# Management of Performance Requirements for Information Systems

Brian A. Nixon

**Abstract**—Management of performance requirements is a major challenge for information systems as well as other software systems. This is because performance requirements (e.g., "a student records system should have good response time for registering students") can have a global impact on the target system. In addition, there are interactions and trade-offs among performance requirements, other nonfunctional requirements (NFRs, or software quality attributes, e.g., accuracy), and the numerous alternatives for the target system. To provide a systematic approach to managing performance requirements, this paper presents a "Performance Requirements Framework" (PeRF). It integrates and catalogues a variety of kinds of knowledge of information systems and performance. These include performance concepts, Software Performance Engineering (SPE) principles for "building performance into" systems, and information systems development knowledge (including requirements, design, implementation, and performance). In addition, layered structures organize performance knowledge and the development process. All this knowledge is represented using an existing goal-oriented approach, the "NFR Framework," which offers a developer-directed graphical treatment for stating NFRs, analyzing and interrelating them, and determining the impact of decisions upon NFRs. This approach allows customized solutions to be built, taking into account the characteristics of the particular domain. The use of PeRF in managing performance requirements is illustrated in a study of performance requirements and other NFRs for a university's student records system. This paper concludes with a summary of other studies of information systems, tool support, and directions for future work.

**Index Terms**—Performance requirements, information systems, requirements engineering, nonfunctional requirements, Software Performance Engineering (SPE), semantic data models (SDMs), catalogues of performance and development knowledge, conflicts and trade-offs, student records system, Year 2000 compliance, NFR Framework, Performance Requirements Framework (PeRF).

## 1 Introduction

An important challenge during the development of information systems is to manage with *performance requirements.* For a student records system, for example, a developer may consider performance requirements to "achieve good response time for registering students" and "achieve good space usage for storing student information." In practice, performance requirements often focus on response time and throughput. These requirements are often specific to a particular application system. While performance requirements are often stated very briefly, their ramifications can be quite complex.

*Information systems* with good performance are needed for the operations of many organizations. Examples of information systems range from credit card systems to student records systems to government administrative systems. Unlike other types of systems (e.g., operating system software), information systems have to maintain a persistent database, handle incoming data and transactions, and provide an interface with external users.

To be effective, such systems should meet both functional requirements and nonfunctional requirements. *Functional requirements* describe what a system should do, typically stating the intended functionality in terms of input-output relationships. For example, a student records system could have functional requirements that fee invoices be printed. By comparison, *nonfunctional requirements (NFRs* or *software quality attributes*) deal with such software quality factors as performance, accuracy, security, reliability, and robustness.

This paper presents a framework for managing performance requirements for information systems. The "Performance Requirements Framework" (PeRF) integrates and catalogues a variety of types of knowledge of performance and information systems. These include performance concepts, Software Performance Engineering (SPE) principles for "building performance into" systems, and information systems development knowledge (including requirements, design, implementation, and performance). In addition, layered structures organize performance knowledge and the development process. By treating performance requirements as NFRs, all this knowledge is represented using an existing approach for managing NFRs, the "NFR Framework" [13], [39], [19]. The NFR Framework offers a goal-oriented treatment for stating NFRs, analyzing and interrelating them, and determining the impact of decisions upon NFRs. It records all the developer's actions in a graphical representation, the "Softgoal Interdependency Graph." This developer-directed approach allows customized solutions to be built, taking into account the characteristics of the particular domain.

This paper also illustrates the use of PeRF in managing performance requirements by way of a study of performance requirements for a university's student records

• *The author is with the Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 3H5.*
*E-mail: nixon@cs.toronto.edu.*

*Manuscript received 29 Apr. 1999; revised 6 Mar. 2000; accepted 15 Mar. 2000.*
*Recommended for acceptance by A. Cheng, P. Clements, and M. Woodside.*
*For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 111939.*

system. Based to a large extent on information from the actual domain, the study deals with trade-offs and priorities using SPE principles, uses a layered structure to organize treatment of a variety of issues, and treats performance and other NFRs (such as accuracy and survivability) together, using the NFR Framework's graphical representation.

This paper is organized as follows: The remainder of Section 1 presents several issues for managing performance requirements, along with our general approach, and an overview of our specific approach. It also introduces the domain studied in this paper. Section 2 presents the Framework in detail. Then, Section 3 uses the Framework to study a student records system. Section 4 summarizes other empirical studies which apply PeRF to information systems of organizations from a variety of domains with varying characteristics, workloads, and priorities. This paper concludes with a discussion of related work, tool support for the Framework, performance prediction, and directions for future work.

### 1.1 Issues and General Approach for Managing Performance Requirements

Managing performance requirements can be quite difficult due to the nature of performance requirements. First, performance requirements can have a global impact on the target system. Meeting one such requirement may change several parts of a system. Indeed, one cannot simply add a "performance module" to produce a system with good performance. Rather, one must consider performance requirements throughout the system and throughout the development process.

In addition, our experience [44] in implementing information system designs indicates that there can be quite *complex interactions and trade-offs* among performance requirements, other NFRs, target implementation techniques, and features of the source conceptual design specification language. For example, making a transaction as fast as possible may decrease flexibility for future changes in design.

Furthermore, a developer must choose from the many implementation techniques available, each with varying performance characteristics. Thus, there are many factors to consider and many decisions to be made. Finally, to meet performance requirements, which vary from system to system, one needs to consider the characteristics of the particular organization and system—such as its workload.

Given the nature of performance requirements for information systems, we feel that the following general approach will aid their effective management. To help deal with the large number of decisions, we feel there is a need to offer a structured, systematic approach. We want to structure consideration of issues so that the developer can reduce the number of concepts addressed at any one time. In addition, it is desirable to use principles for "building performance into" systems, rather than an ad hoc approach.

To deal with the large body of knowledge, including performance concepts and information system development techniques, we see the need for *catalogues of knowledge* to collect and organize the knowledge and make it available to the developer. As the body of knowledge can grow, these catalogues should be extensible.

NFRs interact with each other and can have an impact on several parts of a system. Analysis of NFRs into more specialized requirements having more localized impact, can be done by a goal-oriented process, an application of goal-oriented requirements engineering (see, e.g., [22], [21]).

In the development process, the importance of developers' expertise cannot be underestimated; hence, we look to an *interactive* approach, giving control of the development process to the *developer.* A developer-directed process can be supported by tools, e.g., to aid retrieval of catalogued knowledge.

In keeping with the above general approach, this paper presents a "Performance Requirements Framework" (PeRF). PeRF specializes the core NFR Framework, which uses a *"satisficing"* approach [56]. In this way, NFRs for software systems are represented as "softgoals," which are expected to achieve NFRs with acceptable limits (sufficiently satisfactorily), rather than be absolutely satisfied.

### 1.2 Overview of the Domain Studied

We now introduce the Student Records System (SRS) of the University of Toronto. Section 3 studies the use of the Framework for the SRS. The SRS is also used in Section 2 to illustrate aspects of the Framework. Note that we did not participate in developing the SRS; instead, we tried to model and analyze performance requirements.

The system is large, handling the records of 54,000 students, taking tens of thousands of courses. SRS reflects the complexity of the organization of the university, with 18 faculties, scores of departments, and three campuses. The SRS database is implemented in DB2, using 400 relational tables, including internal tables.

Until recently, most students were able to register for courses and request other registration services using an automated telephone service with interactive voice response. On 15 March 1999, a new SRS became operational, supporting the entire student records process from the time of application to graduation. It provides much more functionality than the old system. All students now use the new system, accessing it via World Wide Web or telephone.

The main priority in the domain was to make the new system Year 2000 compliant. As the old system was not compliant, the new system had to be in operation by 15 March 1999 in order to handle the 1999–2000 academic year. Given compliance, other actual priorities were to use newer technology and to increase functionality within the time-frame available.

The new system was developed by starting with a compliant system from another university, then making modifications. Service standards and requirements are not really made in advance. However, SRS staff do respond to student complaints re telephone service and do track when the system goes down.

The largest faculty gives graduating students priority for registering for courses. In the new system, all graduating students can use the system during the first week of registration, while others are locked out. In the second week, all graduating and third-year students can use the system and other groups of students are given access in subsequent weeks.

Domain information was primarily obtained in discussions and correspondence with Dr. Eva V. Swenson, Director of Student Information Systems of the University of Toronto, who is also a faculty member of the Department of Computer Science, University of Toronto. Additional information was obtained from a requirements document for the new system [60], as well as student registration handbooks [61], [62]; these documents have not been "published" in the academic sense. Much, but not all, of the requirements document has been implemented in the new system.

We try to faithfully reflect the domain information available to us and indicate where we have supplemented the domain information with some assumptions (creative imagination) to fill in gaps in our knowledge or to illustrate various aspects of the Framework. Parts of some illustrations use actual domain information (e.g., faculty priorities), while other parts are assumed (e.g., particular NFRs). For example, Section 4 uses the SRS and its actual registration statistics, but makes several assumptions about the SRS. When making assumptions, we have tried to be reasonable. For example, we make assumptions about seeking low response time and low space usage.

## 2 The Performance Requirements Framework

This section presents the Performance Requirements Framework (PeRF). We start with a description of the inputs and outputs of the process of using PeRF. We then present the NFR Framework's generic components and structure for representing and using knowledge of NFRs and system development, which PeRF adopts. We then present PeRF's knowledge of performance concepts, performance requirements, and information system development.

A premise of this work is that *performance and development knowledge must be organized.* PeRF's catalogues represent and organize a variety of types of knowledge which must be considered when addressing performance requirements for information systems. This includes implementation techniques for a class of information system design languages, semantic data models (SDMs, see [2], [25], [53] for overviews). Layered performance structures (based on [26]) are used to further organize catalogues, as well as the development process. Features of SDMs are used to structure the layering. Developers can use and augment catalogues.

### 2.1 Input and Output Factors

When using PeRF to manage performance requirements for an information system, there are a number of input and output factors of the process.

Inputs include a number of *catalogues* of knowledge, which are somewhat independent of the particular domain, and applicable to a variety of systems. These include catalogues of: performance concepts (e.g., [34]); SPE principles for building performance into systems [57], [58]; basic techniques for information system development, e.g., database implementation; and generic techniques for handling NFRs.

For a particular domain, *basic input factors* include: *performance requirements* for the system, e.g., sales authorization should be fast; *other NFRs* for the system (e.g., requiring accuracy and security in maintaining student information); *functional requirements* for the system; *priorities* for the organization and system [58], e.g., graduating students should be given priority when registering for courses; and the expected *workload.* Workload can be expressed at various degrees of detail, ranging from individual operations to applications to overall organizations. In this paper, we focus on the workload of an organization expressed in domain terms, e.g., the number of students and the number of courses taken per term.

The results of the development process include the following *outputs: a record of development decisions made* and the *reasons* for decisions; *an indication of which performance requirements are met* and to what extent; *a record of the the interactions and trade-offs* among NFRs, priorities, workload, decisions made, and alternatives not chosen. In addition, *a prediction of performance* of the selected system may optionally be produced, e.g., using the approach of [45].

Additional input factors may be considered, the details of which depend on the particular development approach taken and the phase at which they are considered. The additional input factors include: *development techniques* to produce a target system from a source specification and their associated performance characteristics and trade-offs; *interactions and trade-offs* among NFRs, priorities, development techniques, etc.; *the source specification of the system,* here including definitions of students, registration operations, and constraints ("rules" in [60]); and *features of the source specification language.*

PeRF especially considers the implementation phase of development, which takes a (source system) conceptual design specification expressed in a particular SDM, TaxisDL, and produces a (target system) implementation expressed in a relational database programming language, DBPL (TaxisDL and DBPL will be described later in this section).

The analysis using PeRF's basic input factors applies regardless of the particular source and target languages used. The actual system for student registration used a relational database, but did not use TaxisDL and DBPL. However, we feel that the analysis made using PeRF's additional input factors can be helpful at some level of abstraction of details. In addition, PeRF could be extended to handle other source and target languages.

### 2.2 The NFR Framework

PeRF incorporates the NFR Framework, which offers a goal-oriented treatment for a developer to interactively state NFRs as softgoals, then analyze, interrelate, and prioritize them, and determine the impact of decisions upon NFRs. All this information is represented in *softgoal interdependency graphs (SIGs),* which were influenced by work in decision support systems [36].

#### 2.2.1 Treatment of NFRs

Meeting NFRs may involve dealing with trade-offs as well as requirements which are subject to change or are not clearly defined. Hence, due to their nature, NFRs may differ

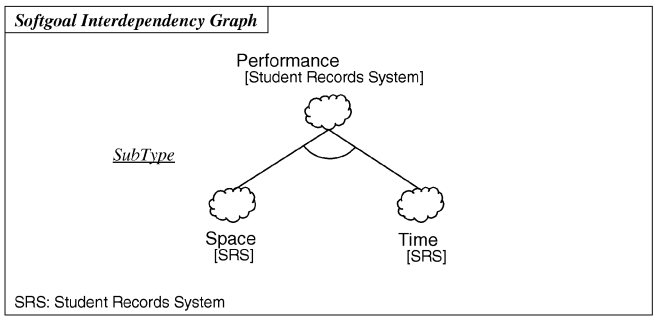


Fig. 1. An initial performance softgoal and a refinement.

from "goals" (as in traditional problem-solving in artificial intelligence [40]), which are fully satisfiable. Hence, NFRs are called "softgoals" in the NFR Framework to indicate a degree of softness or flexibility in treating NFRs.

Treating performance requirements (as well as requirements for accuracy, security, etc.) as softgoals seems appropriate, especially when there are trade-offs (e.g., faster response can be obtained if more space is used). In addition, even "hard" performance requirements can be addressed by having the developer ensure that the requirements actually are met.

Earlier work [5] on quality characteristics of software noted that, when developers are made aware of software quality concerns, that by itself helps improve the overall software quality. The NFR Framework provides a *process-oriented approach* to software quality, reflecting the view that the quality of a software *product* is largely determined by the quality of the *process* used to construct it. Development decisions are justified by arguing that they help a system meet certain NFRs or explaining why they do not. By comparison, *product-oriented* approaches to NFRs (overviewed in [32]) develop formal definitions of NFRs so that a software system can be evaluated (often quantitatively) by the degree to which it meets its requirements. While product-oriented approaches are often *quantitative* [32], [5], [4], the NFR Framework's process-oriented treatment is definitely *qualitative,* adopting ideas from qualitative reasoning [1]. Actually, product-oriented quantitative metrics and process-oriented qualitative measures are best seen as complementary, both contributing to the treatment of NFRs.

We now consider some elements of the NFR Framework. Development starts by collecting functional requirements, NFRs, workload and other domain information, and available catalogues of development knowledge.

### 2.2.2 Softgoals

NFRs are represented as *NFR softgoals,* which are one of three kinds of softgoals. A softgoal has an *NFR type* (NFR concept, e.g., Performance or Accuracy), along with one or more *topics* (or subjects or parameters) (e.g., Student Records System (SRS)). For example, Performance [StudentRecordsSystem] is an NFR softgoal (and a performance softgoal) meaning that there should be good performance for the SRS. This softgoal is shown at the top of Fig. 1 in a softgoal interdependency graph (SIG). For all figures, the reader is referred to the legend given

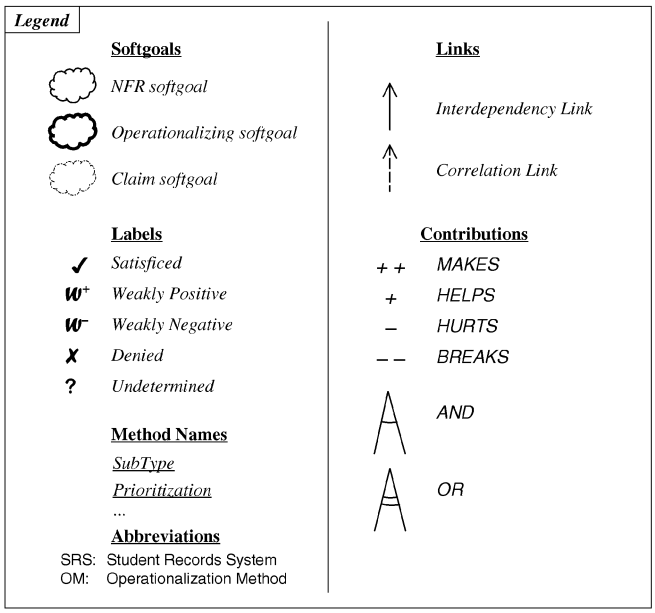


Fig. 2. Legend for figures.

in Fig. 2. The SIGs in Section 2 are mainly to illustrate the Framework and contain some assumptions not necessarily in the domain.

The NFR Framework provides the *NFR Type Catalogue,* with entries for the most general NFRs, including performance, accuracy, and security. It also provides a subtyping mechanism.

PeRF provides the type details for performance by providing a *Performance Type Catalogue* as an extension to the entry for the Performance type in the NFR Type Catalogue. We interpret softgoals for the subtypes of Performance, as follows: Time[...] is read as good time performance. For Time and ResponseTime, this means seeking decreased or low time. However, Throughput[...] means increased (or high) throughput. Good space performance means decreased or low usage.

### 2.2.3 Interrelating Softgoals

The developer interactively and iteratively refines softgoals into smaller and more specific softgoals. Developers decide what to refine and how much to refine. They use their expertise to determine how to decompose problems and what to prioritize.

Softgoals are interdependent and are interrelated by *interdependencies.* Interdependencies can be created in a number of ways. One is by *refining* softgoals into other softgoals. One form of refinement is *decompositions*, which refine NFR softgoals into other NFR softgoals.

Fig. 1 shows a refinement of the "parent" softgoal Performance[SRS] downward into "offspring" subgoals Space[SRS] and Time[SRS]. This creates an interdependency between the parent and the offspring. While refinements work *downward,* "contributions" work *upward,* showing the result for the parent if the offspring are satisficed. Here, the parent and offspring are related by an *AND* "contribution." The interdependency (including the refinement and contribution) can be summarized as: Space[SRS] *AND* Time[SRS] SATISFICE Performance[SRS]. This means

that *if* there is good space performance for the SRS *and* there is good time performance for the SRS, *then* there is good performance for the SRS. All offspring must be satisficed (sufficiently satisfied, but not necessarily absolutely satisfied) for the parent to be satisficed. If any offspring are denied (sufficiently, but not necessarily absolutely), then so will the parent.

Topics (e.g., SRS) of a softgoal can also be refined. For example, Time[SRS] could be refined into time softgoals, each dealing with a different component of the SRS.

### 2.2.4 Contributions

Meeting an offspring softgoal may help or hinder the meeting of a parent softgoal. Hence, softgoal relationships are represented as *contributions* to indicate the nature of the interdependency's contribution to meeting parent softgoals. The various contributions are shown in the legend of Fig. 2 and are defined elsewhere [13], [39].

An *AND* contribution is satisficed if all of its offspring are satisficed, while an *OR* contribution is satisficed if any one offspring is satisficed.

With a *MAKES* contribution ("++" in figures), satisficing the offspring is enough to "make" (satisfice) the parent, is *labeled* as satisficed ("$\surd$"). If, however, the offspring is denied (not satisficed), then the parent will be labeled as denied ("$\times$"). Another contribution is BREAKS ("–"), which is roughly the opposite of *MAKES*. Here, satisficing the offspring is enough to "break" (deny) the parent. If the offspring is denied, then there will be *weak positive* evidence for the satisficing of the parent (shown as a "$W^+$" label). If developers feel that the result should be a satisficed label, they can change the parent's label accordingly.

A *HELPS* ("+") contribution indicates that satisficing the offspring gives a weak positive contribution to meeting the parent, but alone will not satisfy the parent. If, however, the offspring is denied, then there will be weak negative evidence for the satisficing of the parent ("$W^-$"). A related contribution is *HURTS* ("-" in figures). Satisfying the offspring "hurts," but does not deny, the satisficing of the parent, i.e., the parent receives a *weak negative* label. If the offspring is denied, then there will be *weak positive* evidence for the satisficing of the parent.

### 2.2.5 Argumentation and Prioritization

Developers can state reasons for their decisions using *claim softgoals* (arguments), shown as dashed clouds. Domain information, such as workload (e.g., the number of students) can be used as arguments for choosing among alternatives. In Fig. 3, a continuation of part of Fig. 1, student complaints about poor service are used as a reason for giving priority to response time. While the student complaints actually are in the domain, we have made assumptions about the prioritization and particular NFRs. Here, the developer determines that the claim is satisficed, hence the softgoal is labeled as satisficed ("$\surd$"). If, however, the claim were denied, it would be labeled ("$\times$").

The figure shows an example of domain information being used as design rationale to justify *prioritization.* The developer *prioritizes* response time for the SRS, making it a *critical* softgoal. This is done by refining parent softgoal ResponseTime[SRS] into a prioritized

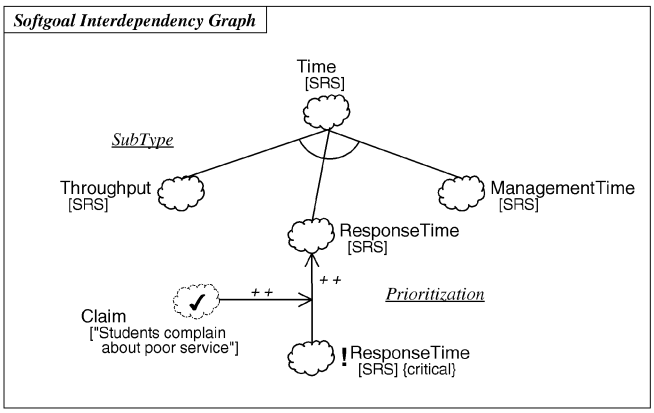


Fig. 3. Identifying a priority and stating reasons.

offspring, !ResponseTime[SRS]{critical}. All priority softgoals have an exclamation mark as a prefix. The kind of priority (e.g., critical or dominant) is not a topic of the softgoal, but is an attribute of the softgoal.

Previously, we have seen softgoals contributing to other softgoals. Here, we see the claim softgoal contributing to the *interdependency link* between !ResponseTime[SRS]{critical} and ResponseTime[SRS]. This *MAKES* contribution means: If the claim is satisficed, then so is the interdependency link. *Interdependency links* are usually satisficed, in which case the rules for contributions between softgoals hold. The reader may wish to look ahead at Fig. 21 for an example of a denied interdependency link, where the "$\times$" along the link indicates that it is *denied.* In this case, the parent is no longer considered.

### 2.2.6 Consideration of Techniques for the Target System

Implementation techniques for a target system are considered and represented as *operationalizing softgoals* (shown as dark clouds in figures) which represent techniques used to realize (or "operationalize") softgoals by "satisficing" them.

A development decision, i.e., a selection among development alternatives, does not necessarily satisfy NFRs, but may make positive or negative partial contributions to NFRs. Accordingly, a softgoal is associated with a "label," which indicates the degree to which the softgoal is satisficed or denied. Selected operationalizations are labeled as satisficed ("$\surd$"), while rejected alternatives are labeled as denied ("$\times$").

### 2.2.7 Evaluating the Impact of Decisions

The impact of decisions upon NFRs is determined using the NFR Framework's *evaluation procedure.* This works bottom up, starting with leaf softgoals. It considers the labels of offspring softgoals and the contributions of interdependency links to determine the labels of the parent softgoals. Importantly, the procedure can dynamically ask developers for their expertise, e.g., when there are inconsistencies. Eventually, it determines if overall (top) softgoals are satisficed.

As an example of a rule for evaluation, if an offspring is satisficed ("$\surd$") and *HELPS* ("+") its parent, then the parent

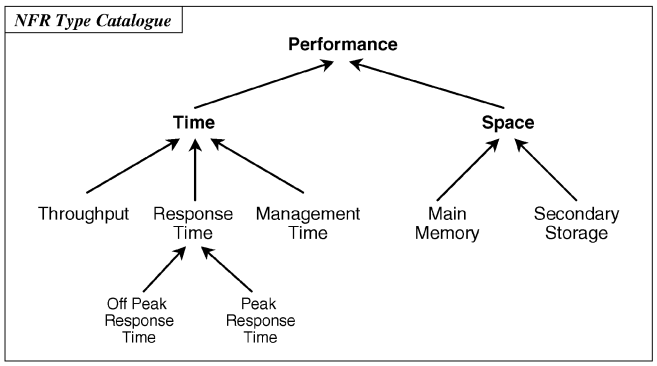


Fig. 4. The performance type.

will be weakly satisficed ("W+"). If the developer feels that the situation allows it, the W+ label could be changed to satisficed. The discussion above about contributions also gives some of the rules for evaluation.

#### 2.2.8 Methods

*Methods* (techniques) for refining softgoals, based on industrial experience and academic results, can be catalogued for ready use by a developer. The NFR Framework provides three kinds of refinement methods, as well as a number of generic decomposition methods and argumentation templates.

*Decomposition methods* refine an NFR softgoal into other NFR softgoals. Both the types and topics of softgoals can be refined.

*Operationalization methods* refine a softgoal (either an NFR softgoal or an operationalizing softgoal) into operationalizing softgoals, which stand for alternatives in the target system. Note that operationalizations may result in choices for data structures, operations, etc., in the target system and are not restricted to operations.

*Argumentation methods and templates* state an argument using a claim softgoal. Using *prioritization methods,* softgoals can be associated with a priority.

One generic decomposition method is the SubType method, which takes a softgoal dealing with an NFR type and refines it into several softgoals, each dealing with a subtype. For example, to apply this method to deal with performance subtypes, we use the performance type (Fig. 4). The refinement in Fig. 1 can be viewed as an application (or instantiation) of a parameterized decomposition method: Space[*Info*] *AND* Time[*Info*] SATISFICE Performance[*Info*]. Here, *Info* is a parameter, standing for an "information item," which can be a class, an attribute of a class, or an operation on a class. By simple substitution, SRS can replace *Info* to provide an application of the method. The SubType method is placed in a catalogue and is available for a developer to consider when refining a SIG. In SIGs, method names are underlined.

Developers use their experience and expertise to define methods and determine which softgoal types and topics are parameterized. For example, while developing a SIG, certain patterns may recur in the graph, possibly varying in type or topic; this certainly happened when we were conducting case studies. In this case, the developer may consider defining additional methods based on these patterns. As another example, general rules or principles (e.g., SPE principles) may be recorded as methods, likely with parameters. In the development of the NFR Framework and PeRF, parameterization on type has been found useful to interrelate softgoals with the same type or to take one softgoal with a given type and relate it to softgoals involving subtypes. Similarly, parameterization on topic can interrelate softgoals with the same topic; in addition, structural decomposition methods (see Fig. 7) take one softgoal with a given topic and relate it to softgoals involving components, subsets, attributes, subclasses, instances, or functions of the topic.

In addition to methods, trade-offs among softgoals can be catalogued as *correlations.* For example, time-space trade-offs can be catalogued. Then, when a softgoal interdependency graph contains a time softgoal and a space softgoal, a correlation could be detected between them and an appropriate interdependency added to the graph.

Section 4 will present more on the methodology for using the framework.

Now, we consider how PeRF uses the above components of the NFR Framework to catalogue knowledge of performance and information systems. PeRF provides the Performance Type, uses SPE principles, provides a layered structuring for considering performance issues, and adds and specializes refinement methods and correlations to deal with performance and information systems.

### 2.3 Performance Concepts and the Performance Type

To provide a terminology for specifying performance requirements, PeRF catalogues basic performance concepts (cf. [34]) in the *Performance Type.* The *Performance Type catalogue,* shown in Fig. 4, is available for specification of performance softgoals and for refining softgoals dealing with general performance concepts into softgoals addressing more specific concepts.

The Performance type has subtypes Time and Space, representing respective time and space performance requirements on a particular system. In turn, time can be decomposed into throughput, response time, and "management time" (discussed below), while space can be decomposed into main memory and secondary storage.

Smith [58] and Jain [27] call for performance goals to be specific and measurable; for example, does a goal apply all day or at peak time? Jain provides the acronym "SMART" as a reminder that performance requirements should be specific, measurable, acceptable, reliable, and thorough. Concerning specificity of softgoals, we can distinguish softgoals for peak time vs. those for off-peak times and softgoals for a good mean (e.g., low response time) vs. goals for good uniformity of values (e.g., low standard deviation in response time). These *characteristics* of performance types are represented as qualifiers (e.g., as prefixes) to the standard types (e.g., peak-time response time) and can be combined (e.g., uniform peak-time response time). See the bottom of Fig. 4 for some examples. One could further refine types, e.g., to consider such measures as 90th and 95th percentiles, e.g., of response times for classes of transactions.

Work on metrics has produced classifications of NFRs. Jain [27] defines a number of *metrics:* criteria, used to compare the performance of systems, that relate to the speed, accuracy, and availability of services. These include response time, throughput, utilization, reliability, and availability.

The Performance type catalogue can be simply represented in a graph showing types, subtype relationships, and characteristics. The form of the catalogue can be represented in a modeling language, such as Telos [38], to express these relationships.

### 2.4 SPE Principles

Software Performance Engineering (SPE) principles [57], [58] are another kind of performance knowledge that is used in PeRF. We summarize some of the principles used to "build performance into" software systems and help provide good *responsiveness* to users (the response time as seem by an end-user of the system). They include:

- *Centring Principle:* Efforts to improve performance should focus first on the *critical* and *dominant* parts of the workload. We call these *priorities.* From the dominant workload (i.e., the few routines which are executed the most), trim what is unnecessary to do while the user waits. Assuming that the frequent operations can be sped up, overall average response time can be improved.
- *Processing vs. Frequency Tradeoff Principle:* Minimize the product of the processing time per execution of an operation and the frequency of executing the operation. It is important to identify when a slight increase in time causes a great decrease in frequency (or vice versa), such as "family plan" situations where handling several requests does not cost much more than handling one request.
- *Fixing Principle:* To build in performance, "fixing" (the process of mapping functions and information to instructions and data) is done as early as possible.

PeRF draws on the spirit of SPE principles in order to focus on priorities and deal with trade-offs.

### 2.5 Layered Performance Structures

One way to organize the consideration performance issues is to use a *layered structure.* This arranges the various isses into layers. Instead of considering all issues at once, the developer can focus on one layer (and its smaller set of issues) at a time.

Each layer can have a decision-making process with inputs and outputs. Of the inputs, some come from the developer and others come from outputs of other layers. The outputs may be presented to the developer or to other layers. Thus, there is an organized flow of information among the various layers, where the developer can manage *interactions* among issues at different layers.

In some cases, the layers can be placed in a linear top-down order, with outputs of a layer flowing only *downward* into lower layers. This kind of decision-making is shown in Fig. 5. When this kind of constrained information flow is possible, management of interactions should be simpler. This is because decision-making models at higher layers can typically be reexpressed in terms of the (simpler) models at lower layers. PeRF uses such a top-down layered performance structure, which was inspired by Hyslop's layered model for performance prediction of relational databases [26]. The *structure* of Fig. 5 is taken from Hyslop's model. In Hyslop's case, the source is a relational database and the target is at the physical level; performance prediction is done by a model at each level, with inputs flowing into a performance model from higher layers and output performance measures flowing down to prediction models at lower layers.

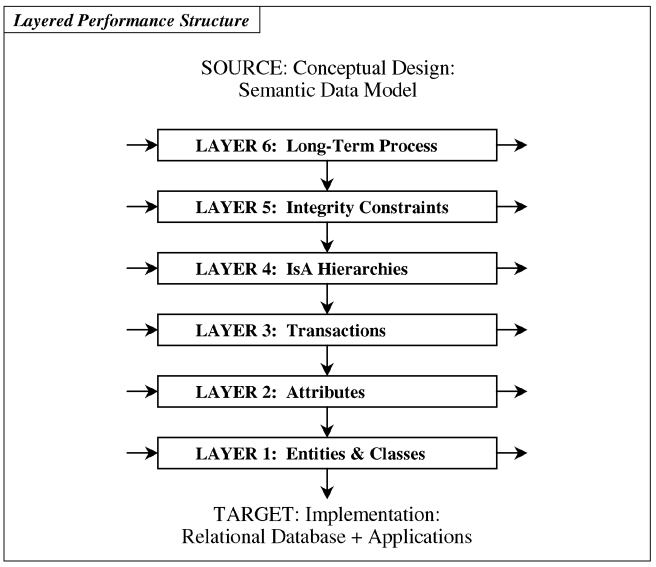


Fig. 5. Layered structures for selecting alternatives and predicting performance.

In our case, the source specification is expressed in an SDM and the target is a relational database. SDMs are based on an Entity-Relationship data model [10] and are used to specify and manage a large and complex information base. Classes (with attributes) are arranged in inheritance (IsA) hierarchies. Our particular experience [41], [11], [42], [12], [43], [44] considered the compilation of designs expressed in an SDM, Taxis [37], [7], into a relational database programming language augmented with application programmes, DBPL [55]. Taxis offers short-term transactions, longer-term processes, and integrity constraints, and applies IsA hierarchies to all these features. A successor to Taxis is the Taxis Design Language, TaxisDL [7].

Let us now explain the basis for determining the issues handled at each layer and the ordering of the layers. Moving from the source specification to the target specification can be done by going through a series of intermediate specifications. Each such specification can have its own specification language and it may be possible to arrange the specification languages as a *series of language subsets.* In this way, the source specification language can be the full language. The language at the next layer down can be a subset of the source language, with some language feature(s) omitted. Moving down to this layer then involves "dealing with" this feature and expressing the result in the simpler subset language. This is repeated at lower layers, where successive languages form smaller and smaller

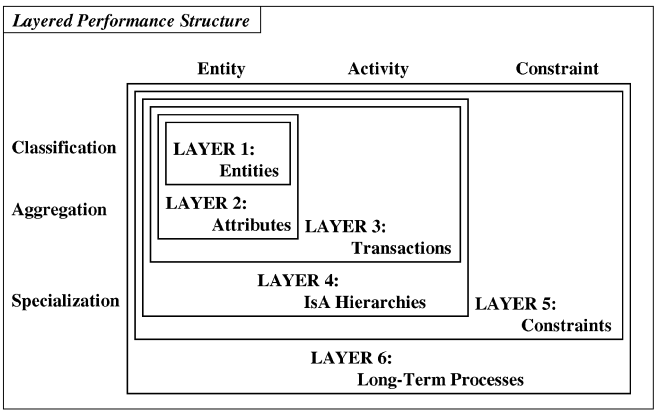


Fig. 6. Taxis language features arranged in layers.

subsets, until the target specification language is reached. By reaching the bottom layer, we have dealt with the features of the source language and done so in a structure manner, resulting in a specification expressed in the target language. Furthermore, if some of the intermediate languages correspond to known languages, the results obtained using the models at those corresponding layers can be compared with known characteristics of those languages.

We use this *language-feature—based* layering to TaxisDL. Now, how are the particular features chosen and ordered? We consider a grid with two axes, shown in Fig. 6. On the horizontal axis, we have the conceptual linguistic features relating to the ontology, i.e., what can be expressed about the world. They are Greenspan et al.'s [24] three kinds of *information items*: entities, activities, and constraints. On the vertical axis are the organizational (epistemological) features. They are classification, aggregation, and specialization [59] offered by SDMs. We produce an ordered layering by selecting a small portion of the grid and forming higher layers by enlarging the selected area, preferably along one axis at a time.

The lowest level language is the target *relational data model.* Layer 1 adds *entities,* both persistent data entities (such as John, an instance of the class Student) and finite entities (e.g., integers), arranged in *classes*. Layer 2 adds *attributes* of classes. Layers 1 and 2 roughly correspond to the Entity-Relationship Model [10]. Layer 3 then adds *transactions,* modeled in TaxisDL as classes with attributes and instance entities. Layer 4 adds *IsA hierarchies,* which are used to arrange classes of both entities and transactions. Layers 1 through 4 roughly correspond to a Taxis subset [42]. Layer 5 adds *integrity constraints* and Layer 6 adds *long-term processes,* whose nature has aspects of entities, activities, and constraints [12]. Layers 1 through 6 comprise the source conceptual design specification language, roughly comparable to Taxis [12] and TaxisDL [7].

We have used only this particular layering in our framework and studies. However, we feel that developers can adapt this layered structure to handle a variety of information systems and information system modeling languages. One reason is that the layering handles the Entity-Relationship model, as well as several subsets of SDMs. Furthermore, the lower layers (entities, attributes, and transactions) have fairly direct counterparts with programming and database concepts (e.g., variables or records, attributes or fields, and transactions or functions). In addition, a developer is able to use different layering structures as desired.

This layering is used in PeRF in a few ways. It is used to arrange softgoals and SIGs in layers. It is also used for organizing catalogues and for organizing performance prediction.

### 2.5.1 Layered Softgoals and SIGs

Layering provides additional structuring for *softgoal interdependency graphs.* Fig. 11 shows a SIG divided into several layers. Matters relating to each layer are grouped in a smaller graph, which is linked to graphs at other layers (such links are not shown in Fig. 11 to simplify its appearance). Consequently, fewer matters are considered in each layer's graph.

As a result of having layered SIGs, we can associate *softgoals* with a particular *layer of a graph.* For example, the softgoal Time[Registration Subprocesses, 3] is a softgoal for good time performance for registration subprocesses. In Fig. 11, it is stated at Layer 3 of a graph, which deals with transactions. Here, "3" is an additional topic, an optional *layer topic.*

### 2.5.2 Other Uses of Layering

Layering also provides additional structure for *organizing catalogues* of knowledge. For example, development techniques relating to a particular language feature are grouped together. In addition, at each of the layers of PeRF, the SPE principles can be applied to dealing with performance requirements for the features at the layer; this is discussed elsewhere [48].

Layering can also be used to organize *performance prediction.* Interestingly, PeRF uses the same layered organization for selection among target alternatives as it does for prediction of performance of a selected implementation, which is discussed in Section 5 and outlined in [45]. By sharing a common organization, results should be more easily shared among components and future extensions. In addition, a layered approach should be applicable to other data models, e.g., object oriented models [33].

### 2.5.3 Interdependencies and Layering

Interdependencies can be stated among softgoals at different layers.

First, a top softgoal, which might not have a layer specified, can be refined into a softgoal at a layer of a SIG (shown in Fig. 12, a continuation of Fig. 3): ResponseTime[Operations(SRS), 3] *HELPS* !ResponseTime[SRS]{critical}. This interdependency link brings us down to Layer 3 of the SIG, which deals with transactions. All the softgoals at Layer 3 are shown with a layer topic of 3. Note that interlayer links work in the same way as other interdependency links.

A layered softgoal can also be refined downward to a softgoal within the same layer. As will be illustrated later, a layered softgoal can also be refined downward to a softgoal at a lower layer, in which case they are connected by an *interlayer interdependency link.* In principle, it is possible to refine a softgoal upward to higher layers, but then we lose the property of only refining downward to simpler layers and the analysis of interactions would be more complex.

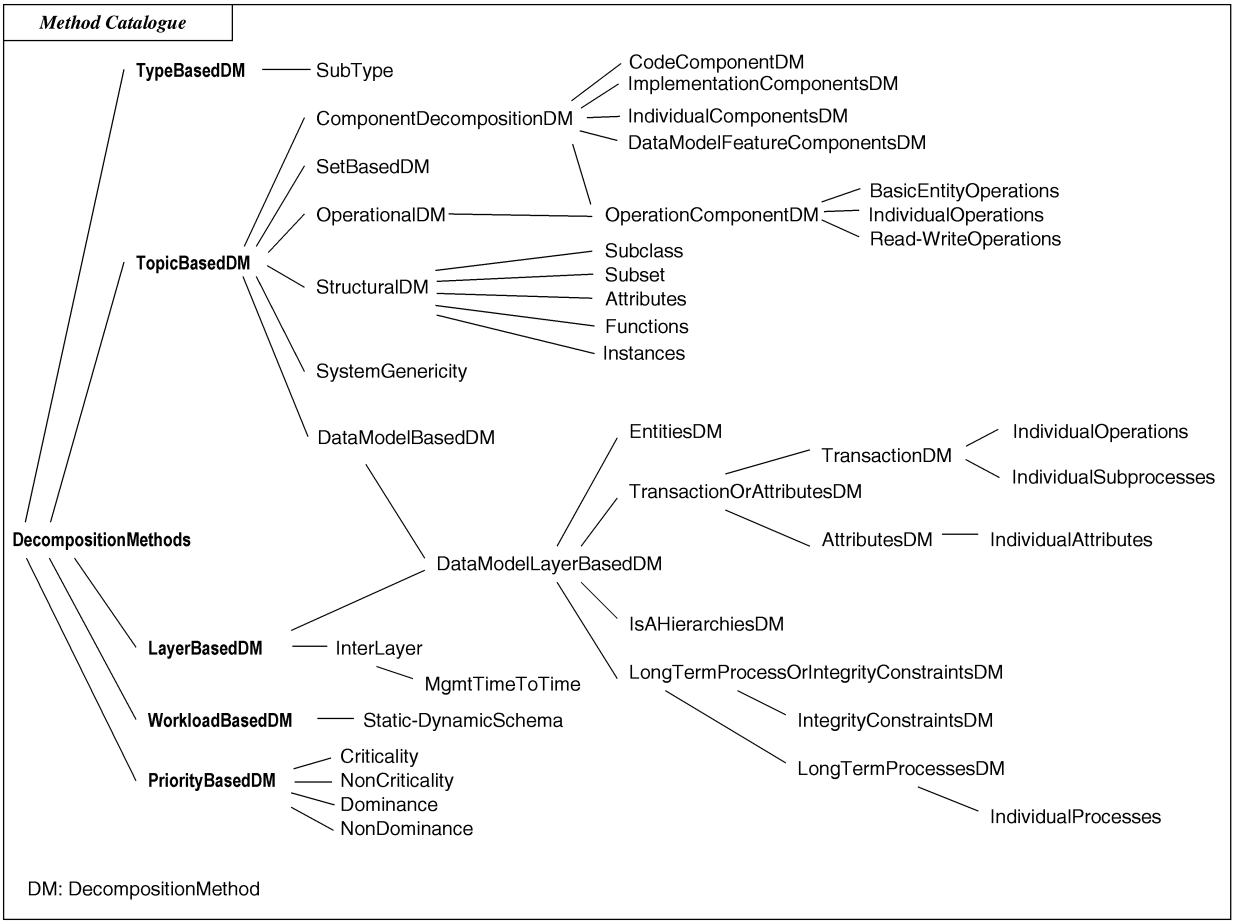


Fig. 7. Catalogue of decomposition methods.

### 2.5.4 InterLayer Evaluation

The result of refinements is SIGs, often containing multiple layers and connected by interlayer interdependency links. Each layer can have operationalizations, representing target alternatives for that layer. The operationalizations can be expressed in the (intermediate) target language for that layer. The developer can then select operationalizations at each layer, typically at the bottom of a layer.

*Evaluation* of a multilayer SIG typically starts with leaf softgoals of the bottom layer. Evaluation then proceeds upward through a layer and then up through higher layers. Some results are percolated upward via interlayer links to higher layers. Interlayer evaluation works using the same procedure used for intralayer evaluation. Eventually, it determines if overall (top) softgoals for each layer have been met.

## 2.6 Performance Refinement Methods and Correlations

PeRF provides catalogues of refinement methods and correlations. The catalogues offer and organize techniques and principles from SPE, SDMs, databases, etc.

### 2.6.1 Catalogues of Methods

PeRF offers catalogues of refinement methods, including decomposition, argumentation, and operationalization methods. Figs. 7, 8, 9, and 10 organize PeRF's decomposition and operationalization methods, including all such methods used in this paper and a representative number of the methods used in earlier empirical studies (Section 4).

Fig. 7, extended by Figs. 9 and 10, presents decomposition methods, while Fig. 8 provides an operationalization catalogue. One can use these catalogues as a kind of roadmap for the descriptions of particular methods which follow.

To facilitate their use by developers, methods are placed in broader groups and more specific subgroups. The "top" level of the catalogue, drawn at the left of the figures, showns the broadest grouping of methods. As we move "down" (actually to the right), more specific subgroups are shown. The first level down shows the major groupings of methods.

Some of these first-level groups deal with the structure of softgoals (type, topic, layer, and priority), some deal with sources of methods (SPE, SDMs, databases, object-oriented systems, etc.), and others deal with system characteristics (workload) and system development strategies. Then, smaller groupings are shown, e.g., dealing with particular layers or particular SPE principles. Groups of operationalization methods and decomposition methods are indicated in the figures by a suffix of "OM" or "DM," respectively, while specific methods generally do not have a suffix. Even where some individual methods are not shown, most groups of methods are given. More detail is shown for methods from SPE and SDMs.

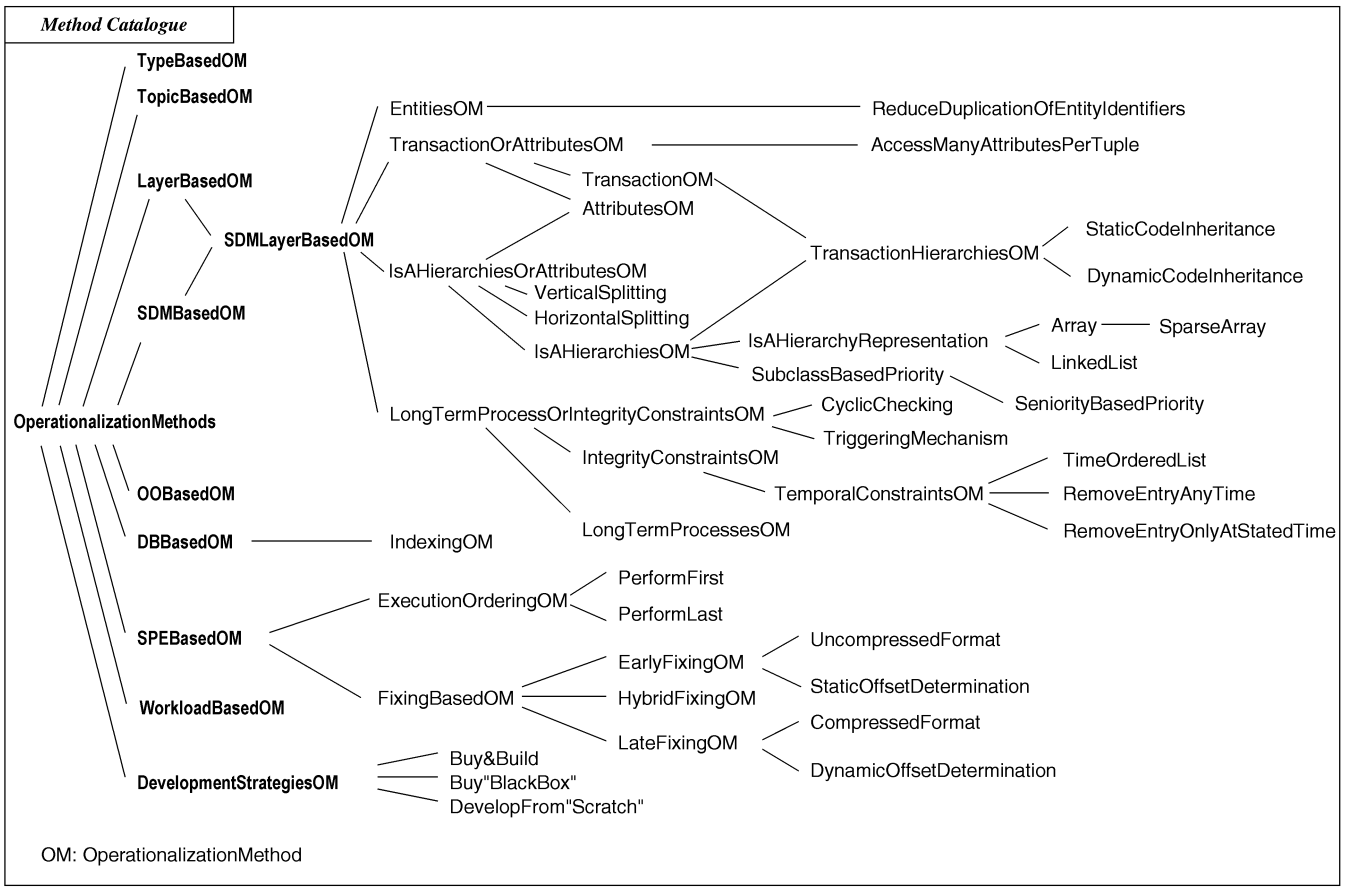


Fig. 8. Catalogue of operationalization methods.

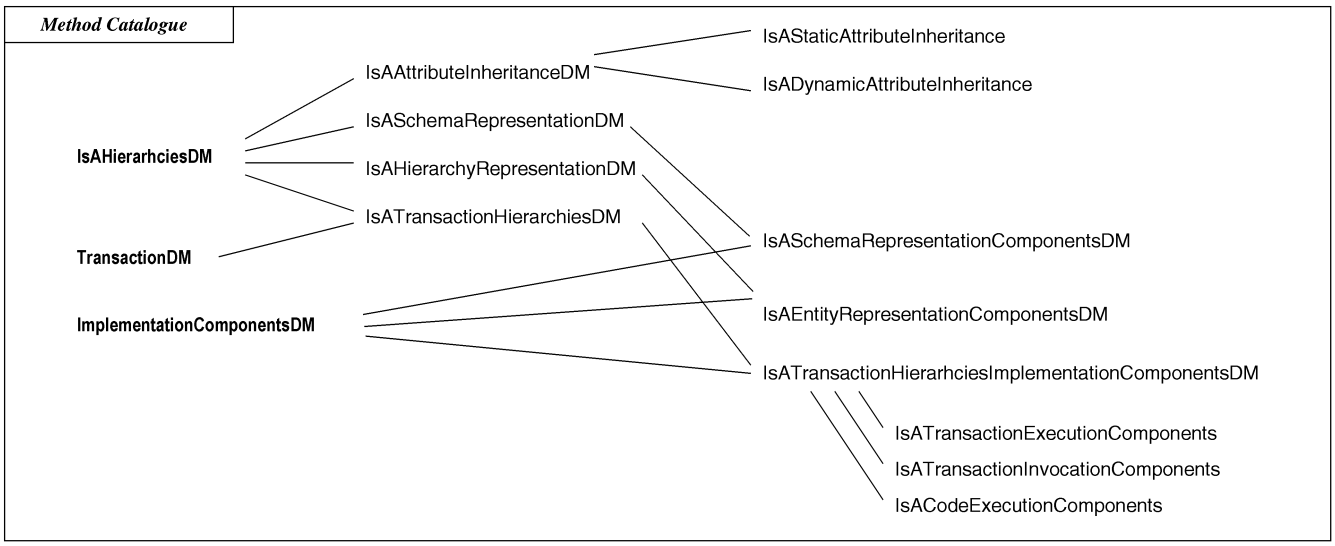


Fig. 9. Catalogue of decomposition methods for Layers 4 and 3.

The broader groups can be thought of as axes which organize the catalogue. Particular methods may draw on one or more of the broader groups of methods. For example, one method might involve a data model feature at Layer 2 and the use of early fixing from SPE, while another method might involve a topic refinement based on SDM implementation techniques.

The catalogues can help a developer to find applicable methods. For example, a developer considering issues from one or more particular axes could examine the relevant portions of the catalogue in order to produce a small set of methods which can then be considered in more detail. For example, a particular softgoal might involve a particular layer, a particular data model feature which has subcomponents, a particular priority, and a particular situation where an SPE principle could be applied. These issues can guide the search of the catalogue. In addition, by examining the catalogue at various levels of groups, the developer can

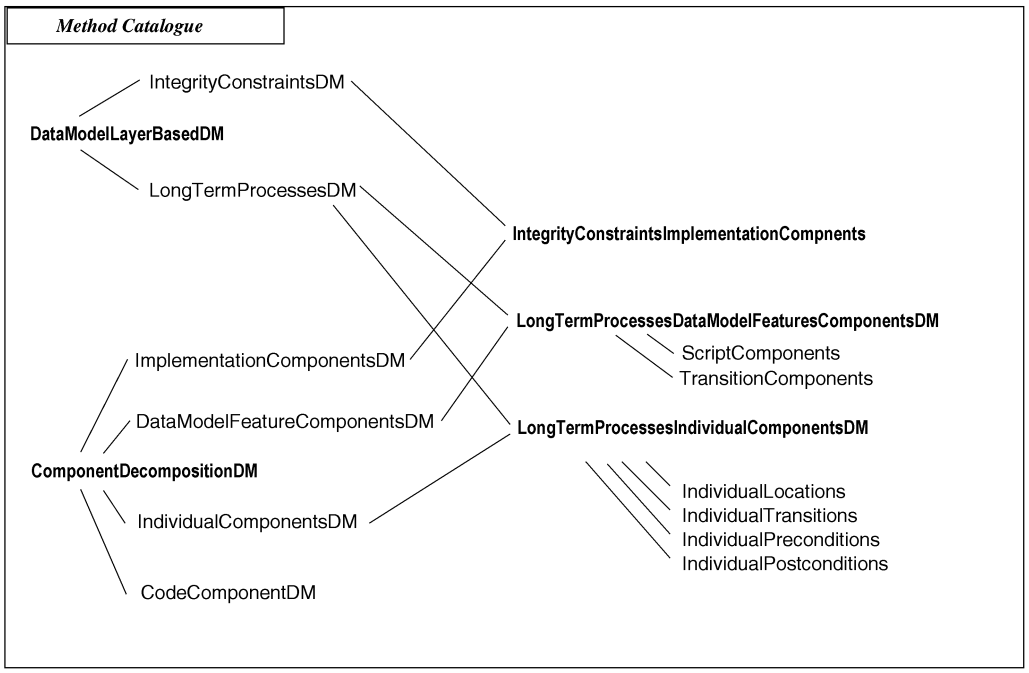


Fig. 10. Catalogue of decomposition methods for Layers 6 and 5 (adapted from [47]).

conduct a search without necessarily reviewing each individual method in the groups.

### 2.6.2 Method Application and Parameterization

While the catalogues are made available, it is the developers who use their experience and expertise to determine which methods to apply, where to apply them, and the extent to which softgoals will be refined. In some cases, knowledge of the domain as well as target techniques will be helpful. For example, suppose there are different techniques (operationalizations) for registering undergraduates and graduate students. Given an time softgoal involving registering students, a developer, anticipating that operationalizations may eventually be needed, may well consider refining the softgoal into one time softgoal for registering undergraduates and another for registering graduates. In other cases, knowledge of priorities may also be useful for determining when to apply methods. For example, if fourth year students are given priority for registration, a softgoal involving students might be refined into a collection of softgoals, each involving students of a different year. In other cases, knowledge of trade-offs will be helpful. For example, a developer who wishes to consider a time-space trade-off could refine a performance softgoal into a time softgoal and a space softgoal.

### 2.6.3 Performance Decomposition Methods

PeRF offers several decomposition methods which refine an NFR softgoal into others.

*Component decomposition methods* refine a softgoal dealing with an item into softgoals dealing with components of the item. For example, a softgoal for a block of code can be refined into softgoals for each subblock [57], [58] using the CodeComponents method. Likewise, a softgoal for an information item can be refined into its *implementation components.*

*Operation component decomposition methods* refine softgoals dealing with operations on information systems. They draw on work on databases and SDMs. For example, using the BasicEntityOperations method, a softgoal for an information item can be refined into softgoals for creating, retrieving, updating, and removing the item. Other operation component methods, including the IndividualOperations and Read-Write Operations methods, will be illustrated in Fig. 12.

*Data model feature decomposition methods* refine a softgoal dealing with a data model feature into softgoals for its components. For example, there are several such methods dealing with features of Taxis.

*Layer-based methods* deal with the layers of softgoals. For example, a softgoal dealing with a long-term process at Layer 6 might be decomposed into several softgoals dealing with short-term transactions at Layer 3.

*Prioritization methods* can be used to associate softgoals with a priority, such as criticality or dominance. Following the SPE principle for centring, attention can then be focused on meeting the priority softgoals.

We now turn to operationalization methods and correlations.

### 2.6.4 Operationalization Methods and Correlations from SPE

Several operationalization methods come from SPE.

*Fixing Methods.* By EarlyFixing, early connection is made between an action and the instructions that achieve it. A number of such methods *HELP* time performance, including indexing and static determination of offsets of attributes. Some also *HURT* space performance, including the use of an uncompressed format for storing information (which uses more space, but avoids the time to uncompress information before accessing it).

In contrast, LateFixing involves run-time overhead, making a late connection between an action and the

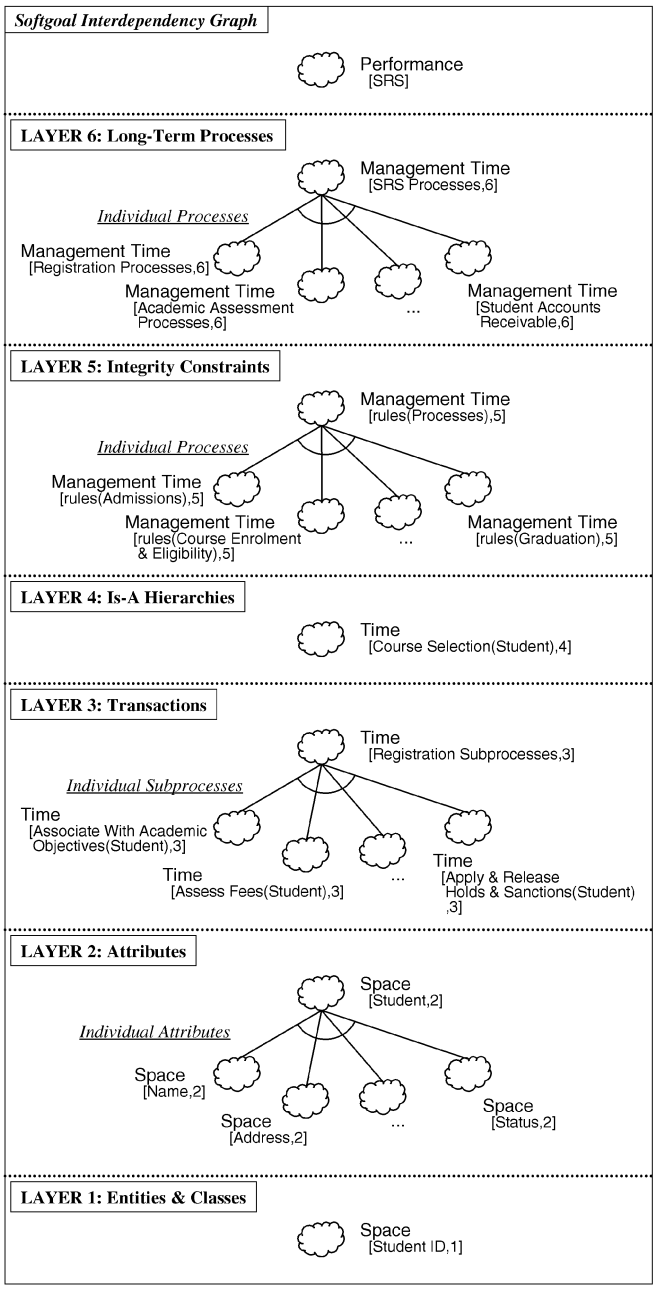


Fig. 11. Some main softgoals at several layers of a graph.

instructions. This may *HURT* average time performance, e.g., if the same connections are repeatedly made at execution time. On the other hand, late fixing may *HELP* time performance, e.g., when a module receive a parameter which is used at execution time to select a very time-efficient algorithm.

*Execution Ordering Methods.* Another group of SPE methods involves stating the relative order of execution of operations. For example, priority tasks can be performed first (PerformFirst), while PerformLast can be used for nonpriority items.

Positive and negative contributions can be very helpful for analyzing trade-offs. These relationships can be represented as *correlations.* As a result, interdependencies which have not been stated by a developer can be generated in a SIG. This is done by using pattern matching to compare the SIG with the correlations. Here are some examples of correlations (the topic and layer parameters have been omitted for readability): PerformFirst *HELPS* Time; PerformLast *HURTS* Time; DynamicOffsetDetermination *HURTS* Time; CompressedFormat *HELPS* Space; and CompressedFormat *HURTS* Time. Correlation catalogues can be drawn as a table, with the operationalizations along one axis and the NFRs along another. The individual table entries are the contribution (e.g., *HELPS*) of the operationalization to the NFR.

### 2.6.5 Operationalization Methods for Semantic Data Models

Techniques from the implementation of SDMs (e.g., [43], [12], [42]) can be represented as operationalization methods and can be associated with particular layers. We consider two issues for SDM implementation, representation of IsA hierarchies and implementation of constraints and long-term processes. Fig. 9 extends Fig. 7 by providing more details of the catalogue of decomposition methods for Layer 4 (and, to some extent, Layer 3), while Fig. 10 provides details for Layers 6 and 5. These figures show narrower groups of methods being formed from more than one larger group.

Several implementation issues arise from the arrangement of classes in *inheritance (IsA) hierarchies* (Layer 4). This leads to a number of target techniques for storage and manipulation of attributes. When storing large amounts of persistent data, say modeled using an SDM, one implementation alternative is to use relational database technology. If we can represent attribute values within a relational schema, we can exploit the efficient storage and access techniques already developed for relational databases. However, inheritance hierarchies result in collections of attribute values whose appearance is more like a "staircase" than a table (Fig. 13). Consider the class of persons, which has a specialization consisting of students, which is further specialized into graduate students. Persons have a name attribute which is inherited by students and graduate students. Students have an additional attribute for their average grade, which is inherited by graduate students who also have an advisor.

As a result, a simple relational representation may waste space. For example, one alternative for representing all attributes is to use a universal relation that has one tuple per entity, one column per attribute, and null values for attributes not applicable to an entity. Obviously, such a representation would be highly inefficient with respect to space usage, although it allows static determination of attribute offsets. A second, more interesting, alternative involves using one relation per class [59]. One may then store all attributes (newly defined or inherited) of a particular class in the corresponding relation (horizontal splitting) or only the newly defined attributes (vertical splitting). Horizontal and vertical splitting are examples of operationalizations which deal with data structures in the target system. Vertical splitting results in storing few attributes per tuple, which can help retrieval time. Meanwhile, horizontal splitting stores several attributes per tuple, which can reduce the total amount of storage per

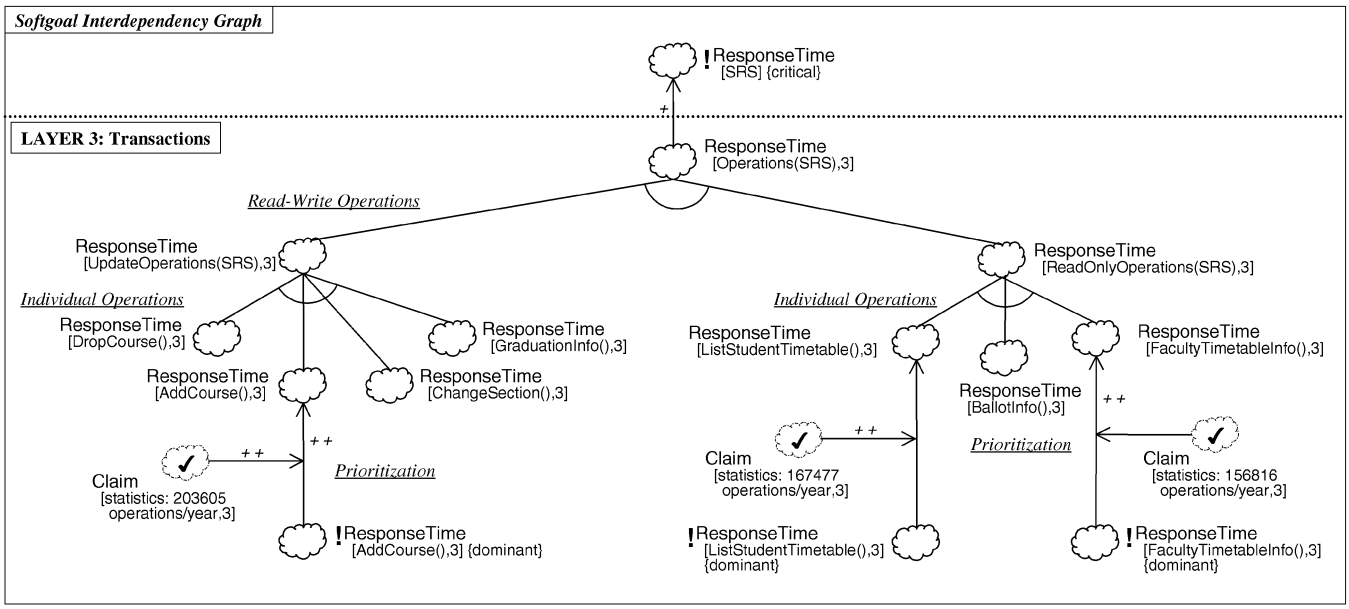


Fig. 12. A graph with statistics.

entity, assuming each tuple has an entity identifier. In addition, time performance depends on expected workload, e.g., whether the name is often retrieved for all students, or mainly for graduate students.

Other operationalization methods relate to *implementation of long-term processes and integrity constraints* (Layers 6 and 5 of PeRF). Long-term processes may last weeks, months, or years, having a number of activities to perform. These activities include determining which short-term transactions to invoke at particular times and determining when to check constraints so that constraints are maintained correctly and efficiently. For example, the system should enforce a constraint that all students should pay their fees; however, it will not be efficient to check every student's account every day of the year. Concerning performance requirements, one observation (made by Mendelzon) is that the time to *manage* these activities is different from response time. Hence, PeRF includes ManagementTime (abbreviated MgmtTime) as part of the Performance Type (Fig. 4). By distinguishing ManagementTime from other time types, a developer is able to be more specific when specifying performance requirements, as advocated by Jain [27] and Smith [58].

Concerning implementation of long-term processes and constraints, a number of alternatives for Taxis are presented elsewhere [12], along with performance characteristics. Also provided are conditions, including domain information, which determine which implementation technique to use. For example, in the student registration system, "cyclic checking" (repeatedly checking all constraints for all entities) might be realistic for a very small faculty (say, with hundreds of students) with very simple constraints.

## 2.7 Example: Refinement and Operationalization

We illustrate refinement and operationalization, using actual domain statistics for a recent full year of the telephone service of the old SRS. The refinements and NFRs are assumptions.

By considering transactions (operations), we illustrate how development could proceed by refining Layer 3 of the SIG in Fig. 12. The softgoal for response time for operations can be refined using the Read-Write Operations method, resulting in two softgoals. Then, the response-time softgoal for updating operations is refined into response-time softgoals for individual operations, such as adding a course or dropping a course. Likewise, the response-time softgoal for read-only operations is refined into softgoals for individual operations.

The developer reviews the statistics and notes that three operations are highly used. They are prioritized as forming the dominant part of the workload.

Offering good response time for adding a student to a course was assumed to be a priority. How can !ResponseTime[AddCourse(Student), 3]{dominant} be "operationalized?" Suppose there are five openings for a course and six students ask to join it at nearly the same time. In the SRS, requests are handled First Come First Served (see Fig. 14, an extension of part of Fig. 12) by millisecond; they are handled in real time, one at a time.

## 2.8 Approaches to Treating Performance

Why does PeRF use a substantially qualitative approach, as is provided by the NFR Framework, for handling performance requirements, when performance is generally quantitative? In addition, why address the quality of the development process when the performance of the software product is so important? Our response is that qualitative and process-oriented aspects are needed, along with quantitative and product-oriented ones, but different aspects will be stressed at different stages of development:

1. *At the early stages of development:* It seems helpful to start *qualitatively*. We start with *brief*, high-level requirements. We want to deal especially with broad overall *relationships* and conflicts among NFRs and between NFRs and implementation techniques.

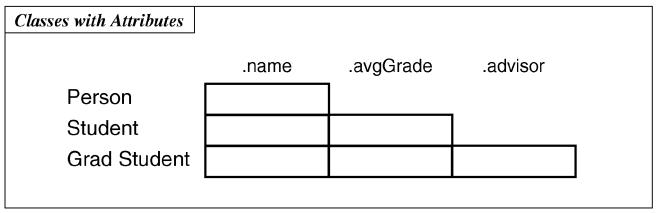


Fig. 13. Classes with inherited attributes.

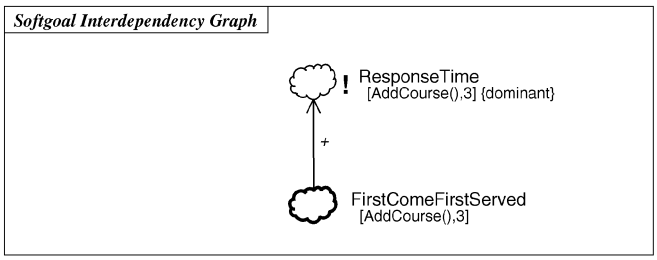


Fig. 14. An operationalization.

Initial efforts may focus on direction, e.g., moving toward or away from a requirement.

2. *At late stages of development:* A *quantitative* approach will certainly be needed. We can quantitatively evaluate the product, e.g., by predicting its performance, to see if it will meet performance requirements.
3. *During intermediate stages of development:* We could use a qualitative approach to select just a few implementation alternatives to evaluate quantitatively, thus decreasing the size of the decision space. The prediction results can also indicate trends, which can help the qualitative decision making process. Note that prediction may be helped by a layered approach: If an intermediate product is at a particular layer, we can evaluate and predict performance more fully at that layer using approach "2" above since we can treat the intermediate product as the "end" product of a smaller step in development.

PeRF focuses on qualitative, process-oriented selection among implementation alternatives, early in development. An early approach [45] to quantitative, product-oriented prediction of performance for the case of SDMs is outlined in Section 5 of this paper. Future work would combine qualitative and quantitative approaches.

## 3 Study: Student Records System

Now we present a study of performance requirements and other NFRs for the Student Records System (SRS) of the University of Toronto, using the Performance Requirements Framework (PeRF). The purpose of this study is to take a description of an actual domain and system with some actual performance requirements and other NFRs and then use PeRF to model NFRs and reason about them.

### 3.1 Conducting the Study

The study was conducted in an interactive and iterative manner. This involved discussions with the domain expert, who described the domain, the system, the priorities, and overall design, and answered questions. This author reviewed information and documents and used the framework to develop (and then revise) SIGs for the system.

We happened to apply the Framework just before (and after) the new system was released, i.e., when most of the system development work had been done. We did not develop the system. Neither did we independently make development decisions for the purpose of comparison with decisions made by the system developers. Rather we attempted to model their requirements, rationale, and decisions, by acting as developers might, using the Framework. To do so, we used the following information.

**Domain information** was presented in Section 1.

**Functional Requirements** are detailed in [60], which describes a number of processes and constraints (rules). The new SRS supports the entire student records process from the time of application to graduation. It provides much more functionality than the old system.

**Workload.** Some statistics for the annual workload of the old student telephone service, expressed in terms of the university domain, were shown in Fig. 12.

**Initial NFRs.** We consider some initial nonfunctional requirements, which will be addressed in this section, and illustrate some aspects of the Framework.

- *Time and accuracy requirements for Year 2000 compliance.* Compliance for the Year 2000 is a very important consideration for the actual system. In discussion with the domain expert, Year 2000 compliance was identified as definitely being needed for the continued operation (survivability) of the system, i.e., to avoid a crash. In addition, compliance was probably needed to maintain the accuracy of student registration data.
- *Priority given to registering graduating students.* This is based on an actual priority from the domain.

Our treatment of the first requirement illustrates trade-offs among actual target alternatives for achieving Year 2000 compliance. The trade-offs and trade-off considerations are taken from the domain. Some of the NFRs and prioritizations are assumptions, prepared in a retrospective way.

Our presentation of treating the second requirement is mainly illustrative of how the Framework can handle a combination of priorities and nonpriorities. It makes assumptions about data model features used and illustrates the use of layers to organize a Softgoal Interdependency Graph (SIG).

At this point, the reader may wish to refer back to Fig. 11 as a kind of overview. SRS has a number of processes (modules), including admissions, registrations, fees, and refunds, student accounts receivable, and registration evaluation. Fig. 11 shows a number of performance softgoals, at different layers, along with some initial refinements. Most of the topics of the softgoals are actually from the domain, including actual domain processes with associated rules (constraints) and subprocesses, as well as names of attributes of students. However, the particular NFRs shown and the layered organization, are based on our assumptions. The study of priority for graduating students

| SIG | Approx. number of softgoals | Layers addressed | Subject of SIG |
|---|---|---|---|
| Figures 16, 17, 18 | 16 | general | Year 2000 compliance, timely delivery, implementation alternatives & tradeoffs |
| Figure 11 | 26 | general, 6, 5, 4, 3, 2, 1 | illustrates layering, illustrates system modules, makes some assumptions |
| Figures 19, 20, 21 | 13 | 6, 4, 3 | priority for graduating students |
| X | 9 | 6, 5 | process & constraint mgmt. |
| Figure 12 | 18 | general, 3 | operations & statistics, target system |
| Y | 18 | 3, 2 | student records, and an assumed trade-off |
| Z | 10 | 3, 2, 1 | peak loads, space usage, target system |

Fig. 15. Complexity of the SIGs used in this study.

draws on Layers 6 and 4 of Fig. 11. We have also assumed that some of the "subprocesses" of [60] are short-term transactions and that operations for special classes of students (e.g., graduate vs. undergraduate) could be modeled by IsA hierarchies.

### 3.2 Effort Involved in Applying the Framework

It is interesting to consider the effort that was involved in applying the Framework in this study and the complexity of the SIGs produced.

The elapsed time to conduct the study, from the initial contact with the domain expert until the completion of the first draft of this paper, was under two-and-a-half months. The author was also working on other projects during that period and tended to work on the study in "bursts" of time.

During one week of the first month, the author and the domain expert had at least three contact times to establish initial contact, discuss the domain and the system and to obtain system statistics.

During the second month, further discussions were held and documentation reviewed and substantially all the SIGs were drawn. Some of the initial SIGs were prepared in the first half of that month, followed by revision and preparation of additional SIGs.

In the first half of that month, the discussions with the domain expert clarified aspects of the domain and system, including workload statistics. In addition, the Year 2000 issue, along with the data model and implementation alternatives and trade-offs, were discussed. As a result, the SIG of Fig. 12 and the SIG of Figs. 16, 17, and 18, were prepared by this author, along with some work on Fig. 11. (See Fig. 15 for a summary of the SIGs of the study.)

In the second half of the second month, there was a discussion on the data model and the requirements document. The requirements document [60] (which had been prepared for the university by a software firm) was then reviewed by this author (the data model was also made available, but, due to misdelivery, was not received in time for this author to use it in the study). This resulted in SIGs being revised or initially prepared, including the SIG of Fig. 11 and the SIG of Figs. 19, 20, and 21. Further discussion at the end of the second month addressed trade-offs and the requirements document, including constraints. Finally, in the first half of the third month, the text of this paper was drafted.

Let us turn to the complexity of the SIGs developed for the study, summarized in the chart of Fig. 15. The figure shows the size of each SIG, given by the approximate number of softgoals per SIG, as well as the layers addressed in each SIG. When SIGs are presented using more than one figure, the statistics are combined into one entry in the chart. Note that portions of some SIGs are omitted from figures. Other SIGs, identified as X, Y, and Z in the chart, are omitted from this paper, Z being a composite of a few small SIGs.

All told, there were four SIGs with approximately 20 softgoals each and three SIGs with approximately 10 softgoals. It is important to note that these SIGs addressed only a few requirements and a few small portions of the SRS and then only in limited detail.

Each layer was addressed in at least one SIG (in addition to Fig. 11), with three layers being addressed in two or more SIGs. In addition to addressing specific layers, some SIGs have softgoals for which no layer is specified. This situation is indicated by a "general" layer entry in the chart.

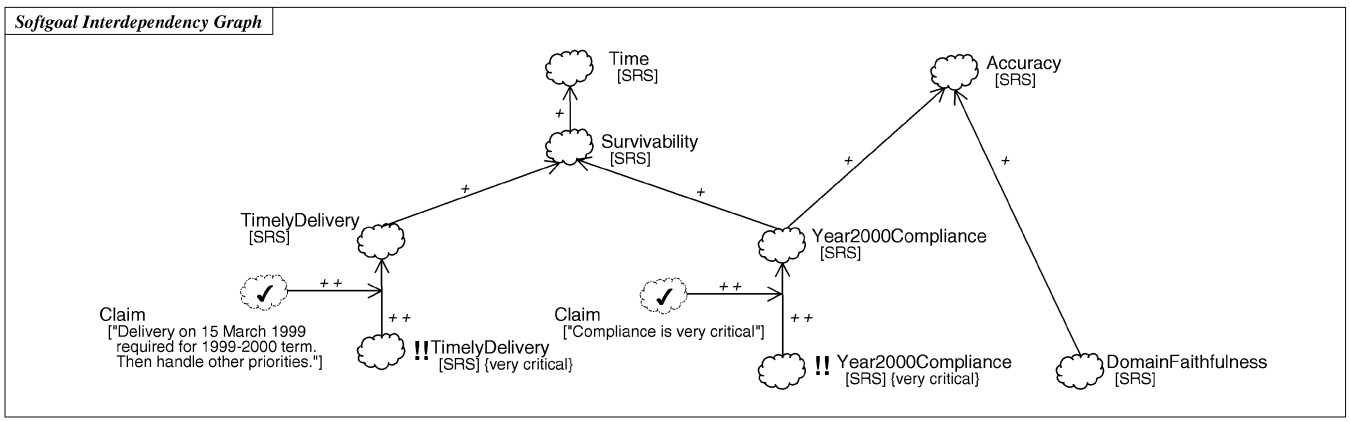


Fig. 16. Some NFRs for the new Student Records System (SRS).

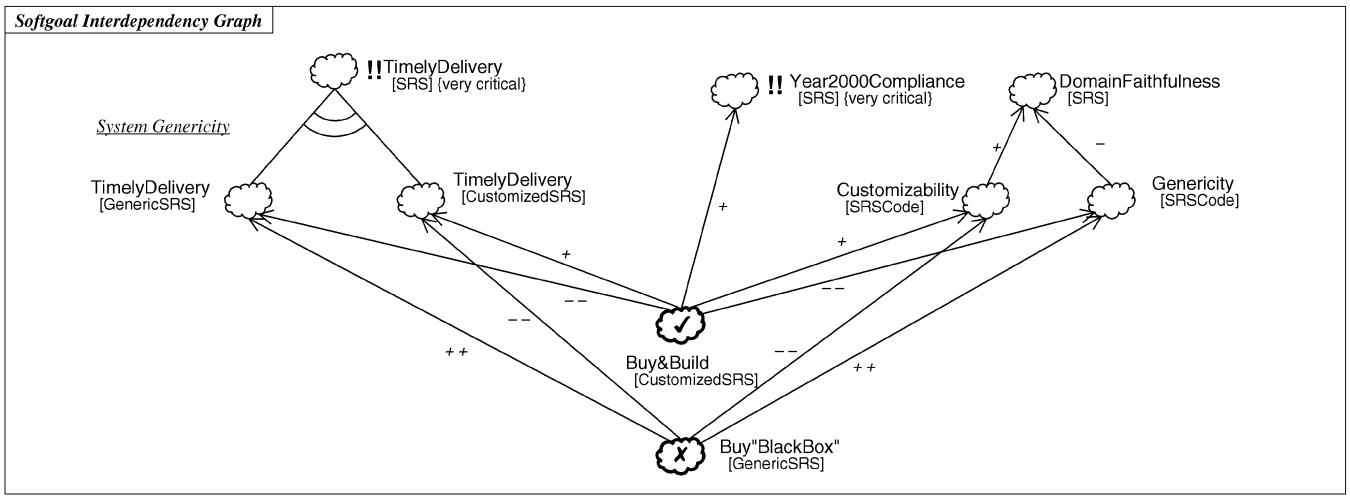


Fig. 17. Target alternatives and their trade-offs.

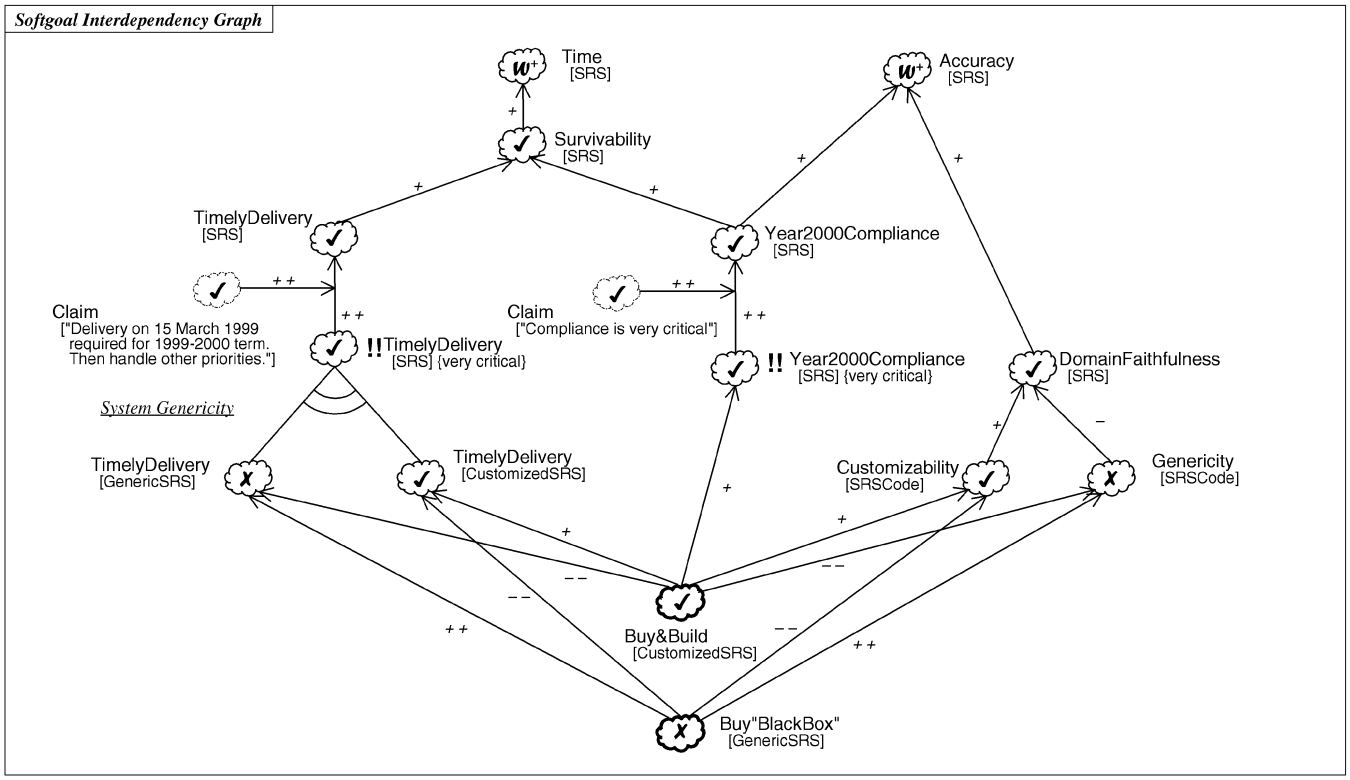


Fig. 18. Evaluating the impact of decisions upon NFRs.

The use of the evaluation procedure was straightforward for all SIGs, regardless of the number of layers or softgoals used. This is because a standard algorithm from the NFR Framework is used, along with developer input.

In the opinion of this author, the SIGs were able to be developed fairly rapidly once the domain and system information had been reviewed and discussed. The series of discussions with the domain expert were most helpful in identifying, clarifying, and focusing issues and trade-offs. As a result, the discussions greatly expedited the author's development of the SIGs.

The development of the SIGs was also expedited by other factors, including the availability of catalogues. In addition, the author's familiarity with the catalogues, having previously used them and compiled a large portion of them, facilitated things. Another expediting factor was the author's experience in conducting previous studies; for example, this aided the identification and treatment of trade-offs.

### 3.3 NFRs for Year 2000 Compliance

Let us consider some nonfunctional requirements for Year 2000 compliance of the new system. As was mentioned in Section 1, the actual domain priority for the new system was to make it Year 2000 compliant. As the old system was not compliant, the new system had to be in operation by 15 March 1999 in order to handle the 1999–2000 academic year.

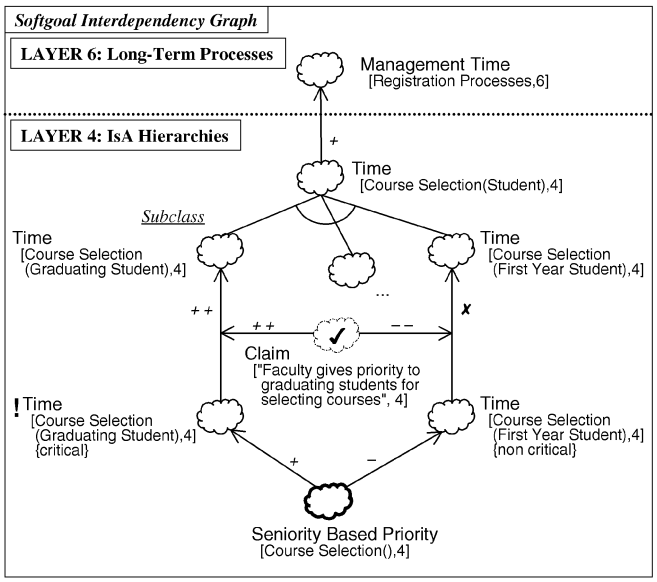


Fig. 19. Giving priority to graduating students.

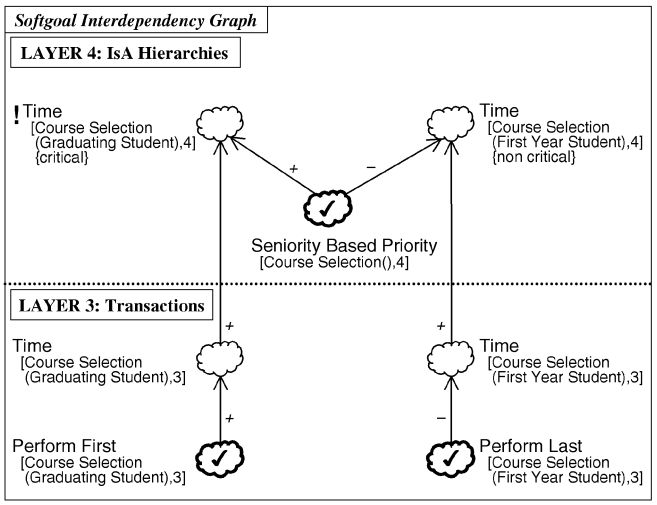


Fig. 20. Refining a softgoal to a lower layer and selecting operationalizations.

Accordingly, we state two NFR softgoals, Year2000-Compliance[SRS] and TimelyDelivery[SRS], in Fig. 16.

These two softgoals are identified as being very critical. Arguments for the prioritizations are stated using claim softgoals. Note that the content (topic) of the claim, e.g., "Delivery on ...," depends on the developer and cannot be syntactically determined by pattern substitution.

NFRs can have an impact on more general NFRs. Here, Year2000Compliance[SRS] *HELPS* both Survivability[SRS] and Accuracy[SRS]. Notice that we are extending the graph *upward*. In this way, the addition of these NFRs is somewhat of a retrospective reconstruction of the top-level NFR softgoals shown in the figure, as they are not the main NFRs in the domain. Similarly, TimelyDelivery[SRS] *HELPS* Survivability[SRS] since the old system would eventually crash if the new one were not delivered on time.

If the system cannot continue operating, there will not be any response, let alone response time. So, we view survivability as being essential, i.e., a sine qua non, for time performance. So, again working upward, Survivability[SRS] *HELPS* Time[SRS]. With Time[SRS] and Accuracy[SRS] now at the top of Fig. 16, we can view this figure as a continuation of Fig. 1.

### 3.3.1 Alternatives and Trade-Offs for the Target System

The university considered two main options for the target system. We show how the trade-offs can be recorded using PeRF.

One major option was to "buy and build." The university could have bought an existing Year 2000 compliant system from another university and then customized the code, giving them more control and the ability to develop what is critical to the University of Toronto.

Another main option was to buy a "black box," a commercial product, used by several universities. They decided against the black box because a lot of customization would be needed before Toronto could use it and the vendor had many clients waiting for customization. The vendor's product made assumptions about how the university works, hence, customization was needed to reflect the institutional complexity of the University of Toronto.

The use of the "buy and build" option definitely helped with timely delivery of the new Toronto system. Fig. 17, a downward continuation of Fig. 16, reflects these considerations for alternatives for the target system.

The selection of decomposition method may be influenced to a great extent by the known target alternatives. We add the operationalizing softgoals, Buy&Build[CustomizedSRS] and Buy"BlackBox"[GenericSRS]. Ignore for a while the contents of those clouds.

Now, from the domain information, the likelihood of timely delivery is quite different for the generic and the customized systems. So, the developer refines !!TimelyDelivery[SRS]{very critical} into two TimelyDelivery softgoals, one for the generic SRS, the other for the customized SRS. Since timely delivery of *either* system will satisfy !!TimelyDelivery[SRS]{very critical}, an *OR* interdependency is used.

Here, we are treating customizability as the opposite of "genericity." A truly generic system might not need customization since it covers all cases. For the SRS, however, customization of the generic system is needed.

Customizability and genericity of the system code also have impact on the ability of the system to accurately reflect the characteristics and priorities of the domain. We call this DomainFaithfulness in Fig. 17. In the case of the University of Toronto, this would mean that the system can handle many faculties, students registered for multiple degrees, and so on. Customizability of the code *HELPS* domain faithfulness, while genericity (perhaps a better term would be "inflexibility for customization") *HURTS* it. We view DomainFaithfulness as contributing to Accuracy. Observe that an NFR (here, domain faithfulness) came into play when considering the impact of different operationalizations, resulting in it being stated at one of the top levels of NFR softgoals.

Having refined several softgoals, we are now able to state the impact of the different operationalizations. Consider buying a generic "black box." Its chief strengths are

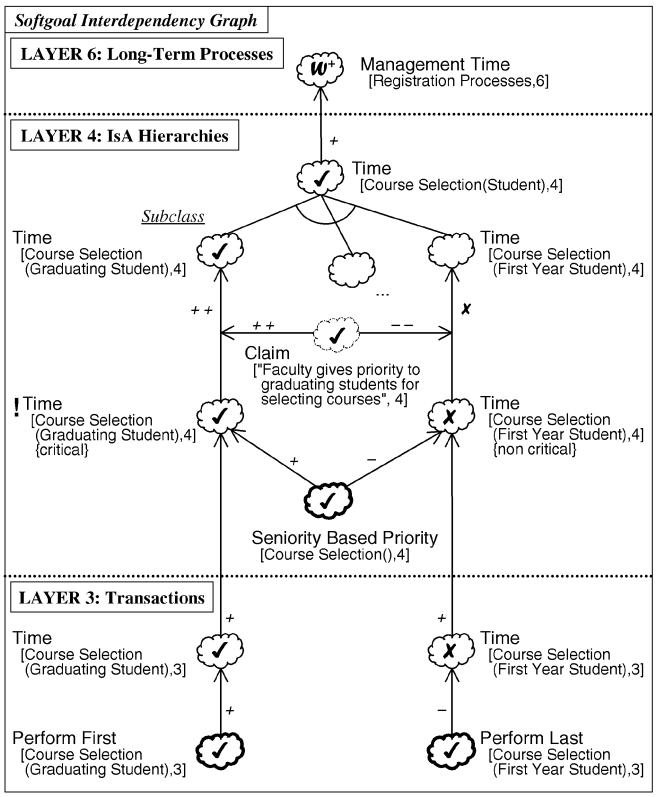


Fig. 21. Selection and evaluation at multiple layers of a graph.

that it is very good for timely delivery of a generic system and that its code is generic. These are shown by *MAKES* contributions. It addition, it *HELPS* contribute to Year 2000 compliance (this contribution is not shown to reduce the complexity of the figure). The main weakness of the "black box" is that it does not support customizability of code; in fact, it *BREAKS* it.

The main advantages of buying and building a customized system are that it *HELPS* contribute to customizability of the system and that it *HELPS* timely delivery of a customized system. In addition, "buy and build" definitely *HELPS* achieve Year 2000 compliance. Its main weakness is that is does not aid genericity of code; it actually *BREAKS* it.

Having done the analysis, resulting in the SIG, the developer may *select* or *reject* alternatives. Here, "buy and build" is selected and "black box" is rejected.

### 3.3.2 Evaluating the Impact of Decisions

We now need to determine the impact of decisions upon higher-level softgoals, using the *evaluation procedure.* In Fig. 18, which combines Figs. 16 and 17, we start with the operationalizing softgoals. If a parent receives more than one contribution from offspring, the propagated label values are combined to result in one label value.

Let us determine the impact of the decisions upon TimelyDelivery[GenericSRS]. Buy"BlackBox"[GenericSRS] is denied and has a *MAKES* contribution. Using the rule for *MAKES*, a denied label is propagated to the parent. At the same time, Buy&Build[CustomizedSRS] is satisficed and has a *BREAKS* contribution to TimelyDelivery[GenericSRS]. Using the rule for *BREAKS*, a denied label is propagated. Combining the two denied labels results in TimelyDelivery [GenericSRS] being denied. Using the same reasoning, Genericity[SRSCode] is also denied.

Now, we turn to TimelyDelivery[CustomizedSRS]. From both satisficed Buy&Build[CustomizedSRS] with a *HELPS* contribution and denied Buy"BlackBox"[GenericSRS] with a *BREAKS* contribution, weak positive labels are contributed to TimelyDelivery[CustomizedSRS]. The developer then decides to change it to satisficed. In the same way, Customizability[SRSCode] is satisficed.

!!TimelyDelivery[SRS]{veryCritical} is satisficed since one of its *OR* offspring is satisficed.

Satisficed Buy&Build[CustomizedSRS] *HELPS* !!Year2000-Compliance[SRS]{veryCritical}, resulting in a weak positive label. At the same time, denied Buy"BlackBox"[GenericSRS] *HELPS* !!Year2000Compliance[SRS]{veryCritical} (this interdependency link is not shown in the figure), resulting in a weak negative label. Now, the developer must deal with a conflict. Given that the two operationalizations are alternatives and that the selected one does make a positive contribution, the developer decides that !!Year2000 Compliance[SRS]{veryCritical} is satisficed.

Now, satisficed Customizability[SRSCode] *HELPS* DomainFaithfulness[SRS] and denied Genericity[SRSCode] *HURTS* DomainFaithfulness[SRS], each contribute a weak positive label. The developer changes the result to satisficed.

The two softgoals considered very critical by domain people have both been satisficed. This meeting of priorities is in keeping with an SPE principle. Then, their parents are satisficed.

The developer feels that the positive contributions for two different NFRs are enough to label Survivability[SRS] as satisficed. In turn, this gives a weak positive contribution to Time[SRS]. Accuracy[SRS] receives two weak positive contributions and the developer decides to leave the label as is since there are other factors contributing to accuracy which have not been considered.

To summarize, the decision about a target system has resulted in the priority softgoals of the domain being satisficed. We have also noted that analysis of NFRs may result in softgoals being added at the top of a SIG. In this case, there is some positive impact on these higher-level NFR softgoals, including Time[SRS] and Accuracy[SRS].

Note that if "black box" were selected instead of "buy and build," the difference in impact would be timely delivery of a generic, noncustomizable system. As a result, domain faithfulness and accuracy would suffer. This is the case, even though "black box" makes two strongly positive contributions, while "buy and build" makes two weakly positive ones and they both have the same number of negative contributions. Hence, it does not always suffice to simply sum up the positive and negative contributions when selecting among alternatives.

## 3.4 Prioritized Course Selection

We now consider a second NFR, for giving priority to graduating students for selecting courses.

### 3.4.1 Processes

We assume that there is a softgoal for good ManagementTime for the registration processes. It is shown in Fig. 19, which is a continuation of Layer 6 of Fig. 11.

#### 3.4.2 Handling IsA Hierarchies at a Lower Layer

In addition to time spent for managing the registration processes, there is the time spent actually handling the course selection for a student. We consider that good time performance for course selection *HELPS* management time for the registration processes. The interdependency between Layers 6 and 4 is an example of a ManagementTime To Time decomposition. It is also an example of an *interlayer interdependency link.*

Now, to handle priority for graduating students, we first decompose the time softgoal for selecting a course for a student into one softgoal for course selection for graduating students, another for third year students, etc. We assume that students of different years are modeled using IsA hierarchies. Hence, this refinement is shown at Layer 4 and is a Subclass decomposition since GraduatingStudent and the other groups of students are subclasses of Student. If, however, the different years are modeled without using IsA hierarchies, we would still need to distinguish graduating students from others, so some kind of refinement will be needed and the analysis shown will still hold, even using a different model for years of students.

Now, using Prioritization, the time softgoal for graduating students is noted as being critical. In Fig. 20, we consider an assumed operationalization method at Layer 4, SeniorityBasedPriority, which gives registration priority to students based on seniority. This will *HELP* time for selecting courses for graduating students, but *HURT* time for first-year ones.

#### 3.4.3 Handling Operationalizations at at Multiple Layers

Now, we have an operationalization at Layer 4 and want to continue refinements at lower layers. For example, we wish to deal with transactions (without IsA hierarchies) at Layer 3.

Can we refine parent SeniorityBasedPriority[Course Selection(), 4] into offspring including Time[CourseSelection (GraduatingStudent), 3] and Time[CourseSelection (FirstYearStudent), 3]? The answer is no, even if the parent and offspring are at the same layer.

The reason is that operationalizing softgoals may be refined into other operationalizing softgoals, but the NFR Framework does not allow operationalizing softgoals to be refined into NFR softgoals. To deal with this, we use an approach suggested by Chung, using the fact that NFR softgoals may be refined into NFR softgoals and into operationalizing softgoals.

An NFR softgoal at Layer 4, such as Time[CourseSelection (GraduatingStudent), 4], is refined into a operationalizing softgoal at that layer, SeniorityBasedPriority[CourseSelection(), 4], and into an NFR softgoal at a lower layer, Time[CourseSelection(GraduatingStudent), 3]. This allows further refinement at various layers, here Layer 3 and below, in a manner consistent with the NFR Framework.

So, the NFR softgoals at Layer 3 are refined. They could be refined into other NFR softgoals, but we just show them being refined into operationalizations at Layer 3. Again giving priority, PerformFirst is used for graduating students, and PerformLast for first-year students (we omit topics of softgoals in the text where there is no ambiguity in figures). Here, PerformFirst is used in the sense of not locking out graduating students, while PerformLast locks out first-year students during the first weeks of registration. Here, FirstComeFirstServed might be used to order incoming transactions that are not locked out.

In Fig. 21, we select (hence, satisfice) SeniorityBasedPriority at Layer 4 and PerformFirst[CourseSelection(GraduatingStudent), 3] and PerformLast[CourseSelection(FirstYearStudent), 3] at Layer 3.

#### 3.4.4 Interlayer Evaluation

In Fig. 21 we start with the operationalizations at Layer 3 which are selected, resulting in weak positive support for Time[CourseSelection(GraduatingStudent), 3], which the developer considers satisficed. Similarly, Time[CourseSelection (FirstYearStudent), 3] is considered denied.

Now, up at Layer 4, !Time[CourseSelection(GraduatingStudent), 4]{critical} receives weak positive support from Time[CourseSelection(GraduatingStudent), 3] at Layer 3 *and from* SeniorityBasedPriority[CourseSelection(), 4] at Layer 4, so we consider the critical softgoal at Layer 3 to be satisficed. In a similar way, the noncritical softgoal at Layer 3 is considered denied.

Consistent with an SPE principle, we have focused on meeting the priorities, even if nonpriorities are not met. As a result, we consider the overall Layer 4 softgoal for all students, Time[CourseSelection(Student), 4] to be satisficed. Note that using a strictly AND/OR type of graph, the results would be different: Using a logic-based mechanism, we would conclude that the overall time and space softgoals are denied because they have *AND*-offspring which are denied.

### 3.5 Study Results

The result of this study is an analysis and interrelating of requirements, priorities, target techniques, design rationale, and decisions for the SRS. We were able to perform this analysis using the existing PeRF framework and record the results in graphs. We were able to produce graphs which, when the evaluation procedure was applied, yielded results which reflected some of the priorities in the domain. We feel that the study gives evidence that the framework can represent reasoning about performance and other NFRs.

We did not predict performance in this study. Of course, the university developers may well have done so for their system.

During the study, the domain expert (who did not have prior knowledge of the framework) and this author discussed the domain and the system. During discussions, this author did not go into details about the graphs. After reading a draft of this paper and seeing the graphs of the study for the first time, the domain expert found it very interesting to see drawn out in detail the thought processes that the expert and the author had gone through.

## 4 Other Empirical Studies

To consider the applicability of the framework to a *variety* of types of information systems, we conducted empirical studies (see Fig. 22) which applied PeRF to portions of a

variety of information systems. The studies address a variety of domains which exhibit a variety of organizational workloads, priorities, and other characteristics. The studies also addressed a variety of data model features and their associated implementation techniques.

The studies illustrated how a developer could deal with performance requirements (and other NFRs) by using the framework and its components to build softgoal interdependency graphs to record the analysis, rationale, and development decisions that might be made in developing such systems.

### 4.1 Domains Studied

We conducted studies of three domains: *A credit card authorization system* [46] which authorizes transactions and cancels stolen cards, *an income tax appeals system* [47] which tracks the progress of tax appeal cases, and a *a bank loan system* addressing changes in performance and other nonfunctional requirements [18].

A bank's credit card system maintains information on cardholders and merchants. In a highly competitive market, it is important to provide fast response time for sales authorizations. To reduce losses due to fraud, lost and stolen cards must be invalidated as soon as the bank is notified.

An Income Tax Appeals system manages the handling of appeals by taxpayers who disagree with decisions made by the Canada Customs and Revenue Agency (formerly the Department of National Revenue). During the appeals process, a taxation officer should contact the taxpayer and may also consult other staff members. This is a highly consultative and interactive processes, subject to constraints, and may last months or even years. The existing system records and tracks information and decisions over a long period of time. We considered [47] a possible extension to this government information system.

A bank loan system maintains loan accounts, calculating interest rates and loan balances. We considered a bank's NFRs of informativeness, time, accuracy, etc., and how it dealt with changes in policies, NFRs, etc.

The systems exhibit a variety of characteristics (see Fig. 22 for a summary of the studies [46], [47], [18], [17], [48], [19]). They include commercial and governmental domains. Short-term and interactive operations, long-term and consultative operations, and batch data processing were used, individually or in combination. The studies address performance requirements, time often being most important. Priorities and trade-offs among requirements vary from domain to domain.

### 4.2 Methodology for Studies

First, we acquired knowledge of performance requirements and developed catalogues.

Then, we acquired domain knowledge. This was drawn from documents which varied in their nature and degree of detail. They included policy manuals, statistical reports, and annual reports. The available statistics gave an indication of organizational workload. We generally did not have access to system specification documents; however, in one case, the operations policy manual gave an indication of the schema. We tried to be faithful to such documentation: In some cases, we supplement the documentation with creative imagination based on our understanding of the domain.

| **Study** | **Tax Appeals** [47], [48], [17] | **Credit Cards** [46], [48], [17] | **Bank Loans** [18] |
|---|---|---|---|
| **Domain** | Government | Commercial | Commercial |
| *Characteristics* | long-term, consultative | interactive | batch, interactive, consultative |
| *Functional Requirements* | - update & retrieve status<br>- statistics | -authorization<br>- cancellation | - calculate rates & balances<br>- statements |
| **NFRs Considered** | time, space, (actually accuracy) | time, space | informativeness, time, accuracy, confidentiality |
| *Priorities* | - timely reminders<br>- fast access to info | - cancellation<br>- authorization | - change rate quickly<br>- enhance statements |
| *Tradeoffs* | - time-space<br>- (actually: time-accuracy) | - time-space | - time-accuracy<br>- replicated data vs. central storage |
| **Workload:**<br>*- Operations*<br>*- Storage* | <br>- long-term processes<br>- fairly few appeals | <br>- very many short transactions<br>- many cardholders | <br>- (frequent) interest & balance calculations<br>- many statements |
| *Data Model Features Considered* | long-term processes integrity constraints, attributes | IsA hierarchies, transactions, attributes | transactions, attributes |
| **Matters Illustrated** | - handle (assumed) priorities, tradeoffs<br>- represent domain info. using tool | - hybrid techniques | - dealing with change |
| *SPE Principles* | centring (focus on priorities) | centring (order of operations) | centring (focus on priorities) |
| **Domain Documents** | - operations policy manual<br>- detailed statistics | - statistical summary<br>- annual reports | - loan policy manual<br>- annual reports |
| **Domain Feedback** | full interview | initial interview only | none |

Fig. 22. Overvew of other empirical studies.

Then, we went through an iterative process of applying the framework, identifying NFRs, stating softgoals, and refining and interrelating them, providing design rationale, and selecting and evaluating operationalizations. This process was done at various layers of graphs. The order of the above steps sometimed varied.

To obtain some feedback on the framework and studies, we interviewed people familiar with some of these and other domains studied. They were interviewed after the studies and were not involved during the preparation of the studies.

The studies were were carried out "off-line," without our involvement as actual participants during the development process. We ourselves did not develop the systems, but did consider performance (and other quality) issues for such systems.

We had contact with the organizations at the start of the studies, when we obtained their documentation, and then at the end, when we obtained feedback. During the actual studies, we examined their documentation, but generally did not communicate with them.

More detail on methodology is available [17], [48].

### 4.3 Description and Results of Studies

The credit card study used statistics on card usage, which indicated a high daily volume of transactions and a large number of customers. We considered an assumed time-space trade-off between processing time and storage of customer transaction information.

One SIG addressed fast cancellation of lost cards. It used 28 softgoals, at Layers 3 and 2. Another SIG address space requirements for cardholder information. It used 15 softgoals, at Layers 4, 2, and 1. We used IsA hierarchies to model different kinds of accounts (e.g., gold cards and regular cards). During the study, we were also developing the framework and the catalogues, including methods for Layers 4 through 1. This study was done before the others.

The income tax appeals system has a relatively low number of appeals. However, they last a long time—months or even years—and are subject to a number of constraints imposed by legislation or government policy. Accordingly, we used this study to extend the catalogues to deal with long-term processes and integrity constraints at Layers 6 and 5. We considered a proposed enhancement to the system to provide a reminder service to staff in order to help expedite the handling of appeals. The SIG which addressed this matter used 28 softgoals, at Layers 6, 5, and 2.

The domain information included detailed statistics on the volume and duration of appeals, as well as a policy manual on the handling of appeals. After this study, we had an interview with a domain expert. We had considered an assumed time-space trade-off; in fact, space was not a major issue due to the small number of appeals; however, there was an important time-accuracy trade-off that we had not considered. From this experience, we saw the need for a more interactive approach to consulting domain experts (which was done for the SRS study).

The credit card and income tax studies used the same layering as is described in this paper.

A bank loan policy manual guided the loan system. The bank changed the policy manual to provide, inter alia, the customer with more informative statements. We considered an assumed improvement to statements to provide more details on loan rates.

First, we prepared a SIG using the initial policy manual. Given this initial SIG, we found that we could very rapidly produce a revised SIG to reflect the situation with the revised policy manual. This was the case even though the revised SIG addressed a change in priorities, a change in functional requirements (additional detail on statements), a change (addition) of NFRs (e.g., time requirements to produce detailed statements), and a change in operationalizations. The initial SIG had 21 softgoals and the revised SIG had six additional softgoals and 11 additional interdependency links. We were also able to handle interactions and trade-offs among a variety of NFRs. The SIGs involved several NFRs, performance and others, and did not specifically use layering. However, the SIGs drew on performance types and methods and addressed issues typically handled at Layers 3 and 2. This study also benefitted from existing catalogues for a few NFRs (performance, accuracy, security) and our experience in conducting previous studies involving these NFRs.

The credit card and tax appeals studies took relatively longer to conduct than subsequent ones. In part, this was because we were augmenting catalogues while conducting studies and were gaining experience in using the framework for the first times. We also tried to address data model features for each of the six layers.

Augmenting catalogues involves considering features of a new method (e.g., is it a decomposition, operationalization, or argumentation method?), considering what types, topics, and priorities it involved, determining if it addresses a particular layer or group of layers and/or addresses particular data model features, determining what information system features it addresses, and determining which performance concepts it addresses. Augmenting catalogues while conducting studies was a somewhat iterative process; e.g., in looking at a SIG, we would consider a new method, apply it to the SIG, and add it to the catalogue; on the other hand, changing a method entry in the catalogue would lead to changing SIGs which had already used the method.

One benefit of the tax appeals study was that we further organized the catalogues and also recorded the catalogues using a software tool. Much of the material in the method catalogues shown earlier in Figs. 7, 8, 9, and 10 was produced during the taxation study (which was done after, and drew upon methods used in, the credit card study.)

### 4.4 Domain Feedback

We consulted some domain experts to review some of these studies and some other studies of NFRs to obtain feedback on the framework and studies [17]. The feedback indicates some initial support, while pointing out room for improvement and fruitful research. From the perspective of the domain experts, the NFR Framework would be helpful for developers. The cataloguing of development techniques and NFR-specific knowledge would be helpful and, in some cases, is more advanced than current practice in the domains considered. Our studies were based on their domains and illustrated our approach. However, the detailed conclusions in the studies are not necessarily what would have been reached by the organizations studied: This seems mainly due to our use of source documents without ongoing contact with domain experts during our study.

We felt that future studies could benefit from a more interactive and consultative approach, involving repeated contact early in the study to ensure that we have appropriately captured the priorities for requirements. Thus, we have tried to do so in conducting the present SRS study.

In addition to studies, the NFR Framework has been applied to other areas, including software architecture [16] and case-based reasoning [30]. A variety of NFRs, including performance requirements, are considered in these applications.

## 5 Conclusions

We conclude by discussing contributions, related work, and directions for the work.

### 5.1 Contributions

This paper has presented a framework for managing performance requirements for information systems. This developer-directed approach allows customized solutions to be built, taking into account the characteristics (such as needs, priorities, and workload) of the particular domain. The Performance Requirements Framework integrates and catalogues knowledge of performance concepts, Software Performance Engineering principles for "building performance into" systems, layered structures for organizing the development process, and characteristics of information systems, their design, implementation, and performance. This knowledge is represented using the NFR Framework, an existing goal-oriented approach offering facilities for interactively and iteratively stating NFRs, analyzing and interrelating them, and determining the impact of decisions upon NFRs, and recording all the developer's actions in Softgoal Interdependency Graphs.

We have used the Performance Requirements Framework to study performance requirements for a real domain, a university's student records system. The study used actual domain and workload information where possible and indicated where assumptions were made. The study illustrated how the Framework can be used to represent and reason with NFRs, deal with trade-offs and priorities using SPE principles, and deal with a variety of data model features using a layered structure. Interestingly, the study addressed performance requirements as well as other NFRs (e.g., accuracy) together, using the NFR Framework's graphical representation.

### 5.2 Related Work

The NFR Framework was influenced by the DAIDA environment for information system development [28], [6], which also addresses implementation of conceptual designs, but focuses on correctness rather than performance. It also deals with the "mapping" of conceptual designs to database implementations, focusing on helping the developer establish the *correctness* of refinement steps. For the future, one can envision an hybrid implementation tool which would address the complementary issues of performance and correctness.

A framework for performance engineering for information systems is presented by Opdahl [51]. Focusing on the prediction and improvement of performance of information systems during development, it incorporates models of software, hardware, the organization, and the applications. It offers sensitivity analysis and a number of tools and is integrated into an environment for a larger performance evaluation project [8]. The project was also applied to workflow systems [9]. Opdahl [52] argues for quantifying performance demands during requirements specification.

It is interesting to contrast the treatment offered in this work with other research based on the transformational approach, such as the TI system [3]. TI, focuses on making sure that the generated implementation is consistent with the original specification. Performance, if treated at all, is treated as a selection criterion among alternative transformations. Kant and Barstow's early work [31], on the other hand, does address performance goals. Their framework, however, focuses on conventional programming-in-the-small rather than information system development, relies on quantitative performance measures (which are available for their chosen domain but are, unfortunately, not always available for information systems because of their complexity), and assumes an automatic programming setting rather than the dialectical software development process adopted here.

Glitter [22] automates parts of using the transformation implementation approach. It takes a goal-oriented, formal approach, using problem solving to automatically find and apply portions of transformations.

Glitter tries to automate many tasks, leaving small portions of tasks to developers. In this regard, it is more automated than the NFR Framework. Both Glitter and the NFR Framework represent such concepts as goals, methods, and design rationale.

For automating physical database design, for relational and postrelational databases, Rozen [54] offers a framework and a tool. Sets of design options which do not conflict with each other are considered and low-cost designs are found by an approximation algorithm.

Preliminary evidence that the NFR Framework can improve the software development process is presented in [20], which describes an adaptation of the NFR Framework and an associated case study.

A full presentation of Performance Requirements Framework is presented in a thesis [48], with much of it also presented as part of a recent book [19]. An early conceptual framework for this work was presented in [45], which drew on earlier experience in implementing SDMs [41], [42], [43], [44]. The development of the Performance Requirements Framework continued in part of [39] and then in [46], [47]. This paper uses the newer notation and terminology of [19] and presents a new case study, conducted with more interaction with a domain expert than was done during several earlier studies [46], [47], [17], [18].

### 5.3 Directions for the Work

We discuss some initial work that has been done for a developer's tool and performance prediction and conclude by mentioning some future work.

#### 5.3.1 A Developer's Tool

A partial developer's tool, the Performance Requirements Tool, helps organize a variety of types of knowledge for use by a developer. This knowledge is recorded using a knowledge representation language, Telos [38]. The tool is part of a family of "NFR Tools" (e.g., [13], [15]) intended to help a developer use the NFR Framework. Some versions of the Tools used the knowledge base facilities of ConceptBase [29] and one version used Telos tools developed by Stanley and Kramer at the University of Toronto.

Additional work is need to extend the Performance Requirements Tool to offer the operational facilities offered by other NFR Tools, to allow construction of softgoal interdependency graphs, detection of correlations, and evaluation of results. This would involve the provision of implementations for the methods which are already defined. Performance prediction facilities should also be added to the Performance Requirements Tool.

### 5.3.2 Performance Prediction

An initial start on performance prediction was made [45] by identifying input and output values for prediction, which is structured using the layering described in Section 2. Each layer addresses certain performance issues through a performance model which takes input values and produces performance measures. Outputs from one layer can be used as inputs to lower layers. However, much more work is needed on this topic. We would need to develop more formulae for predicting the output values given the inputs, provide a prediction tool, and validate prediction results using actual workloads.

Let's consider performance prediction for IsA hierarchies, which are handled at Layer 4. The resulting prediction problems are expressed in terms of entities, classes, attributes, and transactions, which are handled at lower layers. Thus, Layer 4 would handle performance prediction for the different implementations of the "staircase" of attributes introduced in Section 2. Inputs would include information about the schema (such as the number of classes and the average number of attributes in a class) and the instances (e.g., the total number of entities, such as students). Given the particular choice of representation of attribute inheritance (e.g., horizontal or vertical splitting), Layer 4 can then calculate the expected number of tuples needed in the running system and the size of each tuple. Lower layers can then predict the times needed for typical operations (e.g., creating an entity or retrieving an attribute). Further details are given in [45].

The Framework could be extended to deal with the combination of quantitative and qualitative reasoning. We would expect quantitative performance prediction to help qualitative selection among alternatives and vice versa (Fig. 23). Prediction results can be used as inputs to the process of selecting a target system (e.g., by using prediction results to choose one implementation over another). In the other direction, the results of selecting a system can be used as input to the predictor in order to estimate the performance of a selected system.

### 5.3.3 Future Work

We see a number of directions for future work.

*Cataloguing and Retargetting.* We would like to catalogue more knowledge about performance and development techniques. One concern is that the existing catalogues may be too language-specific. We could try to retarget the approach to other families of languages. Similarly, we could consider treating performance at the requirements-to-design phase (rather than during design-to-implementation).[1]

*Support for Common Situations.* It would be helpful to identify and define "templates" (portions of softgoal interdependency graphs) for commonly occurring patterns of decomposition, using case-based reasoning to review a library of softgoal interdependency graphs and search for patterns.[2]

*Studies.* We would like to conduct larger studies, covering a larger variety of systems. Another avenue would be to use the framework independently to develop a system, and compare the results with the domain's development team.

1. Thanks to Gunnar Brataas for this suggestion and others.
2. Thanks to Igor Jurisica for suggesting this.

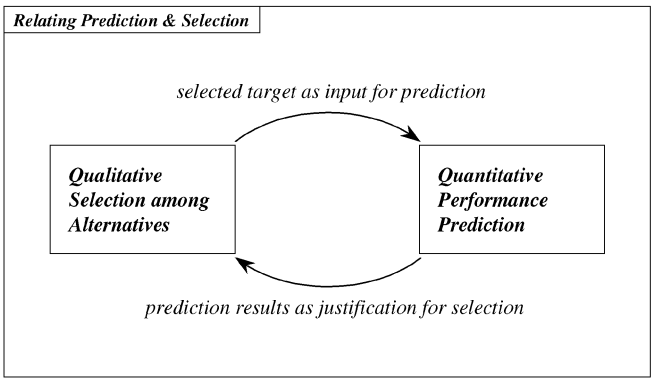


Fig. 23. Relating prediction and selection.

Studies and the gathering of feedback should be designed with an appropriate methodology so that results could be analyzed with rigour [35].

The final test of such research, of course, will be whether it aids developers in building complex information systems while addressing performance requirements. Hence, to establish results, we need to involve developers in the review, testing, and refinement of techniques and tools that we offer.

## ACKNOWLEDGMENTS

The author would like to thank Professor John Mylopoulos for ongoing direction for this work over the years. He also appreciates his suggestion of conducting a new study for this paper. The author extends gratitude toward the members of his thesis supervision committee, including Ken Sevcik, who introduced him to SPE. Thanks to Lawrence Chung for a wealth of help on the issue of dealing with nonfunctional requirements and Eric Yu for helpful discussions. He also wishes to thank Matthias Jarke, Thomas Rose, David Lauzon, Martin Stanley, and Bryan Kramer for their help with tool implementation. Although tools are not detailed in this paper, their work helped him to refine some of the ideas of the Framework.

Sincere thanks to Dr. Eva V. Swenson, director, Student Information Systems, University of Toronto, for sharing her wealth of knowledge thoroughly, quickly, and cheerfully. Her tremendous help, by way of discussion, correspondence, and provision of documentation, made the study of the Student Records System possible. Thanks also to Donna George for providing university registration information for the study.

Thanks to Elizabeth D'Angelo for her excellent work, who contributed to this research by carefully preparing diagrams, making the presentation consistent, and correcting errors. The style of figures is taken from [19], developed by the book authors and Elizabeth D'Angelo.

The author also thanks the organizers of WOSP 1998 for arranging a most enjoyable workshop. He wishes to sincerely thank Professors Murray Woodside, Paul Clements, and Albert Cheng for taking the initiative to have a follow-up collection of journal papers and for all their work, kindness, and patience in handling the reviewing process. He also extends gratitude toward the TSE and WOSP

reviewers for their thorough work and most helpful comments and to the organizers and reviewers of the RE '97 doctoral consortium. Many thanks to Michael Brodie and Ken Sevcik for their insights on the use of performance requirements in industry.

The author gratefully acknowledges a postdoctoral fellowship at the Department of Computer Science, University of Toronto, to conduct the case study reported herein. This paper includes material adapted from [50] (which was based on [49] and [48]) and [48].

Finally, the author thanks Heaven for his dear family—a constant source of love and encouragement.

## REFERENCES

[1] *Artificial Intelligence,* vol. 24, nos. 1–3, Dec. 1984.
[2] M.P. Atkinson and O.P. Buneman, "Types and Persistence in Database Programming Languages," *Computing Surveys,* vol. 19, no. 2, pp. 105–190, June 1987.
[3] R. Balzer, "A 15 Year Perspective on Automatic Programming," *IEEE Trans. Software Eng.,* vol. 11, no. 11, pp. 1,257–1,268, Nov. 1985.
[4] V.R. Basili and J.D. Musa, "The Future Engineering of Software: A Management Perspective," *Computer,* vol. 24, no. 9, pp. 90–96, Sept. 1991.
[5] B.W. Boehm, J.R. Brown, H. Kaspar, M. Lipow, G.J. MacLeod, and M.J. Merritt, *Characteristics of Software Quality.* Amsterdam: North-Holland, 1978.
[6] A. Borgida, J. Mylopoulos, J.W. Schmidt, and I. Wetzel, "Support for Data-Intensive Applications: Conceptual Design and Software Development," *Proc. Second Int'l Workshop Database Programming Languages,* R. Hull, R. Morrison, and D. Stemple, eds., pp. 258–280, 1990.
[7] A. Borgida, J. Mylopoulos, and J.W. Schmidt, "The TaxisDL Software Description Language," *Database Application Eng. with DAIDA,* M. Jarke, ed., pp. 65–84, 1993.
[8] G. Brataas, A.L. Opdahl, V. Vetland, and A. Sølvberg, "Information Systems: Final Evaluation of the IMSE," technical report, IMSE Project Deliverable D6.6–2, SINTEF, Univ. of Trondheim, Norway, Feb. 1992.
[9] G. Brataas, "Performance Engineering Method for Workflow Systems: An Integrated View of Human and Computerised Work Processes," doctoral thesis, Norwegian Univ. of Science and Technology, 1996.
[10] P.P.-S. Chen, "The Entity-Relationship Model—Toward a Unified View of Data," *ACM Trans. Database Systems,* vol. 1, no. 1, pp. 9–36, Mar. 1976.
[11] K.L. Chung, "An Extended Taxis Compiler," masters thesis, Dept. of Computer Science, Univ. of Toronto, 1984.
[12] K.L. Chung, D. Rios-Zertuche, B.A. Nixon, and J. Mylopoulos, "Process Management and Assertion Enforcement for a Semantic Data Model," *Proc. Int'l Conf. Extending Database Technology, EDBT '88,* J.W. Schmidt, S. Ceri, and M. Missikof, eds., pp. 469–487, 1988.
[13] K.L. Chung, "Representing and Using Non-Functional Requirements: A Process-Oriented Approach," doctoral thesis, Dept. of Computer Science, Univ. of Toronto, June 1993.
[14] L. Chung, B.A. Nixon, and E. Yu, "Using Quality Requirements to Systematically Develop Quality Software," *Proc. Fourth Int'l Conf. Software Quality,* Oct. 1994.
[15] L. Chung and B.A. Nixon, "Tool Support for Systematic Treatment of Non-Functional Requirements," Dec. 1994.
[16] L. Chung, B.A. Nixon, and E. Yu, "Using Non-Functional Requirements to Systematically Select Among Alternatives in Architectural Design," *Proc.Workshop Architectures for Software Systems, ICSE-17,* Apr. 1995.
[17] L. Chung and B.A. Nixon, "Dealing with Non-Functional Requirements: Three Experimental Studies of a Process-Oriented Approach," *Proc. 17th Int'l Conf. Software Eng.,* pp. 25–37, Apr. 1995.
[18] L. Chung, B.A. Nixon, and E. Yu, "Dealing with Change: An Approach Using Non-Functional Requirements," *Requirements Eng.,* vol. 1, no. 4, pp. 238–260, 1996.
[19] L. Chung, B.A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering.* Boston: Kluwer Academic, 2000.
[20] L.M. Cysneiros and J.C. Sampaio do Prado Leite, "Integrating Non-Functional Requirements into Data Modeling," *Proc. Int'l Symp. Requirements Eng.,* 1999.
[21] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal-Directed Requirements Acquisition," *Science of Computer Programming,* vol. 20, pp. 3–50, 1993.
[22] S.F. Fickas, "Automating the Transformational Development of Software," *IEEE Trans. Software Eng.,* vol. 11, no. 11, pp. 1,268–1,277, Nov. 1985.
[23] S. Green, "Goal-Driven Approaches to Requirements Engineering," Technical Report 94/9, Dept. of Computing, Univ. of the West of England, Bristol, 1994.
[24] S.J. Greenspan, J. Mylopoulos, and A. Borgida, "Capturing More World Knowledge in the Requirements Specification," *Proc. Sixth Int'l Conf. Software Eng.,* pp. 225–234, 1982.
[25] R. Hull and R. King, "Semantic Database Modeling: Survey, Applications, and Research Issues," *Computing Surveys,* vol. 19, no. 3, pp. 201–260, Sept. 1987.
[26] W.F. Hyslop, "Performance Prediction of Relational Database Management Systems," doctoral thesis, Dept. of Computer Science, Univ. of Toronto, 1991.
[27] R. Jain, *The Art of Computer System Performance Analysis.* New York: Wiley, 1991.
[28] M. Jarke, J. Mylopoulos, J.W. Schmidt, and Y. Vassiliou, "DAIDA: An Environment for Evolving Information Systems," *ACM Trans. Information Systems,* vol. 10, no. 1, pp. 1–50, Jan. 1992.
[29] *ConceptBase V3.1 User Manual,* M. Jarke, ed., Univ. of Passau, 1992.
[30] I. Jurisica and B.A. Nixon, "Building Quality into Case-Based Reasoning Systems," *Proc. 10th Int'l Conf. Advanced Information Systems Eng., CAiSE '98,* B. Pernici and C. Thanos, eds., pp. 363–380, June 1998.
[31] E. Kant and D.R. Barstow, "The Refinement Paradigm: The Interaction of Coding and Efficiency Knowledge in Program Synthesis," *IEEE Trans. Software Eng.,* vol. 7, no. 5, pp. 458–471, Sept. 1981.
[32] S.E. Keller, L.G. Kahn, and R.B. Panara, "Specifying Software Quality Requirements with Metrics," *Tutorial: System and Software Requirements Eng.,* R.H. Thayer and M. Dorfman, eds., pp. 145–163, 1990.
[33] W. Kim, K.-C. Kim, and A. Dale, "Indexing Techniques for Object-Oriented Databases," *Object-Oriented Concepts, Databases, and Applications,* W. Kim and F.H. Lochovsky, eds., pp. 371–394, 1989.
[34] E.D. Lazowska, J. Zahorjan, G.S. Graham, and K.C. Sevcik, *Quantitative System Performance.* Englewood Cliffs, N.J.: Prentice Hall, 1984.
[35] A.S. Lee, "A Scientific Methodology for MIS Case Studies," *MIS Quarterly,* pp. 30–50, Mar. 1991.
[36] J. Lee, "Extending the Potts and Bruns Model for Recording Design Rationale," *Proc. 13th Int'l Conf. Software Eng.,* pp. 114–125, May 1991.
[37] J. Mylopoulos, P.A. Bernstein, and H.K.T. Wong, "A Language Facility for Designing Database-Intensive Applications," *ACM Trans. Database Systems,* vol. 5, no. 2, pp. 185–207, 1980.
[38] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis, "Telos: Representing Knowledge About Information Systems," *ACM Trans. Information Systems,* vol. 8, pp. 325–362, Oct. 1990.
[39] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and Using Non-Functional Requirements: A Process-Oriented Approach," *IEEE Trans. Software Eng.,* vol. 18, no. 6, pp. 483–497, June 1992.
[40] N. Nilsson, *Problem-Solving Methods in Artificial Intelligence.* New York: McGraw-Hill, 1971.
[41] B.A. Nixon, "A Taxis Compiler," master's thesis, Dept. of Computer Science, Univ. of Toronto, 1983.
[42] B. Nixon, L. Chung, D. Lauzon, A. Borgida, J. Mylopoulos, and M. Stanley, "Implementation of a Compiler for a Semantic Data Model: Experiences with Taxis," *Proc. ACM SIGMOD '87,* U. Dayal and I. Traiger, eds., pp. 118–131, May 1987.
[43] B.A. Nixon, K.L. Chung,, D. Lauzon, A. Borgida, J. Mylopoulos, and M. Stanley, "Design of a Compiler for a Semantic Data Model," *Foundations of Knowledge Base Management,* J.W. Schmidt and C. Thanos, eds., pp. 293–343, 1989.
[44] B. Nixon and J. Mylopoulos, "Integration Issues in Implementing Semantic Data Models," *Advances in Database Programming Languages,* F. Bancilhon and P. Buneman, eds., 1990.

[45] B. Nixon, "Implementation of Information System Design Specifications: A Performance Perspective," *Proc. Third Int'l Workshop Database Programming Languages: Bulk Types and Persistent Data,* P. Kanellakis and J.W. Schmidt, eds., pp. 149–168, Aug. 1991.
[46] B.A. Nixon, "Dealing with Performance Requirements during the Development of Information Systems," *Proc. IEEE Int'l Symp. Requirements Eng.,* pp. 42–49, Jan. 1993.
[47] B.A. Nixon, "Representing and Using Performance Requirements during the Development of Information Systems," *Proc. Fourth Int'l Conf. Extending Database Technology, EDBT '94,* M. Jarke, J. Bubenko, and K. Jeffery, eds. pp. 187–200, Mar. 1994.
[48] B.A. Nixon, "Performance Requirements for Information Systems," doctoral thesis, Dept. of Computer Science, Univ. of Toronto, 1997.
[49] B.A. Nixon, "An Approach to Dealing with Performance Requirements during Information System Development," *Proc. Doctoral Consortion, Third IEEE Int'l Symp. Requirements Eng. RE '97,* pp. 73–88, Jan. 1997.
[50] B.A. Nixon, "Managing Performance Requirements for Information Systems," *Proc. First Int'l Workshop Software and Performance, WOSP '98,* pp. 131–144, Oct. 1998.
[51] A.L. Opdahl, "Performance Engineering during Information System Development," doctoral thesis and Technical Report 1992:5, Information Systems Group, Faculty of Computer Science and Electrical Eng., Norwegian Inst. of Technology, Univ. of Trondheim, Norway, 1992.
[52] A.L. Opdahl, "Requirements Engineering for Software Performance," *Proc. First Int'l Workshop on Requirements Eng. Foundation of Software Quality, REFSQ '94,* pp. 16–32, June 1994.
[53] J. Peckham and F. Maryanski, "Semantic Data Models," *Computing Surveys,* vol. 20, no. 3, pp. 153–189, Sept. 1988.
[54] S. Rozen, "Automating Physical Database Design: An Extensible Approach," doctoral dissertation, Dept. of Computer Science, New York Univ., 1993.
[55] J.W. Schmidt and F. Matthes, "The DBPL Project: Advances in Modular Database Programming," *Information Systems,* vol. 19, no. 2, pp. 121–140, 1994.
[56] H.A. Simon, *The Sciences of the Artificial,* second ed. Cambridge, Mass.: MIT Press, 1981.
[57] C.U. Smith, "Independent General Principles for Constructing Responsive Software Systems," *ACM Trans. Computer Systems,* vol. 4, no. 1, pp. 1–31, Feb. 1986.
[58] C.U. Smith, *Performance Engineering of Software Systems.* Reading, Mass.: Addison-Wesley, 1990.
[59] J.M. Smith and D.C.P. Smith, "Database Abstractions: Aggregation and Generalization," *ACM Trans. Database Systems,* vol. 2, no. 2, pp. 105–133, June 1977.
[60] Software AG of Canada, "University of Toronto Student Records System: High Level Business Requirements Study," Apr. 1996 (unpublished).
[61] University of Toronto, Faculty of Arts and Science, "1998–1999 Registration Handbook & Timetable," 1998.
[62] University of Toronto, Faculty of Arts and Science, St. George Campus, "Registration Handbook & Timetable, 1999–2000 Fall/Winter Session," 1999.
[63] E.S.K. Yu, "Modelling Strategic Relationships for Process Reengineering," doctoral thesis, Dept. of Computer Science, Univ. of Toronto, Dec. 1994.
[64] P. Zave, "Classification of Research Efforts in Requirements Engineering," *Computing Surveys,* vol. 29, no. 4, pp. 315–321, 1997.
[65] M.D. Zisman, "Use of Production Systems for Modeling Concurrent Processes," *Pattern-Directed Inference Systems,* D.A. Waterman and F. Hayes-Roth, eds., pp. 53–68, New York: Academic Press, 1978.

**Brian A. Nixon** received the BSc, MSc, and PhD degrees, all from the University of Toronto, Canada, the latter being earned in 1997. His doctoral thesis deals with performance requirements for information systems. As a postdoctoral fellow at the same university, he conducted the case study reported within this paper and edited and coauthored a book on nonfunctional requirements. His research interests include information system development (requirements, design, implementation, performance), requirements engineering, software quality, semantic data models, and applications of artificial intelligence to databases, software engineering, and programming languages.