benefit of exposing these scaling characteristics became apparent when eliciting scaling assumptions on these variables and discovering that stakeholders had diverging views about the ranges of values such variables may take, and therefore about the software scalability requirements that depend on these assumptions.

Two important observations can be made from our experience in identifying scalability obstacles for the IEF system:

1. *The completeness of the generated scalability obstacles is relative to the completeness of the goal model.* An important concern when identifying scalability obstacles is to try to identify *all* domain quantities whose variations have an impact on the system's ability to satisfy its goals. The systematic process in which scalability obstacles are generated for each agent and every goal assigned to it ensures completeness of the identified obstacles with respect to specified software requirements and domain expectations. By completeness here we mean that a scalability obstacle of the form Goal Load Exceeds Agent Capacity has been considered for all leaf goals and agent resources that have been identified in the model. This completeness is of course relative to the quality of the initial goal model—if some important agents, requirements, or domain expectations are missing, it is likely that their scalability obstacles will be overlooked as well.
2. *Detailed domain knowledge is needed to define fine-grained scalability obstacles.* Defining what constitutes the load of a particular goal was not always straightforward and could not be derived simply from the goal definition alone. It often required a good knowledge of the application domain and of the factors affecting the computational complexities of the functions to be performed for satisfying the goals. This was illustrated above for the goals assigned to the Alert Generator. In our example, it required good knowledge of the complexities of the different alert generation rules used in fraud detection systems. When this knowledge is not available, scalability obstacles still can be identified from the goal definition, but the scalability obstacles as defined remain much more coarse grained. For example, without expert knowledge, our scalability obstacles to the Alert Generator's goals would have been defined in terms of numbers of transactions only, as was done in Section 4.1. Similarly, the scalability obstacles we have identified for the goals assigned to the Fraud Investigator agents remain quite coarse grained because during our case study we lacked access to the knowledge of what domain characteristics affect the speed at which alerts are investigated. A benefit of our scalability obstacle identification process is that it structures the elicitation activities

TABLE 6
Criticality of Scalability Obstacles

Scalability Obstacle	Likelihood	Criticality
Number of Txn Checks Exceeds Alert Generator Processing Speed	High	High
Amount of Data Exceeds Data Storage Space	High	High
Number of Alerts Exceeds Fraud Investigator Speed	High	Moderate

that need to be performed for identifying detailed scalability obstacles.

5.2 Assessing Scalability Obstacles and Eliciting Scaling Assumptions

Once identified, scalability obstacles need to be assessed in terms of their likelihood and criticality so that decisions can be made about which obstacles need to be resolved. Since obstacle analysis is an iterative process, scalability obstacles need to be reassessed after their resolutions. During this reassessment, obstacle likelihoods and criticality are used also to guide the selection among alternative resolutions.

In a first assessment, we considered all identified scalability obstacles equally likely to happen. Since we had no information about the potential scaling characteristics of the various domain quantities, we pessimistically assumed that they all could rise to levels that would lead to violations of idealized goals.

The criticality of the identified scalability obstacles was estimated by the first author and the project manager by identifying which higher level goals would be violated if the obstacles were to occur. Table 6 shows this qualitative assessment for three obstacles.

After this first assessment, we started eliciting scaling assumptions for all domain quantities involved in the scalability obstacles and updated the goal model by applying the tactic **introduce scaling assumption** to the obstructed goals. This tactic generated new obstacles that had to be assessed as well. We did not yet consider other scalability obstacle resolution tactics because we felt that we needed first to understand the scaling assumptions of the variables involved before investigating other resolution tactics.

For example, the goal Achieve [Batch Processed Overnight] was refined into the scaling assumption and scalability requirement shown in Fig. 4.

The figure also shows new obstacles to the scaling assumption and the scalability requirement. Assessing the

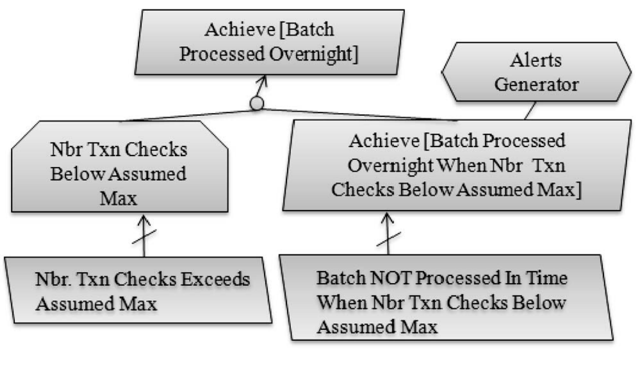
Achieve [Batch Processed Overnight]
Alerts Generator
Nbr Txn Checks Below Assumed Max
Achieve [Batch Processed Overnight When Nbr Txn Checks Below Assumed Max]
Nbr. Txn Checks Exceeds Assumed Max
Batch NOT Processed In Time When Nbr Txn Checks Below Assumed Max

Fig. 4. Obstacles to a scaling assumption and a scalability requirement.

likelihoods of these two obstacles consists of 1) asking how likely it is that the number of transaction checks in a batch will exceed the assumed maximum value, and 2) how likely it is that the batch will not be processed in time despite the fact that the number of transaction checks is below the assumed maximum value, respectively.

We also asked developers to estimate how feasible it would be to develop a system that would meet the scalability requirements given these scaling assumptions. When there was a perceived risk that the scalability requirements may not be achievable, this signaled the need to investigate additional resolutions for the scalability obstacles to these requirements. The exploration of these resolutions will be discussed in the next section.

From our experience in assessing scalability obstacles, we can make the following observations:

1. *Assessing scalability obstacle likelihoods and eliciting scaling assumptions are strongly related activities.* The standard identify-assess-resolve loop of obstacle analysis assumes that obstacle likelihoods can be assessed before deciding on their resolution. This proved to be impossible for the first assessment of scalability obstacles because assessing how likely a scalability obstacle is to occur requires knowledge of the scaling characteristics of the domain quantities concerned by the obstacles. For this reason, we deviated from the standard obstacle analysis process and combined the tasks of assessing scalability obstacles and eliciting scaling assumptions as described above.
2. *Another difficulty we faced was that the capacity of some agents is still unknown when the obstacle likelihood needs to be assessed.* Indeed, one of the purposes of the requirements engineering process is to decide what the required capacities for these agents should be. Such analysis should take into consideration the varying likelihood of scalability obstacles to different envisioned agents capacities. For example, the scalability obstacle Batch NOT Processed In Time When Nbr Txn Checks Below Assumed Max in Fig. 4 required asking IEF developers to assess the likelihood of this obstacle occurring for feasible Alert Generator capacities (i.e., its processing speed). However, we feel that a technique for more thorough analysis of the variation of obstacle likelihood with respect to the variation of feasible agent capacities is needed, and this is a subject of future research.
3. *Eliciting and reaching agreements on the projected values for domain quantities in the scaling assumptions proved to be the most difficult and time consuming tasks of our case study.* This was due to the uncertainties and

divergence of opinions among stakeholders about future values for some of the domain characteristics, and to large variations in the customer base that led people to make different (often implicit) assumptions. There was also a lack of awareness by nonsoftware engineers of the impact that some of the projected values would have on the software design, leading sometimes to infeasible or extremely costly requirements due to unrealistically high scaling characteristics. To help resolve these divergences, we created a simple quantitative goal model [23] relating projected values for the domain quantities involved in scaling assumptions (e.g., the expected growth rate in the number of accounts and number of transactions per account) to scalability requirements for the software system (e.g., in terms of throughput and storage). This model was used during stakeholder meetings to help reach agreement on the scaling assumptions.

4. *The qualitative assessment of the scalability obstacles' criticality was straightforward and aided by the goal model structure.* This simply consisted of following goal-refinement links bottom-up from the obstructed goal to higher-level business goals.
5. *Qualitative versus quantitative assessment of scalability obstacles.* Our assessment of scalability obstacles remained entirely qualitative. A finer-grained quantitative assessment of the likelihood of violations of scaling assumptions was felt to be too uncertain to be of value. Accurate quantitative assessments of the likelihood of meeting the scalability requirements are also difficult to obtain. These potentially could be obtained by analyzing models of the intended software architecture or by testing system prototypes, but such analyses were not performed. A qualitative assessment of scalability obstacles was sufficient for our purposes of identifying what the important scalability obstacles are and how these can be resolved through the introduction of new goals and requirements. We note that similar observations about the limitation of quantitative assessments and benefits of qualitative assessments has been made about the use of fault tress for safety critical systems [11]. However, our case study did not involve detailed comparisons of alternative system designs for which a more precise, quantitative assessment of scalability obstacles might be needed. This is an area where further research is required.

5.3 Resolving Scalability Obstacles

For resolving scalability obstacles, we have used the catalog of scalability obstacle resolution tactics in Section 4.3 to explore systematically some potential resolutions of a few key scalability obstacles that were judged to be critical. Some of the resolutions we identified correspond to design decisions that had been taken independently of our study by Searchspace for the new system version. Other resolutions correspond to alternative and additional requirements that had not been considered or had been considered but not retained. For the case study, we focused on generating a wide range of resolutions for the identified scalability obstacles. We did not perform detailed evaluations of the generated alternatives in order to guide the selection of the preferred ones.

The process of generating resolutions to scalability obstacles consists of systematically considering the potential application of each resolution tactic in Table 3 to each obstacle. To illustrate this process, Tables 7 and 8 illustrate the set of model transformations that can be applied to resolve the obstacles listed in Table 5. Some of the goal specifications and goal graphs obtained by these model transformations can be found in Duboc [33]. The identify-assess-resolve loop of obstacle analysis is iterative. Once a resolution tactic has been selected, one needs to identify *its* obstacles and explore new obstacle resolution tactics. In Tables 7 and 8, we only discuss resolutions for the original obstacles.

A few observations can be made from our experience in resolving scalability obstacles for the IEF system:

1. *We felt that the biggest benefit of the catalog of obstacle resolution tactics was in encouraging a systematic exploration of a wide range of alternative options for resolving a scalability obstacle.* Although we did not do this during this case study, we believe that such a catalog can be used during meetings with stakeholders, domain experts, and system designers to facilitate the exploration of creative design solutions to envisioned scalability risks. Further studies, such as those explored by Maiden and Robertson [35], would be needed to investigate the effectiveness of scalability resolution tactics in this setting.
2. *A question we faced was at which level of detail to specify the models generated by the application of the resolution tactics.* In practice, when exploring alternative resolutions, we did not specify all alternative models explicitly as the enunciation of the applied resolution tactic is in most cases enough to understand what the model would be. We felt it would be sufficient to specify the actual requirements and domain assumption only once an alternative was selected. We did not specify detailed formal specifications for the resulting new goals because this was not felt to be of benefit to the project. However, the formal descriptions of the model transformations were helpful to clarify and disambiguate what each resolution tactic could achieve.

5.4 Summary

Our case study produced valuable results to Searchspace. Project managers easily understood and accepted the basic concepts of our approach and were particularly impressed by the ability of the model to reveal the impact of distinct projected scaling bounds on leaf goals and to expose conflicting requirements. The goal model provided a rationale for low-level requirements and, in particular, it allowed a more precise and justifiable 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. One of the tools that enabled us to elaborate goals and estimate bounds more precisely was to get an explicit statement of all underlying assumptions about the application domain. The final model

TABLE 7
Resolving the Scalability Obstacle Nbr of Txn Checks Exceeds Alert Generator Processing Speed

Resolution Strategy	**Model Transformations**
Goal Substitution	• *none*, because the obstructed goals are essential and have no alternatives
Agent Substitution	• **Transfer goal to non-overloaded agent:** Responsibility for the goal Achieve [Fraud Patterns Tested on Past Transactions] can be transferred from the Alert Generator to a Sandbox agent providing a separate environment for testing new fraud detection rules. • **Split goal load among multiple agents:** *none*; we could identify no subtasks or sub-cases for the obstructed goals that could be offloaded to other agents.
Obstacle Prevention	• **Introduce scaling assumption:** This tactic generates a series of scaling assumptions—on the numbers of transactions in a batch, distinct business entities in a batch, transactions per second in the transactions stream, distinct business entities per second in the transactions stream, transactions in testing data, distinct business entities in testing data, fraudulent patterns, fraudulent patterns added in the last day—together with the subgoals obtained by application of the refinement pattern. • **Introduce scalability obstacle prevention goal:** This tactic generates the new goal Avoid [Nbr of Txn Checks Exceeds Alert Generator Processing Speed]. Alternative refinements for this goal are as follows: ○ **Set agent capacity according to load:** The tactic **set agent capacity upfront to worst-case load** refines the goal into the assumption Assumed Worst-Case Nbr of Txn Checks and goal Maintain [Alert Generator's Speed above Worst-Case Nbr of Txn Checks], while the tactic **adapt agent capacity at runtime according to load** refines it into the subgoals Maintain [Accurate Prediction on Max Nbr of Txn Checks] and Maintain [Alert Generator's Speed above Predicted Max Nbr of Txn Checks]. ○ **Limit goal load according to agent capacity:** Applying the tactic **limit goal load according to fixed agent capacity** imposes fixed limits on the number of transactions that a bank can provide in its daily batches and continuous stream, while the tactic **limit goal load according to varying agent capacity** imposes similar limits but with bounds that can vary over time based on the capacities of the Alert Generator. Alternative ways of implementing these tactics have been described in Section 4.3.3.
Obstacle Reduction	• **Influence goal load distribution:** Pricing incentives can be introduced to influence the goal load (for example, by setting up tariffs that will influence the time at which transactions are submitted).
Goal Weakening	• **Weaken goal definition with scaling assumption** and **weaken goal definition by strengthening scaling assumption** consist in revisiting the scaling assumptions elicited during the scalability obstacles assessment step (see Section 5.2) so as to weaken the requirements on the Alert Generator. • **Weaken goal objective function:** This tactic consists in modifying the Alert Generator's requirements by relaxing its real-time requirement, relaxing its required levels of satisfaction, or both, as illustrated in Section 4.3.5.
Goal Restoration and Obstacle Mitigation	• These tactics led us to specify goals to be satisfied when the Alert Generator fails to satisfy the obstructed goals. These include the requirements to make generated alerts available to a Fraud Investigator even if a batch has not been processed entirely, and to finish processing an uncompleted batch as soon as possible. For the real-time transaction processing, it includes defining how to clear real-time transactions if the Alert Generator fails and how eventually to check for frauds in transactions that have been cleared during the Alert Generator's failure if it were to happen.

contains a number of goals that had been overlooked in Searchspace's initial requirements specification and replaces others that were idealized in that specification. The approach also helped to document and resolve numerous stakeholder disagreements. The goal-obstacle analysis process allowed us to expose a whole range of potential system failures, not only scalability-related ones. Searchspace has used the goal model to support the development of the new version of the IEF. Furthermore, the model can be used by them to aid a number of other activities, such as the derivation of test cases and the derivation of hardware requirements for their different clients based on their expected scaling characteristics.

6 Related Work

Scalability is commonly discussed in the context of specific technologies. Virtualization, for example, has been used for years and a number of studies have reported on the scalability of systems running in virtual environments [36], [37], [38]. Earlier, the advent of grid computing was driven by the need to share resources and sometimes to achieve high performance in collaborative problem-solving strategies which emerged in industry, science, and engineering [39]. Currently, cloud computing mechanisms, such as cloudbursting, allow companies to maintain their private cloud and leverage a public cloud to handle spikes in the processing requirements [40]. In fact, the idea of using

TABLE 8
Resolving the Scalability Obstacle Amount of Data Exceeds Data Storage Space

Resolution Strategy	Model Transformations
Goal Substitution	• *none*, although some of the goals obtained though goal weakening (see below) were first identified when looking for goal substitutions
Agent Substitution	• **Transfer goal to non-overloaded agent:** Responsibilities for storing the transactions and alert data can be transferred to an external data store. • **Split goal load among multiple agents:** This tactic consists in splitting the Data Store agent into multiple components, each of which can be responsible for storing part of the data. However, deciding how to organize the Data Store into multiple components is a design decision rather than a requirements one. The scalability requirements define the constraints that the Data Store must satisfy without prescribing what its architecture should be.
Obstacle Prevention	• **Introduce scaling assumption:** This tactic introduces scaling assumptions on the number and size of transactions, generated alerts, and stored fraud patterns, together with the corresponding scalability subgoals. • **Introduce scalability obstacle prevention goal:** This tactic generates the new goal Avoid [Amount of Data Exceeds Data Storage Space] for which we generate the following alternative refinements: ◦ **Set agent capacity according to load:** The tactic **set agent capacity upfront to worst-case load** is not applicable because the worst-case load in this case is infinite if we assume no end to the system's lifetime. The tactic **adapt agent capacity at runtime according to load** suggests that we plan for a gradual increase of the storage space during the system's lifetime. ◦ **Limit goal load according to agent capacity:** This tactic suggests increasing the alerting threshold so as to limit the number of generated alerts when storage space is insufficient.
Obstacle Reduction	• **Influence goal load distribution:** Influencing the number of bank transactions is out of scope and reducing this number would be against the company's interest. The number of false alerts could be reduced by influencing the behaviors of account holders (e.g., by allowing and encouraging them to provide information to the bank to reduce the risk of false alerts.)
Goal Weakening	• **Weaken goal definition with scaling assumption** and **weaken goal definition by strengthening scaling assumption** consist in revising the scaling assumptions and have not been considered. • **Weaken goal objective function:** The data storage goal can be weakened by limiting the amount of time during which transactions and alerts need to remain stored. This is one of the tactics implemented in the system. Many factors influence when transactions and alert data can be removed, notably legal requirements on the minimum time during which transactions and alert data must remain accessible.
Goal Restoration and Obstacle Mitigation	• If the obstacle cannot be completely avoided, back up plans may consist of having reserved storage space always on stand-by, erasing older data to allow for new data to be stored, or stopping all processing until the obstacle is resolved.

shared resources as needed has been around for decades [41] and yet the software engineering community struggles to develop scalable software. Such technologies may make it easier to implement software systems that satisfy scalability goals, but do not offer any intrinsic help in identifying, communicating, and analyzing these goals. Our approach elaborates scalability goals independently of the technology being used to fulfill them.

Works on scalability analysis recognize the need for precise scalability requirements. However, the great majority of them take the elicitation and specification of these requirements for granted [3], [4], [42], [43], [44]. Our method can be used to elaborate the models these techniques need as input.

The few works in this area that attempt to give some guidance on elaborating scalability requirements take an oversimplified view of the requirements engineering process. In particular, they tend to be limited to specifying the information that scalability requirements should contain [6], [7] or to providing only vague guidelines on the kinds of scenarios such requirements should cover [45], [46], [47]. Our approach provides a systematic approach for elaborating scaling assumptions, scalability goals, and scalability requirements from an initial, idealized goal model using scalability obstacle analysis.

Our definition of scalability [4] differs from other definitions. For example, Weinstock and Goodenough distinguish two main uses of the term scalability: 1) the ability to handle increased workload without adding resources to a system, and 2) the ability to handle increased workload by repeatedly applying a cost-effective strategy for extending a system's capacity [31]. These definitions allude to alternative strategies for satisfying (stated or unstated) scalability goals and the costs involved in doing so, rather than ways in which the scalability goals are identified and specified initially. In contrast, our framework distinguishes scalability goals (goals whose specification is related to one or more scaling assumptions) from the tactics for achieving these goals (specified as goal refinements and obstacle resolutions). Our resolution tactics cover both notions of scalability discussed by Weinstock and Goodenough.

Cost is an important quality that has strong interactions with scalability goals, as well as with all other goals of a system. In our goal-oriented framework, cost is a separate criterion that is used to *select* among alternative scalability obstacle resolution tactics, but it is not part of the definition of the scalability goals themselves. A view similar to ours is taken by Bahsoon and Emmerich, who relate costs and scalability in order to value the ranges in which a given software architecture can scale to support likely changes in load [3]. A different approach is taken by Jogalekar and Woodside, who define a scalability metric for distributed systems that takes into account throughput, stakeholder's value and running cost [48]. In future work, we intend to investigate how metrics such as this can be used to select among alternative scalability strategies.

Existing goal-oriented requirements engineering methods provide no specific support for eliciting, modeling, and reasoning about scalability requirements [8], [18]. As we summarize in the next section, the work described in this paper rectifies this gap.

7 Conclusion

Accurate scalability goals should be identified in the earliest stages of system development. Not only can this help system designers to take proper advantage of current technologies and to avoid costly mistakes, but it also can give them greater freedom to explore alternative system designs and system evolution paths for achieving scalability than if such concerns were left to later stages. However, identifying scalability goals in those early stages is not trivial, and to date little research has been done on how to elicit, model, and reason about them in a precise and systematic way. This is surprising given the importance of scalability for many software-intensive systems, and it contrasts with the extensive treatment received by other critical quality requirements such as performance, safety, and security.

To address this problem, we have presented precise definitions for the concepts of scalability goals and scaling assumptions in the context of the KAOS goal modeling framework. These definitions provide templates that requirements engineers can use to write precise and testable specifications of scalability goals. These specifications are testable in the sense that they are falsifiable by a counterexample and the counterexample is reproducible [49]. This corresponds to the notion of testability in requirements engineering [50] and to our view of scalability analysis as a form of experimental analysis [4], [51].

In addition to being precise and testable, scalability goals also should be complete and accurate in the sense that they correspond to what is truly needed for the system. In order to help achieve this objective, we have developed a systematic method to assist requirements engineers in elaborating scalability goals from an initial goal model that has been elaborated without concern for scalability issues. The method consists of systematically identifying and resolving scalability obstacles to the initial goals assigned to each system agent. The result of this is a consolidated requirements model in which the initial goal model has been extended with precise scaling assumptions, scalability goals, and additional requirements to help avoid, reduce, or mitigate scalability obstacles. Our method generates a set of alternative design options for resolving scalability obstacles that need to be evaluated in terms of costs and benefits so that the most appropriate options can be selected for implementation.

Our method thus contributes to addressing several of the key issues identified at the start of the paper:

- By extending the KAOS modeling framework with the concepts of scalability goals and scaling assumption, we facilitate the precise specification of testable scalability requirements.
- Our scalability obstacle analysis method provides eight new obstacle resolution tactics with 14 variants for managing scalability-related risks early in the development process, for identifying scalability obstacles induced by domain quantities whose expected operational ranges could prevent the system from satisfying its goals, and for exploring a wide range of alternative designs to prevent, reduce, or mitigate those scalability obstacles. Scalability obstacles have not been considered explicitly in previous techniques, such as Lamsweerde and Letier [20]. Furthermore, an important and distinguishing feature of our method is that we generate each scalability obstacle from *all* goals involving the same agent, in contrast to previous patterns and heuristics that generate obstacles one goal at a time.
- The elaboration of scaling assumptions required by our method helps requirements analysts begin to uncover uncertainties and disagreements about expected operational ranges for the domain quantities, as well as the impact that such quantities may have on the system's goals; exposing uncertainties and disagreements that would otherwise remain implicit is a first step toward resolving these issues.
- The result of our elaboration method is a goal-oriented requirements model in which software scalability requirements are related to scalability obstacles, domain scaling assumptions, and high-level business goals; such a model helps stakeholders understand the rationale and costs of specific requirements.
- The models resulting from our elaboration method also can be used as input for other techniques to evaluate alternative system designs and to make tradeoffs among competing goals, including those related to cost and time-to-market [3], [4], [23], [43], [44], [52].

We have validated our method by applying it to the major redesign of a state-of-the-art financial fraud detection system currently in use in many financial institutions throughout the world. Other smaller examples of the application of scalability obstacle analysis, notably on a previous model of the London Ambulance Service, have been presented by Duboc [33].

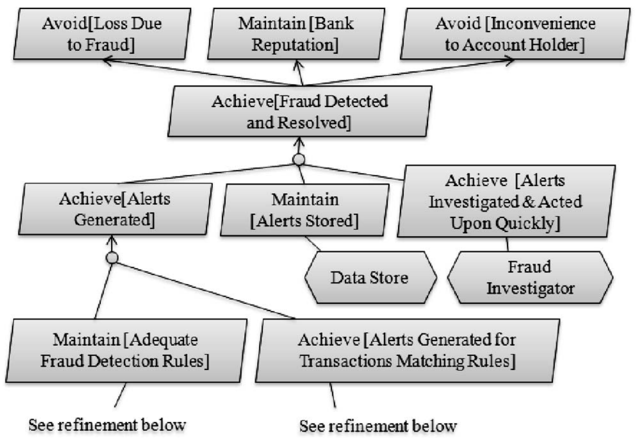
Avoid[Loss Due to Fraud]
Maintain [Bank Reputation]
Avoid [Inconvenience to Account Holder]
Achieve[Fraud Detected and Resolved]
Achieve[Alerts Generated]
Maintain [Alerts Stored]
Achieve [Alerts Investigated & Acted Upon Quickly]
Data Store
Fraud Investigator
Maintain [Adequate Fraud Detection Rules]
Achieve [Alerts Generated for Transactions Matching Rules]
See refinement below
See refinement below

Fig. 5. Refinement of the IEF top-level goals.

For future work, we intend to develop additional support for our technique to address some of the issues that have not been addressed fully in this paper. This includes the quantitative assessment of scalability goals and obstacles. For this purpose, we envision extending techniques for simulating and optimizing quantitative goal models [53] to handle time-varying domain assumptions, obstacle likelihoods, and levels of goal satisfaction needed to reason about scalability. In addition, we plan to undertake further investigation of scalability modeling and analysis in the context of cloud computing. We are collaborating currently with researchers at the University of Birmingham to model the scalability goals of an infrastructure for automatically composing service-based systems in a cloud environment [54] and to use goal modeling to systematically guide an organization in evaluating the choice and risks of adopting a cloud-based solution [55] and evaluating the tradeoff between scalability and consistency for replication techniques in clouds [56]. We also are beginning to explore new techniques for testing that a system satisfies its scalability requirements.

Appendix

Partial IEF Goal Model

See Figs. 5, 6 and 7.

Acknowledgments

This work was partially supported by the European IST FET program in project SENSORIA (IST-2005-016004), a UCL studentship to Leticia Duboc, the EPSRC grant EP/H011447/1, and a Wolfson Research Merit Award from the Royal Society to David Rosenblum. The authors thank Capacitas and NICE Actimize (formerly Searchspace) for discussions that led to many of the scalability tactics presented in this paper, and the anonymous referees for their constructive comments.

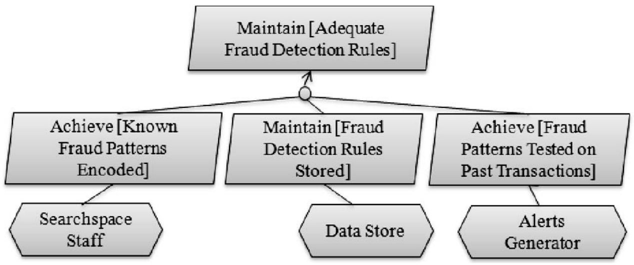
Maintain [Adequate Fraud Detection Rules]
Achieve [Known Fraud Patterns Encoded]
Maintain [Fraud Detection Rules Stored]
Achieve [Fraud Patterns Tested on Past Transactions]
Searchspace Staff
Data Store
Alerts Generator

Fig. 6. Refinement of the goal Maintain [Adequate Fraud Detection Rules].

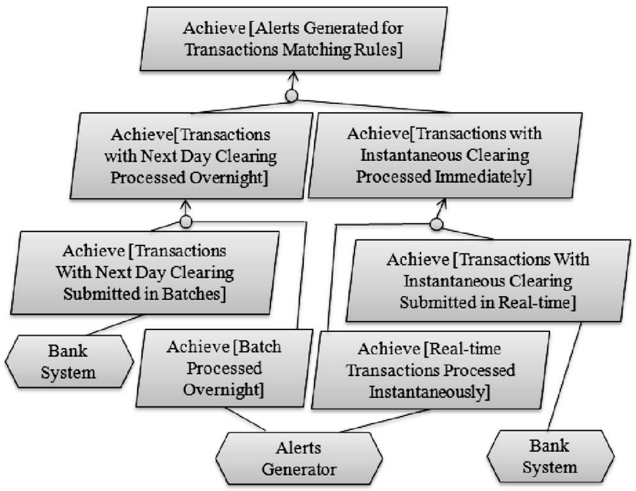
Achieve [Alerts Generated for Transactions Matching Rules]
Achieve[Transactions with Next Day Clearing Processed Overnight]
Achieve[Transactions with Instantaneous Clearing Processed Immediately]
Achieve [Transactions With Next Day Clearing Submitted in Batches]
Achieve [Transactions With Instantaneous Clearing Submitted in Real-time]
Bank System
Achieve [Batch Processed Overnight]
Achieve [Real-time Transactions Processed Instantaneously]
Alerts Generator
Bank System

Fig. 7. Refinement of the goal Achieve [Alerts Generated For Transactions Matching Fraud Detection Rules].

References

[1] R.P. Gabriel, L. Northrop, D.C. Schmidt, and K. Sullivan, "Ultra-Large-Scale Systems," *Proc. 21st ACM SIGPLAN Symp. Object-Oriented Programming Systems, Languages and Applications*, pp. 632-634, 2006.

[2] D.S. Rosenblum and A.L. Wolf, "A Design Framework for Internet-Scale Even Observation and Notification," *Proc. Sixth European Software Eng. Conf. Held Jointly with the Fifth ACM SIGSOFT Symp. Foundations of Software Eng.*, pp. 334-360, 1997.

[3] R. Bahsoon and W. Emmerich, "An Economics-Driven Approach for Valuing Scalability in Distributed Architectures," *Proc. IEEE/IFIP Seventh Working Conf. Software Architecture*, pp. 9-18, 2008.

[4] L. Duboc, D. Rosenblum, and T. Wicks, "A Framework for Characterization and Analysis of Software System Scalability," *Proc. Sixth Joint Meeting of the European Software Eng. Conf. and the ACM SIGSOFT Symp. Foundations of Software Eng.*, pp. 375-384, 2007.

[5] L. Duboc, E. Letier, D. Rosenblum, and T. Wicks, "A Case Study in Eliciting Scalability Requirements," *Proc. 16th IEEE Int'l Symp. Requirements Eng.*, 2008.

[6] Actuate Corporation, "Meeting Scalability Requirements for Enterprise Reporting Solutions," http://www.onixnet.com/actuate/scalability.pdf, July 2009.

[7] Microsoft TechNet Library, "Quantifying Availability and Scalability Requirements," July 2009.

[8] A. van Lamsweerde, *Systematic Requirements Engineering: From System Goals to UML Models to Software Specifications.* John Wiley & Sons, 2008.

[9] E.S.K. Yu, "Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering," *Proc. Third IEEE Int'l Symp. Requirements Eng.*, p. 226, 1997.

[10] M. Jackson, *Problem Frames: Analyzing and Structuring Software Development Problems.* Addison-Wesley, 2001.

[11] N.G. Leveson, *Safeware: System Safety and Computer.* Addison-Wesley, 1995.

[12] R.R. Lutz, "Software Engineering for Safety: A Roadmap," *Proc. The Future of Software Eng.*, pp. 213-226, 2000.

[13] R.J. Anderson, *Security Engineering: A Guide to Building Dependable Disributed Systems.* Wiley, 2001.

[14] A. van Lamsweerde, "Elaborating Security Requirements by Construction of Intentional Anti-Models," *Proc. 26nd Int'l Conf. Software Eng.*, pp. 148-157, 2004.

[15] C.B. Haley, R.C. Laney, J.D. Moffett, and B. Nuseibeh, "Security Requirements Engineering: A Framework for Representation and Analysis," *IEEE Trans. Software Eng.*, vol. 34, no. 1, pp. 133-153, Jan./Feb. 2008.

[16] ISO 9126, "Software Product Evaluation: Quality Characteristics and Guidelines for Their Use," 1991.

[17] B.W. Boehm, J.R. Brown, and M. Lipow, "Quantitative Evaluation of Software Quality," *Proc. Second Int'l Conf. Software Eng.*, pp. 592-605, 1976.

[18] L. Chung, B.A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering.* Kluwer Academic Publishers, 2000.

[19] P. Kruchten, *The Rational Unified Process: An Introduction.* Addison-Wesley, 2003.

[20] A. van Lamsweerde and E. Letier, "Handling Obstacles in Goal-Oriented Requirements Engineering," *IEEE Trans. Software Eng.*, vol. 26, no. 10, pp. 978-1005, Oct. 2000.

[21] E. Letier and A. van Lamsweerde, "Agent-Based Tactics for Goal-Oriented Requirements Elaboration," *Proc. 24th Int'l Conf. Software Eng.*, pp. 83-93, 2002.

[22] R. Koymans, *Specifying Message Passing and Time-Critical Systems with Temporal Logic.* Springer-Verlag, 1992.

[23] E. Letier and A. van Lamsweerde, "Reasoning about Partial Goal Satisfaction for Requirements and Design Engineering," *Proc. 12th Int'l Symp. Foundations of Software Eng.*, pp. 53-62, 2004.

[24] C. Potts, "Using Schematic Scenarios to Understand User Needs," *Proc. First Conf. Designing Interactive Systems*, pp. 247-256, 1995.

[25] Y. Akao, *Quality Function Deployment QFD: Integrating Customer Requirements into Product Design.* Productivity Press, 1990.

[26] W.N. Robinson, "Negotiation Behavior during Requirements Specification," *Proc. 12th Int'l Conf. Software Eng.*, pp. 268-276, 1990.

[27] J. Yen and W.A. Tiao, "A Systematic Tradeoff Analysis for Conflicting Imprecise Requirements," *Proc. IEEE Third Int'l Symp. Requirements Eng.*, p. 87, 1997.

[28] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, "Reasoning with Goal Models," *Proc. 21st Int'l Conf. Conceptual Modeling*, pp. 167-181, 2002.

[29] M. Feather, S. Cornford, J. Dunphy, and K. Hicks, "A Quantitative Risk Model for Early Lifecycle Decision Making," *Proc. Conf. Integrated Design and Process Technology*, 2002.

[30] P. Zave and M. Jackson, "Four Dark Corners of Requirements Engineering," *ACM Trans. Software Eng. and Methodology*, vol. 6, no. 1, pp. 1-30, 1997.

[31] C.B. Weinstock and J.B. Goodenough, "On Systems Scalability," Software Eng. Inst., Technical Note CMU/SEI-2006-TN-012, Mar. 2006, http://www.sei.cmu.edu/publications/documents/06.reports/06tn012.html, July 2008.

[32] E. Letier, "Reasoning about Agents in Goal-Oriented Requirements Engineering," PhD thesis, Université Catholique de Louvain, Dépt. Ingénierie Informatique, Louvain-la-Neuve, Belgium, 2001.

[33] L. Duboc, "A Framework for Characterization and Analysis of Software Systems Scalability," PhD thesis, Univ. College London, Dept. of Computer Science,, 2010.

[34] R. Darimont and A. van Lamsweerde, "Formal Refinement Patterns for Goal-Driven Requirements Elaboration," *Proc. Fourth ACM SIGSOFT Symp. Foundations of Software Eng.*, pp. 179-190, 1996.

[35] N. Maiden and S. Robertson, "Integrating Creativity into Requirements Processes: Experiences with an Air Traffic Management System," *Proc. 13th IEEE Int'l Conf. Requirements Eng.*, pp. 105-114, Aug. 2005.

[36] V. Chaudhary, M. Cha, J. Walters, S. Guercio, and S. Gallo, "A Comparison of Virtualization Technologies for HPC," *Proc. 22nd Int'l Conf. Advanced Information Networking and Applications*, pp. 861-868, Mar. 2008.

[37] J. Wiegert, G. Regnier, and J. Jackson, "Challenges for Scalable Networking in a Virtualized Server," *Proc. 16th Int'l Conf. Computer Comm. and Networks*, pp. 179-184, Aug. 2007.

[38] Z. Yanzhou, L. Chaoling, L. Lixin, and L. Wenjun, "Improving the Scalability of PrivacyCAs," *Proc. Second Int'l Workshop CS and Eng.*, vol. 2, pp. 111-116, Oct. 2009.

[39] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Int'l J. High Performance and Computer Applications*, vol. 15, no. 3, pp. 200-222, 2001.

[40] D. Linthicum, "The Case for the Hybrid Cloud," *Infoworld Cloud Computing*, Mar. 2010.

[41] R.M. Fano, "The MAC System: The Computer Utility Approach," *IEEE Spectrum*, vol. 2, no. 1, pp. 56-64, Jan. 1965.

[42] S. Masticola, A.B. Bondi, and M. Hettish, "Model-Based Scalability Estimation in Inception-Phase Software Architecture," *Proc. Eighth Int'l Conf. Model Driven Eng. Languages and Systems*, pp. 355-366, 2005.

[43] G. Brataas and P. Hughes, "Exploring Architectural Scalability," *Proc. Fourth Int'l Workshop Software and Performance*, pp. 125-129, 2004.

[44] E. Weyuker and A. Avritzer, "A Metric to Predict Software Scalability," *Proc. IEEE Eighth Symp. Software Metrics*, pp. 152-158, 2002.

[45] L.G. Williams and C.U. Smith, "QSEM: Quantitative Scalability Evaluation Method," *Proc. Int'l Computer Measurement Group*, pp. 341-352, 2005.

[46] C.U. Smith and L.G. Williams, *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software.* Addison-Wesley, Sept. 2001.

[47] M. van Steen, S. van der Zijden, and H.J. Sips, "Software Engineering for Scalable Distributed Applications," *Proc. 22nd Int'l Computer Software and Applications Conf.*, pp. 285-293, 1998.

[48] P. Jogalekar and M. Woodside, "Evaluating the Scalability of Distributed Systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 11, no. 6, pp. 589-603, June 2000.

[49] K.R. Popper, *The Logic of Scientific Discovery.* Routledge, 1959.

[50] S. Robertson and J. Robertson, *Mastering the Requirements Process*, second ed. Addison-Wesley, 2006.

[51] J.R. Ruthruff, S. Elbaum, and G. Rothermel, "Experimental Program Analysis: A New Program Analysis Paradigm," *Proc. Int'l Symp. Software Testing and Analysis*, pp. 49-60, 2006.

[52] R. Kazman, J. Asundi, and M. Klein, "Quantifying the Costs and Benefits of Architectural Decisions," *Proc. 23rd Int'l Conf. Software Eng.*, pp. 297-306, 2001.

[53] W. Heaven and E. Letier, "Simulating and Optimizing Design Decisions in Quantitative Goal Models," *Proc. 19th IEEE Requirements Eng. Conf.*, Sept. 2011.

[54] V. Nallur and R. Bahsoon, "Design of a Market-Based Mechanism for Quality Attribute Tradeoff of Services in the Cloud," *Proc. ACM Symp. Applied Computing*, pp. 367-371, 2010.

[55] S. Zardari and R. Bahsoon, "Cloud Adoption: A Goal-Oriented Requirements Engineering Approach," *Proc. Second Int'l Workshop Software Eng. for Cloud Computing*, pp. 29-35, 2011.

[56] T. Chen and R. Bahsoon, "Scalable Service Oriented Replication in the Cloud," *Proc. IEEE Int'l Conf. Cloud Computing*, pp. 766-767, July 2011.

Leticia Duboc received the PhD degree from the Department of Computer Science at University College London in March 2010. Prior to the PhD degree, she worked as a researcher at Fortent (formerly Searchspace) and as a research fellow at University College London. She is a lecturer in the Department of Computer Science at the State University of Rio de Janeiro (UERJ). She also holds an Honorary Research Fellowship with the Software Engineering Research Group at the University of Birmingham. Her research interests include scalability of software systems, requirements engineering, and early analysis of software qualities.

Emmanuel Letier received the PhD degree in software engineering from the University of Louvain in Belgium. He is a lecturer in the Department of Computer Science at University College London. His research interests include requirements engineering and system design. He is particularly interested in the development of practical formal methods to assist engineers in elaborating, analyzing, and evolving large-scale models of complex systems.

David S. Rosenblum received the PhD degree from Stanford University in 1988. He is a professor in the Department of Computer Science of the School of Computing at the National University of Singapore (NUS). Before joining NUS, he was a research scientist at AT&T Bell Laboratories in Murray Hill, New Jersey, from 1988 to 1996; an associate professor at the University of California, Irvine, from 1996 to 2002; the chief technology officer and principal architect of PreCache, Inc., from 2001 to 2003; and a professor of software systems at University College London from 2004 to 2011. His research interests include scalability of large-scale software systems, probabilistic modeling and analysis, and the design and validation of mobile, context-aware ubiquitous computing systems. He is the recipient of the 2002 ICSE Most Influential Paper Award for his ICSE 1992 paper on assertion checking, and the first ACM SIGSOFT Impact Paper Award in 2008 for his ESEC/FSE 1997 paper on Internet-scale event observation and notification (coauthored with Alexander L. Wolf). He is a fellow of the ACM and the IEEE.