Abstraction and Context in Requirements Engineering: A Synthesis of Goal Refinement and Ethnography

Colin Potts and Idris Hsi
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280, USA
{potts, idris}@cc.gatech.edu

Abstract: Most requirements engineering (RE) research and practice embodies a philosophy that we will call *abstractionism*, which involves the building of simplified models of domains of discourse and proposed systems. Abstractionists make much use of formal models, such as goal dependency networks. An alternative design philosophy is *contextualism*, according to which the peculiarities of the context of use of a system must be understood in detail before the requirements can be derived. Contextualists use qualitative methods to uncover and help interpret these particularities. In this paper, we analyze what it would mean to combine the best features of abstractionism and contextualism, and we ground our discussion in an illustration of abstractionist and contextualist thinking about RE through goal refinement (GR). In the context of the domain of meeting scheduling, we contrast a wholly abstractionist approach to GR with one that incorporates data gathered using two ethnographic methods. In doing so, we consider each step of the abstractionist approach, illustrating where ethnographic data obtained in our work environment affects the model produced. As we proceed, we summarize the general lessons learned. We then discuss how other abstractionist and contextualist methods could be integrated.

1 Introduction

Most RE research and practice embodies a philosophy that we call *abstractionism*. "Abstractionists" hold that RE involves the building of simplified models of domains of discourse and proposed systems. For the abstractionist, the main issues in RE practice are the consistency of these models and their faithfulness in their essential details to the reality being modeled. An alternative design philosophy to abstractionism is what we call *contextualism*. "Contextualists" hold that the richness of a system's contexts of use requires immersive exposure, not abstract modeling. Contextualists take observational and qualitative research more seriously, and tend to produce less formal descriptions than the graphical models and definitions of the abstractionists.

The distinction between abstractionism and contextualism echoes distinctions previously made for analyzing existing systems, such as that introduced by Kling [1980] between what he calls "systems rationalism" and "segmented institutionalism." According to Kling, designers and researchers working in the rationalist tradition act as though a system has a well-defined purpose; namely, to fulfill the rational and consensual goals of the host organization. In contrast, those working in the tradition of segmented institutionalism recognize that introducing an information system into an organization can catalyze and reflect processes of conflict and negotiation among the subgroups of the organization. A similar is made by Morgan and Smircic [1980], who classified paradigms for investigating organizations according to their epistemological assumptions. At one extreme, lies objectivist approaches to social science, which corresponds roughly to Kling's systems rationalism and our abstractionism. At the other lies subjectivist approaches, which are closer to our contextualism. Morgan extends this treatment into a number of disjoint paradigms for organizational studies in his book "Images of Organization" [Morgan, 1986] in which he contrasts the mechanistic and organismic metaphors for organizations with paradigms that are more informed by social science observation. Morgan's analyses were intended as views of organizations of people, but they can also be applied to the purposes of automation and software support. Whereas the RE practitioner or theorist typically adopts the metaphor that a system is a virtual machine [e.g., Jackson 1995]) or an artificial goal-seeking organism, Morgan argues that such metaphors encourage analysis of systems only within normative environments, falsely simplifying their richness and can lead to ill-considered interventions or designs.

Contextualism can also be found in cognitive engineering and human-machine systems research. Norman [1992] describes how pilots opportunistically employ commonplace artifacts, such as coffee cups and pens, as external mnemonic devices. Hutchins [1995], using a scenario that took place aboard a Navy vessel, performed a cultural anthropological analysis that reveals that the underlying complexity of the interaction between co-workers is deeply grounded in their environment. Human-machine systems, characterized as complex, multi-agent, technological systems, have a large contextual component to them that demand an ecological approach [Flach 1993]. Ecological task analysis and interface design [Vicente and Rasmussen 1990, Kirlik 1993] are two of the methods used in that field to examine and model the various aspects of the environment and their impact on system design.

In this paper, we analyze what it would mean to combine the best features of abstractionism and contextualism, and we ground our discussion in an illustration of abstractionist and contextualist thinking about RE through goal refinement (GR). The path we will

Published in Annals of Software Engineering, Vol. 3 (1997), pp. 23-61

take is as follows. First, in section 2, we define abstractionism and contextualism in more detail, give examples of abstractionist and contextualist RE methods, consider why a synthesis of the two philosophies is desirable, and discuss what research strategies we can adopt to achieve such a synthesis. Then in section 3, we summarize the main ideas behind GR, explain in what sense it is an abstractionist method and why it needs contextualist input, and outline a particular GR method. The main body of our argument follows in section 4. It takes the form, as all contextualist contributions must, of an extended example of an RE problem: the analysis of work practices involved in scheduling meetings, and design of an interactive meeting scheduler application. In Section 4, we contrast a wholly abstractionist approach to GR with one that incorporates data gathered using two naturalistic inquiry [Lincoln & Guba 1985] methods that are typical of the contextualist approach. Finally in Section 5, we discuss our experience in synthesizing GR with the two naturalistic inquiry methods, suggest how this experience could be transferred to other abstractionist methods.

2 Abstractionism and Contextualism

Many diverse methods can be classified as abstractionist or contextualist. In this section, we describe the two distinguishing features of the two design philosophies.

2.1 Abstractionism

RE methods can be broadly (though not exhaustively) divided into two major families: the process-dominated and object-dominated. Examples of process-dominated abstractionism are Structured Analysis (SA) [DeMarco 1978], SADT [Ross 1977], SREM [Alford 1985] and process-oriented formal specifications, such as CSP [Moore 1990]. In the process-dominated form of abstractionism, the system is viewed primarily as a machine that converts inputs into outputs. A system or context is modeled as a set of interacting processes or activities, frequently hierarchically arranged.

Typical examples of the object-dominated form of abstractionism are entity-relationship modeling [Chen 1976; Barker 1990] and object-oriented analysis OOA [Coad and Yourdon 1991; Coleman *et al*. 1994; Jacobson *et al*. 1992; Rumbaugh *et al*. 1991]. In the object-dominated form of abstractionism, the system is analyzed in terms of its principal objects, entities, or data abstractions and is quite literally an executable model of reality. Such diverse formal specification approaches as the model-oriented [Coleman *et al.* 1994; Jones 1990; Spivey 1988] and algebraic [Guttag *et al.* 1985] are also typical of the object-dominated family of methods, because they emphasize the description of domain objects or data structures and the constraints among them.

Although the process-dominated and object-dominated families have different ontologies, some methods (e.g., SSADM) [Meldrum *et al.* 1993] seek to integrate the process-dominated and object-dominated families by providing sub-languages for both and some rules for putting the sub-languages together.

A third form of abstractionism, teleological abstractionism, has recently arisen in the RE research community. In this form of abstractionism, a system is viewed primarily as a means for achieving goals. Several recent papers on goal refinement (GR) [Anton 1996; Anton *et al.* 1994; Dardenne *et al.* 1993; Green 1994; Potts 1995; Yu and Mylopoulos 1994] explain how a requirements specification can be treated as the systematic elaboration of goals into operational requirements that are allocated to software components. In teleological abstractionism, the system is modeled as a network of interdependent goals and the actions that the system or its environment need to perform to achieve these goals.

On the face of it, SADT, algebraic specification, and GR are very different species of abstractionism. However, abstractionist methods have many common properties. We identify five of these: the role of formal description, the design criteria that receive most attention, the putative origin of the requirements that they model, the role of the users in the RE process, and the community of practice.

2.1.1 Role of description

Abstractionists see designing as a technical activity that develops descriptions (e.g., Jackson [1995]) or models of the planned system and the reality into which the system will be introduced. Abstractionist methods stem from the need to model faithfully some aspects of the user's task and the proposed system.

2.1.2 Design criteria

Abstractionists judge the quality of designs largely against functional criteria: externally, a design should give rise to implementations free of functional defects; internally, it should have a modular structure that promotes future modifications and reuse of existing components.

2.1.3 Origin of requirements

In contrast to the richness of practical guidelines contained in abstractionist methods for modeling and checking information about a proposed system, the guidelines for acquiring requirements and obtaining model fragments from those requirements are either nonexistent or very weak.

2.1.4 Role of users

Abstractionist approaches tend to emphasize the role of the customer in requirements acquisition. The principal sources of requirements are management interview and strategy documents. Observation of the work environment, and other ecologically motivated techniques that describe the environment in user-oriented terms, are used only to the extent that the knowledge obtained is likely to lead to a system design that meets the customer's requirements.

This one-sidedness has been criticized for a variety of reasons. It embodies a mechanistic view of organizations and the authority structures within them, a view that is not universally held by organizational theorists [Morgan 1986; Morgan and Smircich 1980]. Furthermore, it is a view that is culturally constrained and does not apply in countries or cultures with legal and political traditions that demand user involvement in the process of technological change (e.g., the widespread trades unionism among office workers in Scandinavia [Ehn 1988]). The descriptions obtained from managers are typically normative. They describe what should be done within the organization, rather than what might have to be done at a concrete level to meet the goals of management.

2.1.5 Community of practice

Abstractionism has emerged out of the RE, and more generally the software engineering tradition to system development. Model and generalizations are seen as strong mathematical, scientific and engineering tools to be constructed and manipulated in the solution of the problems.

2.2 Contextualism

The range of pragmatic techniques and theoretical positions that characterize contextualism, have largely emerged in response to abstractionism and differ in the degree to which they reject its central theses. Some "soft" methods, although still producing abstract models, aim to produce better ones by incorporating human task data. For example, ecological task analysis [Kirlik 1993] rests on the idea that user tasks must be modeled in light of the real, frequently collaborative, activities that the users perform and the environmental constraints and affordances provided by the workplace situation and artifacts. User interviews, videotaped observations, activity tracking, and analysis of workplace artifacts, also help the designer obtain empirical information about user behavior [Nielsen 1993]. The result, however, is still a task model that abstracts away from the details of the context.

Other methods fall further along the continuum of contextualism. For example, the Soft Systems Methodology (SSM) [Checkland 1972, 1981; Checkland and Scholes 1990], developed as a general-purpose management planning method, leads to a mixture of problem situation descriptions. The resulting conceptual model is a loosely defined flow model that interconnects to the major sub-problems or transformations that comprise the altered core definition. Thus, SSM could be used potentially as an introductory phase for SA, in which the conceptual model is more exact. The Client-Led Design (CLD) method [Stowell and West 1994], a development of SSM specifically for information systems, incorporates data flow diagrams of SA in this way. There have also been attempts to use SSM as an introductory phase for object-dominated abstractionist methods, including Z specifications [Bustard and Lundy 1995] and OOA [Bustard and Dobbin 1996].

In ecological task analysis and SSM, the users and management, respectively, are sources of data for the analysis process, and the human factors expert or management consultant is removed from, or elevated above, direct involvement in the use context. Such distance is characteristic of abstractionism, because it helps the analyst "see the forest for the trees". An alternative approach, however, is for the users or stakeholders themselves to be directly involved in the RE and design processes under the guidance of a team of designers. In these Participatory Design (PD) methods, there is less distance between user and developer. PD is more contextualist than SSM for two reasons. First the mutual involvement of users and designers in the design process means that there is less need for communication between the parties through the medium of specifications and models. PD projects tend to make far greater use of informal descriptions and mock-ups. A second source of contextualism in PD is that PD practitioners tend to value the idiosyncratic properties of the workplace and the legitimacy of users' rights to define and customize their work practices. This is why the users are involved as co-designers in the first place.

An alternative to involving the users in designing is for the design team to immerse itself in the work context so that its members develop an improved understanding of the implications of any design decisions that they make. In this approach, naturalistic social methods, and especially the ethnographic methods of structured interviewing and participant observation are introduced into RE. There have been several promising pilot projects in which ethnographers, working alongside designers trained in computer science, have analyzed workplace situations and made different recommendations about proposed systems than the computer scientists could have made without ethnographic data [Luff *et al.* 1994; Randall *et al.* 1994; Sommerville *et al.* 1993]. In particular, the training of ethnographers to look for the roles played by physical artifacts, spatial layout, and patterns of informal, or "off-task" communications in cultural contexts complements the training of computer scientists to abstract away from these phenomena to get at the "essential" characteristics of the problem.

Ethnography and PD are complimentary approaches, not mutually incompatible alternatives. Some practitioners have codified contextualists RE and design methods that incorporate elements of both, the best known being Contextual Inquiry [Holtzblatt and

Jones 1993], in which requirements emerge from an analysis of a series of brief ethnographic interviews conducted while the user works.

Whereas in the case of abstractionism we have a collection of apparently different methods that are unified by deeply buried and usually tacit assumptions, in the case of contextualism, we have the opposite: a variety of methods that the computer scientist might think are obviously similar, because they incorporate "soft" factors into RE, but which differ in significant ways. The family resemblances among contextual methods are as follows.

2.2.1 Role of description

Contextualists, like abstractionists, construct descriptions. Just how abstract and generalizable these descriptions can be taken to be depends on the form of contextual method in use and the assumptions of the investigator. Many ethnographers repudiate the value of abstract models in favor of "thick descriptions" [Geertz 1973] of the studied situation. SSM practitioners make use of "rich pictures". Designers using ecological or PD methods produce descriptions that illustrate concrete work situations or possible scenarios of system use more than general properties of the proposed system and its environment. Such descriptions usually involve a combination of natural text, informal sketches, and even cardboard mockups [Ehn 1988; Ehn and Kyng 1991]. *The key discriminator is that particularities are as important to understand as generalities.*

2.2.2 Design criteria

Contextualists seek to maximize the usefulness, convenience, task-fit and usability of the resulting system. Design integrity and performance, which are the primary goals of the abstractionist, are only valuable to the extent that they contribute to these ends. "Requirements" are not prospectively derived statements of need, produced before design starts and driving its direction once underway, but instead are statements expressing the important design criteria throughout the system development process. For example, PD practitioners seldom mention requirements, still less engineering, regarding usefulness and contextual fit as design criteria. *The key discriminator is that a system must fit smoothly into its context of use, not vice versa.*

2.2.3 Origin of requirements

In abstractionism, the requirements are abstracted from the stated needs of management and knowledge elicited from domain experts. Requirements may reflect prescriptions and strategic policies that go beyond or even totally replace current work practices. In contextualism, the requirements are derived from the work contexts at hand. In abstractionism, the designer elicits requirements from the customer or is told the requirements. In contextualism, on the other hand, the designer is shown exactly what happens, and the requirements emerge from the resulting understanding of the detailed context. *The key unifying concept here is that current practice, not strategic prescriptions, determine the requirements for a new system.*

2.2.4 Role of users

Users are paramount in the contextualist tradition. But the word "user" betrays a bias toward computer-based implementation and a designer's perspective. We do not think of carpenters as "users" of saws, but as practitioners of a skill for which a saw may be a useful tool; analogously, researchers in the contextualist tradition may view a user of a system not as a "user" but as a "member" of a situation or community of practice, or as a "worker" in an organization. It is these members of workers who are the main sources of information about the system requirements. *The key discriminator here is that it is more valuable to pay attention to the tasks and interactive activities of individuals in the context and their assignment and use of resources than to the prescriptions of management.*

2.2.5 Community of practice

The contextualist philosophy toward RE has evolved most notably in the fields of human-computer interaction (HCI), computer-supported cooperative work (CSCW), and organizational computing, although as explained in the previous sections, the originating ideas come from the social sciences. Until the special issue of *Communications of the ACM* (May, 1995) devoted to the human factor in requirements, most people working in the contextualist tradition seldom even used the word "requirement," preferring to concentrate on the design criteria of usefulness and contextual fit rather than on the documentation of requirements that are intended to lead to the achievement of these criteria. *The key discriminator here is that many contextualists think of themselves as facilitating change, not as engineering solutions to problems.*

Table 1 – Key discriminators between abstractionism and contextualism

	Abstractionism	Contextualism
Role of description	Abstractions are powerful and general	Particularities are as informative as generalities
Design criteria	Design integrity	Contextual fit
Origin of requirements	Prescriptive recommendations	Current practice
Role of users	Management	End-users
Community of practice	RE and software engineering	CSCW and HCI

2.3 Integration of philosophies

These key discriminators are summarized in Table 1. Our position is that abstractionism and contextualism have complementary advantages and limitations and that a synthesis or integration of the two philosophies is needed. Abstractionist RE methods cannot be used effectively without some input about the real contexts in which systems are to be used; nor can contextualism work in isolation, because the resulting system has to be developed at some technical level using the architectural and programming abstractions of a computer science. Let us then consider these complementary advantages and limitations in a bit more detail, and then review how a synthesis of the two design philosophies could be achieved.

2.3.1 Advantages and limitations of abstractionism

The advantages of abstractionism are well known. Many systems are developed for use in multiple use contexts, and so some degree of abstraction away from particular contexts is inevitable. Abstractionist methods provide strategies and guidelines for choosing the right abstractions for communicating with stakeholders and for initiating the implementation process. Sometimes, the physical structure of the use context is used as a starting point for abstraction. For example, some flavors of SA start with the physical modeling of the enterprise, especially by focusing on the form and context of forms and reports [McMenamin and Palmer 1984]. Only later are the essential properties of these artifacts abstracted from the physical model into an "essential model". And in OOA, it is possible to take the physical work environment as a rough guide for identifying significant object classes [Coad and Yourdon 1991].

However, abstractionist methods can go too far in encouraging the practitioner to simplify away the complexities of the real-world context under investigation. Few abstractionist RE methods include any explicit examination of exception handling. Exceptions, in these methods, are unwelcome complications that are added on later. In classical RE, "exceptions" refer to situations in which a physical entity in the system's environment or some component of the system itself breaks down and generates incorrect or untimely data. Thus, exception handling, is often relegated to the technical specialty of fault tolerance, rather than being elevated to central importance in RE. Taking a broader definition of exceptions that would also accommodate non-normative or customized behavior by users shows why this is important. If the context of the system is assumed to be a collection of modeled processes, deviation by a user from a normative process sequence must either be prohibited or will cause the system to function ineffectively. Sometimes, it may be appropriate to constrain user behavior within narrow boundaries, but as Suchman [1983] has shown, the effective performance of even apparently routine clerical tasks necessarily involves much more creativity and discretion on behalf of the clerical worker than a normative account provided by management would usually contain. Special cases, exceptions and unforeseen problems abound in organizational reality. A purely abstractionist perspective, then, often fails to account for the nuances of the user's work practices, thereby leading to systems that lack flexibility.

2.3.2 Advantages and limitations of contextualism

Contextualism's advantages stem from the ability and willingness of contextualists to accommodate the richness and complexity of the real world. However, we may wish to simplify and abstract. Real-world contexts are always full of subtle, concrete details that affect the success of any proposed changes, including information systems. Contextualism's advantages are mainly abstractionism's limitations: by concentrating on concrete and idiosyncratic details, we gain a much better insight into the specific requirements of real users.

A purely contextualist approach, however, harbors problems. It is one thing for software designers to engage in PD practices and build cardboard mockups of equipment or to do participant observation within a work environment. It is quite another thing to turn the insights gained from such activities into design commitments. Software designers cannot easily take a cardboard mockup or thick description of an organizational culture and turn either into a piece of software that works effectively in the investigated context unless they can bridge two gulfs: that between contextual terms and software terms, and that between what exists and what is desired. This is not so much a matter of going from the "soft" to the "hard". On either side of the gulfs are to be found rigorous techniques. (Even ethnography has a large literature on methodological rigor [Bernard and Russell 1994, Lincoln and Guba 1985], even if software engineers might regard it as a "soft" discipline.) Rather, it is a matter of going from one form of rigor to another through the intermediate medium of informal and non-rigorous insight. The danger with contextualism is that a well-intentioned investigation of

context will be lost or only have insignificant effects on the design process, because it cannot be translated into requirements for change. Ethnographers and software engineers working on a design team can have difficulty communicating [Sommerville *et al*. 1993]. Ethnographic descriptions do not naturally flow into prescriptions for change.

Another potential problem with contextualism is that it focuses attention so exclusively on the current situation that it makes it difficult to consider radically different ways of doing things. Similarly, contextualism tends to concentrate on immediate actors in the context (future end users or consumers of the system's outputs). But there may be other stakeholders (e.g., regulatory agencies) that a contextualist approach tends to ignore.

2.3.3 Synthesizing abstractionist and contextualist methods

The advantages and limitations of abstractionism and contextualism are summarized in Table 2. What we would like to see is an integration of abstractionism and contextualism in RE that goes beyond a mere call for contextual awareness and which helps substantively in the transitions from contextual terms into software terms and from statements about the context into statements of requirements.

Table 2 - Advantages and limitations of the abstractionist and contextualist perspectives

	Abstractionism	Contextualism
Advantages	Generalization across contexts Standard methods for constructing abstractions	Accommodation of richness of contexts
Limitations	Oversimplification Overemphasis of normative cases	Descriptive, not prescriptive Overemphasis on immediate actors and current practice

What would an integrated approach be like? There are two aspects to this question: the conceptual and the chronological. First we must ask how the ideas of abstractionist and contextualist thinking can work together in RE; then we must ask when, in a design process, contextualist and abstractionist thinking would dominate.

Taking the conceptual question first, there are three plausible approaches to integrating the abstractionist and contextualist perspectives:

- Use a single existing abstractionist method as the core concept, and enhance it with contextual methods.
- Use a collection of contextual methods as the core methods, and make them more systematic by integrating them with abstractionist techniques.
- Start afresh, and invent a wholly new approach.

We have adopted the first approach because most abstractionist methods provide a prescriptive strategy into which contextualist inquiry and analysis methods could be introduced (although they seldom are). This is also the alternative most likely to give contextualism a voice in an engineering-dominated development organization.

It would be more difficult to start with contextualist methods, because they provide less an overall strategy than a motivating user-centered philosophy (which is laudable, but often too vague), and a collection of empirical methods (which are essential, but too specific and diffuse to act as an organizing framework for the RE process).

We reject the third alternative as being impractical. There are too many methods already to need a new one.

As for the second, chronological question, there are two realistic options:

- Start by performing a contextual investigation and then make the transition to abstractionist modeling.
- Perform contextualist and abstractionist activities in parallel throughout the RE process.

The first alternative has a long history. It echoes roughly the traditional distinction between customer-oriented requirements gathering and developer-oriented specification, and could easily deteriorate into that mode of doing RE. The problem here is that it seems that contextual information is more important early on and can be sidelined later, once the serious business of specification gets underway. There is also a parallel between this alternative and the exploratory use of qualitative methods in social science research prior to controlled experiments or surveys. And, as we have seen, in Section 2.2, the suggested integrations of SSM with Z [Bustard and Lundy 1995] and OOA [Bustard and Dobbin 1996] take the form of doing SSM first and then using the results of applying that method to producing a specification or object model.

We have chosen the second alternative, though, for several reasons. First our experience has been that not only does an investigation of the work context inform the development of models, but that tentative modeling commitments feed back strongly into the gathering of contextual information. Thus, neither activity correctly can be said to precede the other. Second, there is a serious danger in doing most of the contextual investigation before modeling starts. Contextual information may then be seen merely as "background" information about the current way of doing things, and that the requirements for the proposed system emerge from

abstract thinking alone. On the contrary, contextual information can act as a powerful brake to some idealized automaton suggestions, and must therefore be considered fully throughout the RE process.

Integrating two fundamentally different perspectives is an ambitious, long-term goal constituting the establishment of a tradition in its own right. It will require the development and evaluation of many systems in many contexts over many years, combining diverse abstractionist and contextualist methods. This paper is a first step in that direction and in no way intended as a thorough empirical test of the approach that we have just described. *Rather, it provides a proof of concept*.

What we have done so far is explain abstractionism and contextualism, show they have complementary strengths and weaknesses, and discuss what it could mean to integrate them. In the next two sections, we provide our proof of concept by introducing specific abstractionist and contextualist methods and investigating their integration in an illustrative case study.

3 Goal Refinement

Several authors working in the abstractionist tradition have recently been exploring GR methods that start with enterprise goals and explicitly refine these goals into requirements [Potts 1995, Dardenne et al., 1993]. By starting from goals, rather than processes or objects as in the process-dominated and object-dominated types of abstractionism, this teleological form of abstractionism emphasizes the origin of requirements in goals. Successful systems fulfill goals; unsuccessful systems fail to fulfill them.

In this section, we describe one such GR method in enough detail to illustrate its application. Our objective is not to review or evaluate GR methods in general, nor to claim that this GR method is better than its alternatives. (For a review of GR methods, see [Green 1994].) Our purpose in using GR is to illustrate abstraction rather than a more established object-dominated or process-dominated method (such as OMT [Rumbaugh *et al.* 1991] or structured Analysis [Ross 1977] is that teleological abstractionism illustrates abstractionism in its purest form, because it starts from the premise that a system is designed to meet rational and describable objectives. After describing the method, we explain why the teleological abstractionism exemplified by this method requires contextual input as much as the other kinds of abstractionism.

3.1 Goal-refinement method

The form of GR that we illustrate in Section 3 follows the following steps roughly in order, but with inevitable backtracking and rework:

- Analyzing goals, actors, and objects.
- Transforming goals into operational requirements.
- Identifying and handling obstacles.

Next, using the language and philosophy of abstractionism, we shall say more about the constructs identified and analyzed during each of these steps. In Section 3.2, we shall return to the relevance of these constructs to contexts of use.

3.1.1 Goals, agents and objects

An *enterprise* is the collection of processes or real-world activities that the system supports. The *required system* (RS) is the computer-based subsystem to be inserted into the enterprise to make some of its goals easier to achieve or more affordable. By *goals*, we mean states of affairs that the enterprise strives to achieve, maintain, or present. An *object* is some element of the enterprise or the information it manipulates. Goals usually refer to the states of objects. An *actor* is a component of the enterprise that is responsible for achieving goals. An actor could be a user, a group of users, an organization, a subsystem of the RS, or the RS of the whole. Actors are objects, but not all objects are actors. An *action* is anything done by an actor to change the states of objects. Goals are achieved by actors performing coordinated sets of actions.

Goals are related in several ways. Goals may support other goals. For example, a *goal expansion*, shows how a single goal is accomplished by achieving some subgoals. Or the achievement of a goal may have to *precede* the achievement of another. Conversely, one goal may *thwart* another; that is, achieving the first goal may raise an obstacle to achieving the second.

3.1.2 Realization and allocation

Allocation establishes the boundary between the environment and the RS. That is, it reflects the degree and type of automation proposed. When a goal is *realized*, it is replaced by state-changing actions attributed to actors. The actors may be in the environment (e.g., users) or the RS.

For example, one of the subgoals of a collaborative meeting scheduler could be to ensure that all invitees are aware of the request for the meeting. This could be achieved in several ways. The person calling the meeting might inspect an on-line shared calendar maintained by the system and then call each person (perhaps at telephone numbers supplied by the system) to advise them that a meeting is planned. Alternatively, the calendar could act as an agent for the person calling the meeting and would send a predefined message to each invitee automatically. Thus, a single goal could be assigned to the system and its users in many ways (i.e., degrees of automation), and the detailed actions required of the system and its users could be entirely different for different design scenarios.

Choices about goal realization and the allocation of responsibilities to roles raise many non-technical organizational issues, such as the likely effects of RS on power relationships and influence. We ignore those here. Which solution is best for a community of users depends on their responsibilities and ownership of information in the context.

3.1.3 Obstacles and defenses

An *obstacle* is a situation or event that could block the achievement of a goal. One of the principal objectives of our GR method is the identification of obstacles and the definition of *defenses* and *mitigation strategies*. Considering obstacles forces the designer to investigate situations in which the working environment does not meet idealized assumptions, and therefore to think about robust and flexible design solutions. Abstractionist thinking naturally emphasizes expected or normative behavior, and it idealizes and simplifies the likely behavior of users. But interactive systems, such as meeting schedulers, must cope with many types of obstacles: e-mail goes astray; people forget to enter information on calendars; they ignore requests for appointments. Obstacles that occur often in one environment might not arise in another; and those that are most obstructive to goal fulfillment in one environment could be ignored in another. Thus, the identification of obstacles, and the design of practical solutions, depends critically on a knowledge of the work environment for which the system is being designed.

Goals are elaborated by asking the following questions about each goal:

- *Failure occurrence*. Can this goal be obstructed, and if so, when?
- *Failure consequence*. If this goal can be obstructed, what are the consequences?

Deciding which obstacles to deal with is outside the scope of this paper. We assume that the process is largely informal, depending on discussion and scenario exploration [Potts 1995], but in the case of some questions it could involve statistical analysis and safety techniques such as fault-tree analysis.

Two types of elaboration deal with obstacles: adding *defensive* and *mitigation* goals. (These terms are preferable to *prevention* and *recovery*, because obstacles cannot always be wholly overcome.) If an obstacle is not worth defending against, it may be left as an operational risk. The discussion activity leading to these forms of commitment consists of the repeated asking and answering of these two questions:

- *Defense*. If the consequences of the goal being obstructed are significant or the risk of goal obstruction is high, how can the obstacle be defended against?
- *Mitigation*. If the consequences of the goal being obstructed are significant or the risk of goal obstruction is high, how can the obstacle's consequences be mitigated?

3.2 Illustration: meeting scheduling

As an example of a human activity system involving automation, we consider a collaborative meeting scheduler. Van Lamsweerde *et al.* [1995] have documented the requirements for such a system, and we have used their requirements as a guide in constructing the example. (In addition, [Green 1994] reviews several other goal refinement approaches using the meeting scheduler as a common example.) A more detailed treatment of this example has been published elsewhere [Potts 1995].

The scheduler is to help an organization arrange meetings. We assume that two meetings cannot occur in the same place at the same time and a person can only be in one place at a time.

3.2.1 Goals, actors, and objects in the meeting scheduler

Figure 1 shows in outline form the goal hierarchy produced by Van Lamsweerde *et al.* [1995]. There are four actors: two user roles and two software components. The user roles are the initiator (whoever calls the meeting), and the participants (on whose schedules the time of the meeting depends). The two software components are the Scheduler (which is responsible for managing schedules and preferences, and choosing the meeting time and place), and the EMS (which delivers messages to the appropriate people).

In what follows, we shall concentrate exclusively on the lowest-level goals (LLGs) in the hierarchy. Dependencies and obligations are not shown in the figure, but each LLG depends on the achievement of its immediate predecessor LLG (with the exception of the two OR branches), and each LLG obliges the achievement of its immediate successor. No goals thwart other goals.

Meeting Request Satisfied
- Scheduler Available
  - **Initiator: Invoke Scheduler**
  - *Initiator forgets how to invoke*
  - *Scheduler fails*
- Participants' Constraints Known
  - Participants' Constraints Requested
    - **Initiator: Specify Participants**
    - **Initiator: Specify Meeting Details**
    - **Scheduler: Compose Invitation**
    - **EMS: Deliver Invitation**
    - *Wrong people invited*
    - *Wrong meeting details given*
    - *Invitation not delivered*
  - Participants' Constraints Provided
    - **Participant: Compose Constraints Message**
    - **EMS: Deliver Constraints Message**
    - *Participant Ignores Constraints Request*
    - *Constraints Message not Delivered*
    - *Participants' Constraints Confused*
- Meeting Planned
  - Meeting Planned without Negotiation
    - **Scheduler: Determine Best Times and Places**
    - **Scheduler: Determine Best Time and Place**
    - *No Feasible Schedule*
    - *Participant Reschedules*
    - *Rooms Confused*
  - OR
  - Meeting Planned with Negotiation
    - Participants' Preferences Known
      - **Scheduler: Determine Best Times and Places**
      - **Scheduler: Compose Preferences Request**
      - **EMS: Deliver Preferences Request**
      - **Participant: Compose Preferences Message**
      - **EMS: Deliver Preferences Message**
      - *Preferences Request Not Delivered*
      - *Participant Ignores Preferences Request*
      - *Preferences Message Not Delivered*
      - *Participants' Preferences Confused*
      - *Rooms Confused*
    - Final Approval Asked
      - **Scheduler: Best Time and Places**
      - **Scheduler: Compose Approval Request**
      - **EMS: Deliver Approval Request**
      - **Initiator: Compose Approval Message**
      - *No Unanimously Preferred Time*
      - *Approval Message Not Delivered*
      - *Initiator Ignores Approval Request*
- Participants Notified
  - **Scheduler: Compose Notification**
  - **EMS: Deliver Notification**
  - *Notification Not Delivered*
  - *Participant Ignores Notification*

Figure 1. Goal hierarchy for the meeting scheduler (e-mail version). Below each LLG, an episode consisting of actions (in **bold**) is subsumed. Each action is given as: *Actor name: Action name. EMS* = electronic mail system. Below the actions, the obstacles are listed (in *italics*) that can block the LLG. Goal analysis is reproduced from [Van Lamsweerde *et al*. [1995] with permission of the authors.

3.2.2 Realization and allocation in the meeting scheduler

Figure 1 also shows in bold a set of actions for a system based on e-mail communication. These actions are all allocated to one of the four actors. Thus, each goal is realized by the actions that appear below it. The set of realization decisions shown in Fig. 1

reflects a meeting scheduler based on an e-mail metaphor. Other realizations are possible, such as a shared calendar (see [Potts 1995] for a comparison of the two systems).

3.2.3 Obstacles and defenses in the meeting scheduler

We can identify a number of obstacles for each LLG. These are shown in italics in Fig. 1. Each goal can be blocked by the obstacles shown beneath it. For example, the goal Participants' Constraints Provided can be blocked by the obstacle *Constraints Message not Delivered*.

Note that the obstacles include obvious hardware failures or software bugs (such as *Scheduler fails*), user mistakes or failures to follow manual responsibilities (such as *Participant Ignores Constraints Request*), and environmental contingencies (such as *No Feasible Schedule*). These may be handled differently during implementation, and therefore not all qualify as exceptions in the programming language sense of the term, but they are very similar at the teleological level and are therefore treated equivalently.

A possible defensive action to avoid the two obstacles `Wrong People Invited` and `Wrong Meeting Details Given` is for the Scheduler to request confirmation about the meeting details (including the desired participants) from the Initiator before sending the information. Like all confirmation actions, this is defensive rather than strictly preventative, because there is nothing to prevent the Initiator making a further mistake when reading the confirmation request. The following action, specified in an informal model-based style similar to FUSION [Coleman *et al.* 1994] addresses the obstacles:

```
action RequestDetailsConfirmation
    role:          Scheduler
    reads:         mtg
    changes:mtg
    assumes:       mtg.Participants decided
                   mtg.TargetInterval decided
    result:        mtg'.Participants confirmed
                   mtg'.TargetInterval confirmed
    sends:         DetailsConfirmationScreen to Initiator
end RequestDetailsConfirmation
```

where *TargetInterval* is the time interval during which the meeting is to be held and *decided* and *confirmed* are predicates that become true when the Initiator performs corresponding decision making actions. The action to send the Participants' Constraints Request initially has as its *assumes* condition *mtg.Participants decided*, but as a result of the elaboration described here, this condition becomes *mtg.Participants* confirmed.

A mitigation action to undo the effects of the `Wrong people invited` obstacle might be to "uninvite" them. This action can be defined as follows:

```
action UninviteParticipant
    role:          Scheduler
    reads:         mtg
    changes:mtg
    assumes:       mtg.Participants confirmed
                   p in mtg.Participants
    result:        mtg'.Participants = mtg.Participants - {p}
    sends:         uninvitation to p
end UninviteParticipant
```

3.3 Incorporation of contextual methods into goal refinement

The description of the GR method in Section 3.1 and the brief example given in section 3.2 are typical examples of abstractionism. Our treatment closely resembles in its philosophy and form other GR formulations [Van Lamsweerde *et al.* 1995; Green 1994]. Yet, it begs many important questions: How do we know which goals, actors or objects to model, and whose goals take priority? How do we allocate functionality to the RS or its users and why? And where does our knowledge of obstacles come from, and why should we choose any particular defense or mitigation option?

3.3.1 Goals, actors and objects in context

Perhaps the central question for teleological abstractionism is the epistemological question: do the goals really "exist," and, if so, whose are they? Goals may inhere in a system once it is implemented in the same way in which goals may inhere in the way an

organization functions, but these goals are really the goals of some stakeholders. Our treatment of the meeting scheduler concentrates on the problem of scheduling a meeting, but in what sense is that the main goal of a meeting scheduler? Perhaps the goal should be "effectively manage one's time" or (in the words of a Dilbert cartoon) "avoid all meetings with boring morons."

The actors *Scheduler* and *EMS* already presuppose a particular style of implementation. We could have chosen to study meeting scheduling as a problem involving only those people who call and attend meetings.

Although not highlighted in this example, objects play an important role in the GR method because it is their states that are the target conditions of the goals. For example, *Meeting Scheduled* (a goal) aims to change the status of *Meeting* (an object). We did not choose to classify meeting into subtypes in this working of the problem. Some commercial time management applications and meeting schedulers assume a normative classification of meeting and appointment types, and work differently for the different types. Why should we choose one set of objects over another?

It is clear that contextual factors may significantly affect the wisdom of these choices. At what level one pitches one's description of the goals, which actors one includes, and the details and presence in the teleological model of different types of objects all have important ramifications for the usefulness and contextual fit of the system.

3.3.2 Realization and allocation in context

The example describes meeting scheduling in terms of an e-mail metaphor, and a companion paper [Potts 1995] discusses both e-mail and calendar versions of a meeting scheduler. But why stop there? Why not have a meeting scheduler that acts as a kind of arbitration service when meetings cannot be scheduled to everyone's satisfaction? Why not have a system consisting of agents, or electronic assistants that intelligently look after their owner's interests like tenacious secretaries? GR does not recommend any specific operationalization or allocation decisions; it merely aids in the analysis of them once made.

These decisions also affect the contextual fit of the system, because the types of actions that are automated by a system, those that are supported, and those that are assumed to be performed by the users ("allocated" to user actors in the terminology of GR) may conflict with the natural ways of working that have evolved in a context.

3.3.3 Obstacles and defenses in context

Finally, it should be clear that the likelihood that the obstacles shown in Fig. 1 will occur in a given context depends greatly on the policies and habits of the people who work there. In some organizational cultures, it would be anathema not to respond to a request for one's attendance at a meeting; where we work, however, ignored e-mail messages are a daily fact of life. Thus, the types of goal breakdown that occur depend very much on the specific nature of the context, as will the appropriateness of the defensive or mitigation strategies employed by the system designer to make the system more robust in the presence of obstacles.

4 Integrating Goal Refinement and Ethnographic Methods: the Case of Meeting Scheduling

In this section, we stay with the domain of meeting scheduling, but now we look at some real data on how people schedule their time, and discuss the implications of this contextual data for the design process.

4.1 Meeting scheduling domain

Meeting scheduling affords a good example of how the nuances of a work environment strongly affect the feasibility of automating a task and how contextualist methods can be used in conjunction with an abstractionist perspective. Meeting schedulers are often used in the RE and software methodology community (e.g. see [Chandy & Misra 1988; Green 1994; Potts [1995]; Van Lamsweerde *et al*. [1995]), and our discussion of the problem in Section 3 exploited its universal familiarity. Meeting schedulers are also interesting in that they are packaged solutions to a common contextualized problem. Contextualism does not have to be restricted to custom-developed systems.

It is tempting to slide from the familiarity of the real practice of meeting scheduling into an unrealistic abstraction of the problem. In these abstractions, scheduling a meeting involves searching for the intersection of several well-defined schedules. These schedules consist of a sequence of time periods during which the prospective meeting participant is either free or occupied (and never in a state of ambivalence). People can be in only one place at a time, and so overlapping appointments are impossible. Some people are more important than others, and so their preferences receive higher priorities, which means that a system administrator will have to assign these priorities. Very soon, meeting scheduling starts to look like that other favorite laboratory preparation of the software engineering researcher, the lift or elevator scheduler. One finds oneself building an elegant theory about meetings, time intervals, priorities, and scheduling that bears little resemblance to the messy reality that most of us directly experience.

Among others, Grudin [1989] has used meeting schedulers as a canonical example of collaborative systems that are likely to fail precisely because abstractionist designers underestimate the complexity of the work environment. One of the reasons why

commercial meeting schedulers are becoming more widely and effectively used in some organizations is that the effective tools support discretionary work processes more fluidly than their predecessors [Grudin and Palen 1995].

Thus meeting scheduling is more than a toy example. Its familiarity is deceptive and hides great complexity, subtlety, and context-specificity.

4.2 Contextualist Methodology

We present data gathered on how people within our organization schedule meetings. Our methodology consists of two data gathering methods: structured workplace interviews, and the analysis of a corpus of email messages about meetings and appointments.

Interviewing the members of a context can verify the assumptions made in the initial stages of goal formulation or reveal new goals or organizational issues that will have an impact on system development. The interview excerpts in the next few sections are taken from taped conversations with three employees of an academic institution conducted while they worked. Such interviews are standard ethnographic practice [Spradley 1979] and form a major component of Contextual Inquiry [Holtzblatt and Jones 1993].

The employees' work involves the scheduling of meetings and other appointments. One schedules meetings and arranges travel for a single important individual. Another schedules meetings for both a single key individual and for the department. The third schedules meetings for groups of people as part of much broader responsibilities. Prior to the interviews, a series of questions were drafted based on the assumed goal structure of a meeting scheduler. They were left open ended to allow the informants to structure their own answers and to avoid biasing the discussion to simply verifying the existing goal structure. Then the interviews were transcribed and analyzed.

Because our informants told us that most meeting scheduling is done in this organization through the medium of e-mail, we also gathered a corpus of e-mail messages about scheduling. There were 343 such messages in the corpus. One of us (and occasionally both) was the sender or recipient of these messages.

Such a corpus of documents has the additional benefit that it lets us tease out some of the tacit knowledge that users bring to their actions. Interviews are not as effective in providing information about tacit knowledge.

The ethnographic interview and content analysis of documents are typical qualitative methods used by ethnographers and social science researchers. The literature on qualitative methods, e.g., [Lincoln and Guba 1985], abounds with discussions on how representative, typical or exceptional one's informants and data sources should be. If we were designing a meeting scheduler for a specific organization, it would be important for us to address these issues by interviewing more people and different types of people within the organization. It would also be important to gather email messages concerning other people's meetings.

Our purpose here is merely to show how typical data gathered using these methods can be used in conjunction with GR. The representativeness and richness of the data for the specific context is not critical for our purposes, as long as it illustrates the types of contextual data an extended contextualist investigation, such as a participant observation study, would generate. We believe that the data we discuss in the next few sections is typical of the type of contextual data that would be gathered in a longer study.

4.3 Goals, agents and objects

When our informants told us how they scheduled meetings and what they tried to achieve when they did so we discovered significant differences between what they said they did and the idealized goals that appear in the abstractionist treatment given in Section 2. The high-level goals of the users may not be obvious unless one gathers contextual data. Their goals are much more fluid and less exact than the abstractionist treatment would suggest, and the concrete form of their workplace artifacts are just as important to the effectiveness of scheduling as their abstract properties.

4.3.1 Overcoming teleological preconceptions

Meeting scheduling is not always the best way to bound the problem that these informants face, as they have to consider more general time management issues. Contextual data forces the designer to consider incorporating into the functionality of the system some goals that do not appear to belong there. In the following comment, one of our informant explains that part of her job is to optimize her principal's travel schedule. She is not merely scheduling a number of independent meetings, but has to take account their geographical locations.

> *Because he's trying to do several different things on one trip so he might be flying to two or three different cities like the trip he's on today, he's flying out to Denver for a meeting in Boulder but then he's going out to California to meet with other people so it's never just a direct round trip, he's always making three or four stops along the way.*

Here, then, are several goals that do not appear in the abstractionist account, but which have to be achieved in the context:

Optimize trip
- Know about trip locations
- Meetings in trip are scheduled
- Trip is coherent

Not all of these goals may be supported by the system. (*Meetings in trip are scheduled* will be achieved by the previously accepted *Meeting scheduled*.) Abstractionism encourages the simplified and abstract definition of problems; but in this case, artificially bounding the problem as one of "meeting scheduling" rather than "time management" or "trip organization" would preclude even considering whether the goals should be refined further.

The general lesson to be learned here is to use contextual data as a source of potential goals and as way to overcome preconceptions about what the goals should be. We recommend using contextual methods to produce a broader and divergent list of goals than an abstract definition of the problem would seem to require. After clustering the goals and showing the obvious dependencies, a decision must be made about which goals are within the scope of the project and which are not, and (in the case of contexts in which there are clear conflicts between the goals attributed to different stakeholders) who's goals are legitimate to pursue and who's are not. These are, of course, politically loaded decisions and will be made in various ways in different contexts. It is one of the objectives of our of our synthesis of contextualism and abstractionism to clarify the factors that need to be considered when making such decisions, not to recommend potential solutions.

4.3.2 Ambiguity of Goals

The abstractionist treatment considers that a series of binary decisions need to occur before the scheduling goal is achieved. However, like many work activities, the scheduling that our informants do involves a relaxed or incremental approach to commitment and task performance. For example, a person may wish to blank out a period for most purposes, but would accept interruptions that were important or urgent. Sometimes, multiple conflicting commitments, or double bookings, could be made with the expectation that some of them would later become moot. For example, a much-postponed meeting will be postponed again or attendance delegated to another at the last minute.

A system that treated all commitments as binary events may be too inflexible for the demands of the working practices that an administrative assistant describes in the following quote. Even a set of pre-programmed alternatives may not allow for types of ambiguity that occur.

> *A difficult thing that I have is maybe he says "I have got to see Dr. C. before such and such a date." Well, we look at the dates and I've got Dr. C.'s available dates so of course I take whatever times he's available and I may have to bump two or three people in order for him to do that. So I have to cancel whatever might be scheduled to try to work out other times.*

In this example, Dr. C. is more important to the informant and her principal than the people whose previously scheduled appointments get bumped. Here, then, is evidence for a goal that was neglected from the abstractionist analysis: *Know importance of participant*. Recognizing that this is a goal in the context should affect the operational requirements for a meeting scheduler, as we shall describe further in Section 3.4.

Thus, this part of the teleological model could be revised as follows:

Meeting scheduled
- ...
- Bumped meetings scheduled
  - Bumpable meetings known
    - Participants' importance known
    - Meeting details known
  - ...

Here 'bumpable' means capable of being moved to accommodate the meeting in question. Note that just identifying these goals does not mean that any of them should be automated or even supported by the application, merely that they apply in the context of use and that the specified system should not block their achievement. For example, it would be perfectly sensible to allocate *Participants' importance known* to the user, thereby leaving tacit the knowledge and judgment about relative importance or status.

The general lesson here is that contextual data is a good source of information about goal dependencies. Expanding a goal is often more difficult than it may appear, particularly since the achievement of a goal that refers to one object (e.g., scheduling a meeting) may conflict with the same goal's application to another object of the same type (i.e., another already scheduled meeting). Similar scenario fragments occur repeatedly in our transcripts in which informants describe situations that have to be handled carefully. Eliciting stories about special or typical cases seems to be a natural strategy for uncovering this information.

4.3.3 Workplace Artifacts and Object Identification

Office workers often develop ways of working that exploit properties of an artifact that were not deliberately designed into it for that purpose. The work environment is full of concrete artifacts, such as piles of paper, forms, sticky notes and equipment that support tasks or constrain the way they are done. Ethnographers working in RE and design have recognized this for some time. Abstractionists are likely to ignore such tacit cues, because they are absent from the formal communications among co-workers, and the resulting automated system may therefore make coordinated work more difficult then before.

Sometimes a generic scenario or critical incident can point to relevant artifacts. Sometimes artifacts are apparent in the work setting and can be asked about directly.

> Investigator: *You have three types of calendars there?*
>
> Informant: *Three types? I just printed that one out and reformatted it, it's the same thing. But there are different formats. There's like a month at a time. There's the week at a time, this is what I use because again I can block out the length of time. The thing about the monthly, it doesn't give me that luxury. It only says at 8:00 you've got this meeting but it doesn't tell me how long it lasts which is what I need to know and the nice thing about the monthly, though, it gives this banner option that says he's going to be gone for this conference over a period of three days. So I can put a banner across the whole week. But I can't do that in this [refers to the weekly] particular view.*

This administrative assistant uses the calendar as a visual mnemonic. Each calendar format displays time differently and makes different information available that is required to make scheduling decisions. We can also see that hard copies obtained from an on-line system are often more useful than the on-line version. For example, we learn that she will print out a small copy of the schedule for her manager. The small copy allows portability and is easy to read. Some of the incidental properties of the hard copy make it preferable even though it contains the same information as the on-line version and may be out of date.

Concrete artifacts like this can inform the development of teleological abstractions, but one has to be careful. After analyzing their formal description of goals by means of a theorem prover, Van Lamsweerde and colleagues report that they had left out an important goal: *Scheduler available* [Van Lamsweerde *et al.* 1995]. The natural (and, in retrospect, rather obvious) consequence of this new goal is that there must be some means of invoking the Scheduler when it is not available. At first sight to a computer scientist, this may seem to need something as simple or obvious as a command name that should be typed or an icon to be clicked. But, the administrative assistant's description of why she generates hard copy emphasizes that the Scheduler cannot always be available even though scheduling information can be. Satisfying *Scheduler Available* is therefore not a simple matter of providing some means to invoke the program but may involve carrying a portable device or generating hard copy data in advance.

The general lesson to learn from this is to look for the communicative and mnemonic uses of concrete artifacts as sources of information about states of affairs that characterize goal achievement. The precise form of the artifacts may be more important than their abstract information content, because their form (e.g., spatial distribution and layout) may suggest informal, tacit communication patterns and cues for remembering and scanning for information.

4.4 Actions and responsibilities

From the quotation given in Section 3.3, in which an administrative assistant discusses the fluid process she follows in scheduling a meeting between her principal and Dr. C., it is clear that any attempt to build into a meeting scheduler formalized knowledge about the importance or status of one meeting participant vis-a-vis the importance of participants in other meetings would be doomed to failure. This suggests more generally that goals should sometimes be realized by extended sequences of behaviors that gradually increase commitment and support negotiation, rather than by enforcing fixed operation sequences that are different to undo.

Sometimes, tasks are performed within guidelines rather than firm policies. This makes realizing the goals problematic, because the rules governing actions must be specified explicitly to be supported by a system. Preferred times for meetings are a good example of guidelines that do not have the stringency of policies and which, because of this, are often honored more in their breach than their observance. In our organization, we try to keep Fridays free of all administrative appointments. This sometimes makes Fridays a juicy target when scheduling "essential" administrative meetings: most people are free then because of the guideline, thus encouraging the guideline to be overridden. Automation can make this problem worse, not better. A member of technical staff in an engineering organization told one of us that since her organization had installed a shared on-line calendar and scheduling system, she no longer had lunchtimes to herself. Because most people are free at lunchtime (unless they explicitly enter dummy "meetings" with themselves at these times, something she said she was thinking of doing), lunchtimes become a likely candidate slot for any meetings despite their being non-preferred.

4.4.1 Responsibility and Operationalization

An agent who is responsible for an activity has been given authority to carry it out. In this sense, responsibility is not a measure of accountability but rather a way of defining control. The choice of interview subjects implicitly identifies a type of system agent—someone whose exclusive role in the organization is to schedule meetings.

> *Investigator: O.k. Who do you usually schedule meetings for?*
> *Informant: Our whole center and also Dr. P.'s individual personal meetings.*
> *Investigator: So he'll tell you, "Well we need to schedule a meeting for X?" or "Could you schedule a meeting for...."*
> *Informant: Well, lots of times I'm doing it on my own. We want to have a general faculty meeting. I have to schedule a meeting for [this group of] faculty [who] are in ten different locations on campus.*

From this dialogue, we learn that the center is spread out across a large area and that the user is responsible for scheduling meetings for an immediate supervisor and a much larger group of people. We can also infer that the user has some knowledge

regarding what kinds of meetings take place and possesses the authority to initiate these meetings. We can also infer and verify, that the recipients of a meeting initiation are primarily responsible for scheduling their own meetings.

Responsibility for information may also have very little to do with the organization structure. In the scheduling domain, people may guard their own schedules as personal and privileged information. A person may not be permitted to go directly to another's schedule, even a subordinate's, but instead must contact an intermediary.

> *People cannot come in here and just make an appointment unless they go through me ...It can't be accessible to just everybody in the [institution] because everything in the world would get scheduled and we have to prioritize. We have to maintain some sort of control over that. There has to be some way to block out some people or just have these people have access or whatever. Someone shouldn't be able to pull up his schedule and just schedule a meeting. He would say "I don't need that meeting. They need to see so and so." And sometimes that happens. I would say about a third of the phone calls that I get from people who want to meet Dr. H., he's not really the right person. And I'll redirect them to the right person. That's what I do when I screen. Do they really need to talk to Dr. H. or do they really need to talk to someone else.*

This person is responsible for screening incoming messages and phone calls to protect Dr. H's schedule. Some of these requests are redirected to others. A system that encroached on her discretion for purely rationalistic reasons (for example, allowing users, even senior people, to schedule Dr. H's time directly by seizing slots on his calendar) would not fit the context of use of this organization, causing numerous problems. The most obvious difficulty would be a finished schedule that contained too many unnecessary meetings involving too many people.

Another purpose behind inquiring about scenarios and incidents is to test one's understanding about responsibilities.

> Investigator: *If someone contacts you and says "I need to meet with [her principal]." How do you know that they take priority?*
> Informant: *I've been here long enough. I've been here almost two years. So I have a pretty good feel, on-campus, for who gets priority and also because of knowing who they are, I can sort of figure out the nature, I mean, I will screen. I will ask.*

One may want to know which decisions are at the discretion of the informant, when they would defer to another person, or when they would follow some procedures, and whether these procedures are firm policies or just rough guidelines.

One of the standard benefits of automation is that frequently recurring tasks can be packaged into single operations with the system performing (or appearing to perform) the replication of the tasks behind the scene. For example, in scheduling, a replicated scheduling goal for regular appointments can be operationalized as a single command.

> *Things that are internal like faculty meetings are pretty regular. There are a lot of regular kinds of meetings like faculty meetings. [Points to a highlighted piece of the calendar] He's teaching a course this quarter so I know this time is automatically blocked out. Things like that. Regularly scheduled meetings I go ahead and block a quarter at a time.*

An important lesson to be learned from this example is that goals and actions cannot be allocated to actors just because it seems logical to do so. An apparently rational task allocation may contravene subtle authority and responsibility relationships that exist in the context and which are not represented in a formal organization chart. Only by direct investigation of the context can these relationships come to light. In practice, therefore, it would seem sensible to search the contextual data carefully before making any allocation decisions.

4.4.2 Generic Scenarios and Critical Incidents

Generic scenarios are descriptions of typical courses of actions. The purpose behind eliciting generic scenarios is to find out how the goals are currently operationalized. When asked how she goes about scheduling a meeting, one respondent answered as follows:

> *I send email to everyone on the committee and ask them what is a good time for them and what is not.. Then I look at everybody's responses. I see if there's a best time based on the majority of the schedules. Then I see if I can get a room for that particular time. Then when I get the room, I send out an email saying this is the time and place that it will be.*

Usually, a process can follow a number of variant forms depending on the situation, and it is important to elicit scenarios for as many of them as feasible.

> Informant: *It can be anywhere from one other person, like say Dr. T. or a group of fifteen or sixteen people or so and they can be all in [the institute] or maybe scattered across the country.*
>
> Investigator: *Really. How does the across the country work?*
>
> Informant: *Well, I just have to get on the phone and start writing or getting people to give me numbers for various dates and times based on what Dr. H's schedule is.*

Critical incidents are narratives describing specific occurrences that were noteworthy. The purpose behind eliciting critical incidents is to help identify causes of successful or unsuccessful task performance.

> *One meeting I have that I'm in the process of working on right now is a meeting with all the senior faculty in the [institution]. This involves ten people, who also have schedules about as busy as he does. So what I do, he and I get together and decides on dates, depending on when he needs the meeting to occur by such and such a deadline and then look at his availability. Then I send some email out to this group of people, these people have an alias, and I ask "Okay, what is your availability for these dates." Like I start out with a grid [shows another piece of paper], and they start e-*

mailing me back, this is lunch and dinner, are you available for lunch, are you available for dinner on these dates? Then I just kind of wait until I hear back from everybody then I got to see what the best possible date looks like.

The purpose behind eliciting critical incidents is to help identify factors in successful or unsuccessful activity. *The general lesson learned from these critical incidents and generic scenarios is that the current way of working in a context reflects a design choice in which the goals are realized in a particular way*. A new system may operationalize the goals quite differently, but understanding the current process through scenarios and critical incidents serves at least two purposes: By describing actual behaviors, informants can help identify the underlying goals that the behaviors are supposed to support (by asking "why do you do that?" follow-ups); and the concrete behaviors help identify agency/responsibility issues and nonfunctional requirements (by asking "how often?" follow-ups).

4.4.3 Rich Communication Patterns

Rich, socially-mediated communication patterns can tell the designer a lot about a context. For example, many meeting messages in our corpus have a similar structure. First, there is an optional greeting followed by either a meeting-related message or reply, followed by optional non-meeting material, a copy of the message being replied to, and signature information. Each of these sections performs an important communicative function, even though some are not directly relevant to the scheduling task and therefore would be ignored by an abstractionist approach. Here is an example. The sections are tagged by sub-headings enclosed in angle brackets.

```
Date: Mon, 3 Jul 1995 13:18:40 -0400
To: C
From: B
Subject: Re: your mail

<Greeting>
C.

<Meeting Related Reply>
I'm back, too.  I'll be here Wednesday and Friday.  We can talk either of those days.

<Non-meeting Material>
Have you had a chance to talk to D?  I sent him something in the mail but haven't heard back.  CoC put out
a handbook several years ago called "The Teaching AssistantsHandbook:  A gentle Survival Manual for the new
GTA."  It's good and could be used as a resource.  The person listed as the contact person is x@cc.  Is he
or she still around?

I look forward to hearing from you.  It would be great if we could do something this summer, no matter how
small or casual.  Then plan bigger or the Fall.

<Signature>
B

<Copy of previous message>
>I'm back.  I recall that you may be on vacation.  Let me know when you
>get in.
>
>C
```

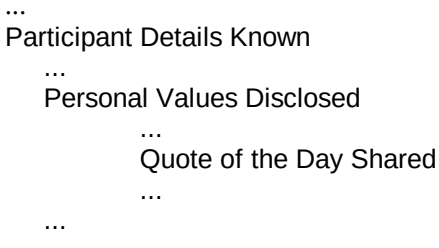
An abstractionist reaction to seeing many messages analyzed in this way might be to define structured messages or forms and conversational moves that abstract this common pattern. The classic example of a system built on such a foundation is the Coordinator [Flores *et al.* 1988], in which a theory of human communication is made explicit in the message types and valid user actions. However, as Robinson [1989] says, this approach may exclude, marginalize or illegitimize the "cultural" dimension of communication that is not so readily made abstract. Our purpose in summarizing rich communication patterns, then, is not to abstract common patterns and make them explicit in the requirements, but to search in these common patterns for revealing instances of informal communication that could be lost if communication were standardized.

The richness of the e-mail messages in our corpus stems in part from the fact that they are not the only communications being sent and received. Not only are the participants in the email dialogue involved in other scheduling activities simultaneously, they are also being bombarded by messages about different subjects. (Our abstractionist treatment of the meeting scheduler in section 3 finessed this entire problem by looking at the life history of a single meeting, ignoring the scheduling of other meetings, other time-management activities, and other workplace interactions. Because of this volume and diversity of email messages, it becomes necessary to include some sort of reminder, in addition to the subject heading, that refreshes the receiver's memory about the purpose of the meeting. Copying the last message is a fairly common behavior among users in our environment. Strictly speaking, this is not necessary, and the designer could make the communications more efficient and by providing some automatic thread or trail

mechanism concerning a given meeting. But including text from previous messages allows a sender to highlight a specific part of the communicative context in which the message is being sent. Like the air traffic controllers' stacks of papers [Sommerville *et al*. 1993], then, the precise form of the communication and the way it is constructed, can convey subtle cues that an abstraction might not.

The meeting message also serves as a type of social lubricant. The non-meeting related information is valuable in its own right to those involved in the dialogue even though it is barely related to the meeting being scheduled. The signature also helps to "humanize" the message and provide background information about the sender. (Some people in our environment attach a quote of the day to their signatures.)

Some of this background information may be irrelevant to scheduling, and the off-task content certainly is. An abstractionist posture would therefore emphasize only the scheduling-specific content, abstracting from it to the type of form or structured message that the system should mediate. This is one valid design option, but it would deny the users the opportunity to include information irrelevant or only tangentially relevant to the scheduling task in their scheduling messages. An alternative approach would be to recognize that informal social interaction and telling other users about oneself are important functions of electronically-mediated communication systems and that these should therefore be explicitly represented as goals. Perhaps, for example, we need to refine the goal *Participant Details Known* thus:



So now, as well as knowing about the person's email address, telephone number, preferred times for meetings, and so on, the system needs to know something about the user's values and how he or she thinks about matters of great import. Some personal time management tools, especially those allied with self-help programs, do go this extra step by providing embedded functions that bring up inspirational quotations which can be either pre-loaded or entered by the user. Others have a syndicated cartoon of the day. But explicitly representing these informal goals in the teleological model that refer to the users' need for inspiration and self-disclosure is surely going too far. Where would we stop? The point is that we cannot know in advance what all the users' goals may be. It is sufficient to conclude that there must be some way of carrying unstructured communication.

The general lesson to be learned here, then, is that workplace communications (and not just electronic communications, although these were the easiest for us to obtain and analyze) provide a rich source of information about how apparently non-task related utterances and messages serve the users' goals. Furthermore, naively ascribing goals to users based on the regular structure of workplace communications can lead to requirements for inflexible systems. We have not developed an elaborate theory of goal types, but it seems that attaining knowledge is an important category of goal in any system. Thus, we can say anthropomorphically that the system "knows about" a meeting, or that a user knows whether the meeting has been scheduled yet. These knowledge goals generally cannot be achieved until some form of communication takes place successfully. We therefore recommend looking carefully at the concrete communications that take place in the context, including those that are ostensibly informal or not directly task-related, to see what types of goal-related and non-goal related information tend to be carried together.

4.5 Obstacles and defenses

Contextual fit depends critically on the robustness of a system in the face of obstacles. This does not mean that a successful system must be fault-tolerant, but that it must be designed to work appropriately when the context in which it is embedded departs from the idealized assumptions made by its designers. An abstractionist analysis of the meeting scheduler may, for example, fail to account for cases in which an initiator of a meeting gets confused and invites the wrong people to it. The tacit assumption made by the abstractionist is that this cannot happen, because users know what they are doing and act rationally in pursuit of their goals. Whether this particular obstacle occurs enough to be worth worrying about is not the issue. Perhaps it does; perhaps it does not. Perhaps its consequences when it does occur are so minor and so easy to ameliorate by a few phone calls that the designer can ignore them. But these are questions about the context, and so it is to the context we must go to answer them.

In general, these are the basic questions that need to be answered:

- What obstacles to achieving goals occur in the context and how often do they occur?
- What are its consequences?
- What should we do about it?

We now consider how contextual data can be used to help answer each of these questions.

4.5.1 Obstacle Identification

Information about naturally occurring obstacles can help to make the requirements for a system more complete by defining what the system should do or what tasks it should support in the presence of environmental imperfections.

> Investigator: *Do they all have computers?*
> Informant: *Mostly, they have computers available to them.... They don't always have them in their office or for some reason, their network is not maintained as well and they have more problems with their mail being flaky and so they just aren't as tuned into it as we are because we're a little bit more technology based and so that can be a problem because you have to know who those people are and let them know that there's a message there for them to read...*

Because e-mail is such an integral part of the way that meetings are scheduled in the academic environment, people who fail to read their e-mail pose an obstacle to successful meeting scheduling. However, as the excerpt reveals, not all users exhibit the same motivation, inhabit the same environment, or possess the same tools. Therefore, a recovery activity other than sending a second message (which would be a rational solution for dealing with an unresponsive participant) is required.

In our transcripts, the mention of some obstacles was closely tied to a way of preventing or recovering from them. We had to ask about the obstacle after an informant had volunteered information about a recovery strategy:

> Informant: *I save all my messages at least for a year or six months or whatever seems to be a reasonable amount of time. That way if I make a mistake, I have them to refer back to.*
> Investigator: *Like what kind of mistake?*
> Informant: *Well, if I checked the wrong box for somebody when I'm going through, doing my spreadsheet, and then somebody sends me an answer and I already have an answer filled in for them, I can refer back to my messages.*

It is important to have some idea of the frequency of obstacle occurrences. Rare obstacles can be ignored if their effects are insignificant. It may be possible to recover from these obstacles creatively or just accept that the goal cannot always be met.

> *Most of the people that we deal with are computer literate people. They're either in hardware or software positions or they have email. Most of the companies we deal with have email. We have visitors coming in and we set up a lot of schedules for visitors, you know, and we do a lot of email that way but almost everything we do is email. Occasionally there will be a [visitor that will call to set up a meeting] but very seldom. Even with our outside visitors we do email.*

Contextual data is obviously a rich source of information about obstacles. Informants often do not understand what is meant when asked what could go wrong. *The general lesson to learn from these excerpts is that obstacles can be identified from information about other issues (in these cases, workplace artifacts and defense strategies).*

4.5.2 Obstacle Defense and Mitigation

Identifying obstacles is one thing. Overcoming them with a system design that robustly operationalizes goals is another. There are two generic ways to overcome obstacles: avoid them by pre-planned short-circuiting of the conditions that can lead to them or recover from them or ameliorate their effects by introducing special recovery behaviors. These mechanisms in turn require that we work out four types of system behaviors: those that predict obstacles before they occur, those that avoid the obstacles once predicted, those that detect that an obstacle has occurred, and those that recover from an obstacle. Although automation provides opportunities for shifting the balance from recovery/amelioration to detection/prevention, it is often worthwhile to find out how office workers currently cope with obstacles.

> *Email sort of helps in that respect because you can send out one message to everybody at the same time. And the only problem there is people who don't read email faithfully, you know, and there are some. So you have to find out who doesn't read their email regularly and make sure that you call them to make sure you say "I sent you an email message about [an appointment], please check your email." or something to get them to read it.*

The likelihood that a specific meeting invitee may not read his or her email, a potential obstacle, is predicted in advance. These predictions are derived from the user's experience. We also learn about the obstacle avoidance behavior, which is to send either a replacement message or a short warning message, such as a telephone call, to tell the person to check his or her mailbox.

> Investigator: *How do you find this out?*
> Informant: *Just by trial and error. People who don't respond to you, eventually you call them up and say "Did you read this email message I sent?" And you find out that they normally don't read it very often and then you have to go back and say I need to find another way to reach these people. I wish everyone read email, it would make this job easier.*

This time, the obstacle detection behavior is a "time-out", a long delay before a reply eventually makes the office worker suspicious that the recipient never read the message. The amelioration behavior is again a warning message.

Because of their familiarity with the context and its tasks, users tend to have already encountered many obstacles that the designer wants to investigate and will usually have developed defense or mitigation strategies that are useful sources of information. In particular, the information gathered may cause the designer to abandon a possible operationalization of a goal that would not be consistent with the strategies currently in use.

And that's how I do my scheduling now. I don't usually just pick one time and try to set up a meeting with one time in mind because then you end up having to send back messages multiple times because that one doesn't work, you have to start over again.

Here the obstacle is *No Feasible Schedule*. The likelihood that this obstacle will occur and the undesirable delay in sending further messages and collating replies means that it should be defended against where possible by providing as broad a range of possible meeting times as feasible. A possible operationalization of the *Meeting Planned with Negotiation* goal that involved the Scheduler proposing a single time and then waiting for replies would run into this negotiation. Although proposing a broad range of options may not sound like an obstacle defense strategy, really it is because it reduces the likelihood that a time conflict will arise.

Defensive strategies may be neither obvious nor what abstractionist analysis would suggest.

Informant: *Yeah, like for example if someone at [that department] does not read email, I'll take my message, print it down to a Word file and then fax it, because I have a fax machine on my computer in addition to the fax machine out there, so then I will fax him a copy of the message that I e-mailed.*
Investigator: *So what happens over there? Do they have a secretary or someone who picks up the message and takes the message to them?*
Informant: *Yeah who takes it to him and then he responds that way. That's what I'm learning to do. I try to do as little as I can by telephone because I think that takes much longer. Then you have to wait for people, and have to wait for responses...*

The initially formulated goals used in requirements development can be too ideal, making assumptions about system behavior that may restrict the range of allowable actions or fail to compensate for certain kinds of obstructions that can take place [Van Lamsweerde, *et al.*, 1995]. For example, an ideal goal for the meeting scheduler would be to assume that the transaction ends when the goal state of maximizing the number of participants that can attend and having a date set has been reached. In the example below, one of the participants is having difficulty making the established meeting time and location.

```
From k@cc.gatech.edu Wed Apr 19 09:00:37 1995
Subject: Meeting
Cc: k@cc.gatech.edu

J,

I'm running a bit late (about 25 minutes, I would estimate).
Sorry to keep you waiting.

K.
```

In this case, the negotiation is implicitly asking whether the meeting time can be extended by a certain amount. If the meeting never takes place due to an error on the part of one of the critical participants, the negotiation will be extended and a new time reached. In such a case, a useful subgoal of the Scheduler would be to establish a set of "backup" times to meet in the event of a failure. Another case where goals require weakening is the case where an assumption is made that every participant/agent communicates their time restrictions in the time specified. Van Lamsweerde, *et al.* [1995] give one retraction of a simplification that can be used to weaken this goal viz. "invited participants who did not react within the prescribed deadline are removed from the list of expected participants." The actual behavior of people who schedule meetings can be very different. One method of maximizing the number of participants is to resend the message.

```
From A, Fri Jun 30 14:29:25 1995
Date: Fri, 30 Jun 1995 14:29:23 -0400 (EDT)
To: B <B@prism.gatech.edu>
cc: B@prism.gatech.edu, C@cc.gatech.edu

> 2) Send out a message to everyone asking them if they are coming.  The
> addresses to which the message should be sent are:  X1@gatech.edu,
> X2@gatech.edu.  You might want to check with C
> (C@cc.gatech.edu) to make sure this are the best addresses to use.

So far I have 22 participants.
Should I send out another message of strong encouragement Monday or are
most people going to be gone between now and the 4th and we should just
assume a huge rush of people on the 5th?

A
```

In this example, the agent for the initiator is confirming whether a second message needs to be sent or whether there are a sufficient number of meeting participants to continue. Under the original goal, once an established deadline has been passed, the participants who haven't responded would just be dropped. By weakening the goal parameters, alternative actions can be taken to help maximize the number of meeting participants, fulfilling the primary goal.

Contextual data seems to be at its richest and least rationally structured when it comes to identifying and handling obstacles. One could use abstractionist methods to expand the potential set of obstacles and solutions, and we have developed techniques for generating scenarios that test specific goal/obstacle combinations [Potts 1995]. A combination of known approaches derived from

these observed behaviors and rationally derived solutions to some of the more obscure obstacles, may be sufficient to cover the needs of a system's users.

5 Discussion

We set out to show how a synthesis of two complementary design philosophies, abstractionism, and contextualism, can produce requirements that are more contextually appropriate than could be produced by abstractionist or contextualist methods alone. In section 3, we demonstrated how GR, an abstractionist method, is applied to generating the requirements for an interactive system, and how GR fails to illuminate many contextual factors that must be considered in the design. In section 4, we presented some contextual data gathered from our organization to show how they provide a rich source of information that supplement or alter the requirements developed using GR in isolation.

Three arguments could be made to refute the claim that a synthesis of abstractionism and contextualism is necessary for the development of better requirements. First, it could be argued that we have not shown that the additional goals and operational requirements discussed in section 4 really would lead to a more useful and contextually appropriate meeting scheduler than a wholly abstractionist approach. This is an empirical claim, and, as we pointed out in section 2.3.3, it will take much research to validate the synthesis of the two approaches. We hope, however, that the examples given in section 4, taken as they were from workplace interviews and documents, make our claim credible. Moreover they resonate with the results reported by others [Anderson 1994; Luff *et al.* 1994; Randall *et al.* 1994; Sommerville *et al.* 1993; Suchman 1983].

A second potential criticism is that we did not do justice to the purely abstractionist version of GR. If we had thought more deeply about meeting scheduling, perhaps our goals would have been richer and more complete, and there would have been no need to gather and analyze contextual data. If this were so, then contextualist methods simply consume time and resources without producing significant additional results. But, as Nielsen [1993] has shown, developers are very poor at predicting user needs. Things always look clearer in hindsight, and problems are always more easily solved in principle than in practice. While it may be true that simply thinking harder about the problem domain would, in principle, reveal many of the same insights that emerged from the contextual data, it is unlikely that designers working in the tradition of abstractionism would think about the context in the right way. Abstractionism encourages simplification, and some simplifications sweep under the carpet the very contextual issues that should receive focused attention. Section 4 contains many examples of contextual issues that have not been considered in previously published treatments of the meeting scheduler.

Neither of these arguments, or our responses are novel. A third potential criticism, therefore, is that we are saying nothing new, that it is well known that contextual factors affect the usefulness of systems and that context should be taken into account during RE. *However, we are saying more than that contextual data should be gathered when doing RE; we are saying that they should be gathered and used in particular ways*. Consider our discussion in section 4.2.2 about how informants' stories about special cases can suggest unforeseen requirements. (The particular example there is how a meeting with an important person can require previously scheduled meetings to be 'bumped'.) A wholly contextualist approach, one in which the RE process was conducted through the media of stories, storyboards, or mockups [Ehn 1988; Ehn and Kyng 1991], or in which an ethnographic thick description of the workplace culture was passed on to the designers, might not have the desired effect. Our specific recommendation in section 4 was to look systematically for instances where goals thwart other goals of the same type (i.e., successfully scheduling one meeting could necessitate rescheduling another). This heuristic could be applied in formulating what questions to ask informants, how to conduct live observations of office work, what coding schemes to choose for interpreting activity logs or videotape records or e-mail messages. Thus, our point is neither that contextualist data improves the requirements obtained from an abstractionist method (which is probably true), nor that abstraction is needed when interpreting contextual data (which is inevitable to some degree), but that the two approaches can support each other in a highly systematic and focused way.

Focusing a contextualist investigation through the lens of an abstractionist approach raises a further set of issues about preconceptions and bias. Early impressions about a context often turn out to be wrong, so we should be careful about restricting the focus or shutting off the investigation prematurely. Some authors advocate "quick and dirty ethnography" [Hughs *et al.* 1995] as a good way to incorporate contextual data into RE. This is controversial, however, because the results obtained are always more suspect than those obtained from more thorough or less focused investigations. The danger is that of committing the "fallacy of the ethnographic moment," in which a superficial exposure to the context under investigation leads to a compelling but incorrect insight about what is going on. In practice, of course, an interviewer or observer always has to ask *some* questions. Some qualitative researchers recommend preparing detailed plans or scripts [Lofland and Lofland 1984] in advance, and, given that these plans must be oriented around some theme, there is nothing essentially wrong with using the conceptual categories of the abstractionist approach for this purpose. As long as the focus suggested by the abstractionist approach is not positively misleading, we would argue that the focus it provides is beneficial because it helps generate a set of requirements.

In this paper, we have looked at a narrow sample of abstractionist and contextualist approaches, so our conclusions can only be transferred to other abstractionist methods with caution. How a given abstractionist method could be integrated with contextualist data gathering and interpretation methods will depend critically on the nature of the methods in question. However, we do believe that

the approach can be transferred to other abstractionist and contextualist methods. For example, OOA is concerned with identifying the kinds of object and actors in an enterprise. It concentrates on identifying object properties and responsibilities, the relationships between objects, and how they interact [Coad and Yourdon 1991; Coleman *et al.* 1994; Rumbaugh *et al.* 1991]. As mentioned in section 2.2, OOA methods do not provide very powerful ways to identify these categories. In OMT, the analyst is supposed to parse a text document, looking for phrases that indicate candidate objects and relations [Rumbaugh *et al.* 1991]. Coad and Yourdon [1991] tell the analyst to think hard about the problem domain, and they give some hints about what types of semantic categories often wind up as objects in system object models. Other writers [Jacobsen *et al.* 1992' Rubin and Goldberg 1992] tell the designer to extract objects from scenarios, but do not say which scenarios to explore or why.

Several qualitative research techniques look like obvious contextualist counterparts to these abstractionist strategies in OOA. Spradley [1970], for example, spends several chapters giving advice to the anthropology student about how to elicit conceptual classifications, taxonomies, and differentia in cultural domains. These cognitive anthropological techniques seem to lend themselves very directly to the early stages of most off-the-shelf OOA methods. And Spradley's 'grand tour' and 'mini-tour' elicitation methods are sufficiently close to what OO writers mean by "scenarios" that they too could probably be integrated quite smoothly with scenario-based OOA methods.

It is our contention that RE methods in the future must integrate techniques from both abstractionism and contextualism. Pure abstractionism may lead to consistent, coherent, and feasible requirements, but they may describe a useless system. Pure contextualism, however, may lead to improved background knowledge about the context, but it does not affect the writing of requirements in a well-understood and systematic way. We have discussed what a synthesis of abstractionism and contextualism means in practice and have given a concrete example to demonstrate how contextual information can systematically inform and improve requirements.

6 Acknowledgements

We would like to thank our colleagues who served as informants for the contextual study in section 3. Lara Catledge, Christiann Ertmann, Mike McCracken, Wendy Newstetter, and the reviewers of an earlier version of this paper made many helpful comments.

7 References

Alford, M. (1985), "SREM at the Age of Eight: The Distributed Computing Design System," *IEEE Computer 18*, 4, 36-46.

Anderson, R.J. (1994), "Representations and requirements: The Value of Ethnography in System Design," *Human-Computer Interaction 9*, 2, 151-182.

Antón, A. (1996), "Goal-Based Requirements Analysis," In *Proceedings of the 2nd International Conference on Requirements Engineering*, IEEE Computer Society Press, Los Alamitos, CA, pp. 135-144.

Antón, A., W.M. McCracken, and C. Potts (1994), "Goal Decomposition and Scenario Analysis in Business Process Reengineering," In *Proceedings of CAISE'94: 6th International Conference on Advanced Information Systems Engineering*, Springer, New York, NY, pp. 94-104.

Barker, R. (1990), *CASE* Method: Entity-Relationship Modeling*, Addison-Wesley, Wokingham, UK.

Bernard and H. Russell (1994), *Research Models in Anthropology: Qualitative and Quantitative Approaches*, 2nd edition, Altamira Press, Walnut Creek, CA.

Bustard, W. and P.J. Lundy (1995), "Enhancing Soft Systems Analysis with Formal Modeling," In *Proceedings of RE'95: 2nd International Symposium on Requirements Engineering*, IEEE Computer Society Press, Los Alamitos, CA, pp. 27-29.

Chandy, J.M. and J.Misra (1988), *Parallel Program Design: A Foundation*, Addison-Wesley, Reading, MA.

Checkland, P. (1972), "Toward a Systems-Based Methodology for Real-World Problem Solving," *Journal of Systems Engineering*, *3*, 2, 87-116.

Checkland, P. and J. Scholes (1990), *Soft systems Methodology in Action*, Wiley, Chichester, UK.

Chen P. (1976), "The Entity-Relationship Model: Toward a Unified View of Data," *ACM Transactions on Database Systems*, *1*, 1, 6-26.

Coad, P. and E. Yourdon (1991), *Object-Oriented Analysis*, 2nd edition, Prentice-Hall, Englewood Cliffs, NJ.

Coleman, D., P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes (1994), *Object-Oriented Development: The FUSION Method*, Prentice-Hall, Englewood Cliffs, NJ.

Dardenne, A., A. van Lamsweerde, and S. Flickas (1993), "Goal-Directed Requirements Acquisition," *Science of Computer Programming*, *20*, 1/2, 3-50.

DeMarco, T. (1978), *Structured Analysis and System Specification*, Yourdon Press, New York, NY.

Ehn, P. (1988), *Work-Oriented Design of Computer Artifacts*, Arbetslivscentrum, Stockholm, Sweden.

Ehn, P. and M.Kyng (1991), "Cardboard Computers: Mocking-it-up, or Hands-on Future," In *Computers and Work*, J. Greenbaum and M.Kyng, Eds., Erlbaum, Hillsdale, NJ, pp. 168-195.

Flach, J.M. (1993), "The Ecology of Human-Machine Systems I: Introduction," *Ecological Psychology*, *2*, 3, 191-205.
Flores, F., M. Graves, B. Hartfield, and T. Winograd (1988), "Computer Systems and the Design of Organizational Interaction," *ACM Transactions on Office Information Systems*, *6*, 2, 153-172.
Geertz, C. (1973), "Thick Description: Toward an Interpretive Theory of Culture," In *The Interpretation of Cultures: Selected Essays*, C. Geertz, Ed., Basic Books, New York, NY, pp. 3-30.
Green, S. (1994) "Goal-Driven Approaches to Requirements Engineering," Technical Report TR-93-42, Imperial College of Science, Technology, and Medicine, Department of Computing, London, UK.
Grudin, J. (1989), "Why Groupware Applications Fail: Problems in Design and Evaluation," *Office: Technology and People*, *4*, 3, 245-264.
Grudin, J. and L. Palen (1995), "Why Groupware Succeeds: Discretion or Mandate," In *Proceedings of ECSCW'95: European Conference on Computer-Supported Cooperative Work*, Kluwer, Boston, MA, pp. 263-278.
Guttag, J.V., J.J. Horning, and J. Wing (1985), "An Overview of the Larch Family of Specification Languages," *IEEE Software*, *2*, 5, 24-36.
Holtzblatt, K. and S. Jones (1993), "Contextual Inquiry: A Participatory Technique for System Design," In *Participartory Design: Principles and Practice*, A. Namioka and D. Shuler, eds., Erlbaum, Hillsdale, NJ, pp. 177-210.
Hughes, J., V. King, T. Rodden, And H. Anderson (1995), "The Role of Ethnography in Interactive Systems Design," *ACM Interactions*, *2*, 2, 57-65.
Hutchins, E. (1995), *Cognition in the Wild*, MIT Press, Boston, MA.
Jackson, M.A. (1995), *Software Requirements and Specifications*, Addison-Wesley, Reading, MA.
Jacobson, I., M. Christerson, P. Jonsson, and G. Overgaard (1992), *Object-Oriented Software Engineering: A Use-Case Oriented Approach*, Addison-Wesley, Wokingham, UK.
Jones, C. (1990), *Systematic Software Development Using VDM*, 2nd edition, Prentice-Hall, New York, NY.
Kirlik, A. (1993), "Requirements for Psychological Models to Support Design: Towards Ecological Task Analysis," In *An Ecological Approach to Human Machine Systems I: A Global Perspective*, J. Flach, P. Hancock, J. Caird, and K. Vicente, eds., Erlbaum, Hillsdale, NJ, pp. 68-120.
Kling, R. (1980), "Social Analysis of Computing: Theoretical Perspectives in Recent Empirical Research," *Computing Surveys*, *12*, 1, 61-110.
Lincoln, Y.S. and E.G. Guba (1985), *Naturalistic Inquiry*, Sage Publications, Newbury Park, CA.
Loffland, J. and L. Loffland (1984), *Analyzing Social Settings*, Wadsworth, Belmont, CA.
Luff, P., C. Heath, and D. Greatbatch (1994), "Work, Interaction, and Technology: The Naturalistic Analysis of Human Conduct and Requirements Analysis," In *Requirements Engineering: Social and Technical Issues*, M. Jirotka and J. Goguen, eds., Academic Press, London, UK, pp. 259-288.
McMenamin, S. And J. Palmer (1984), *Essential Systems Analysis*, Yourdon Press, New York, NY.
Meldrum, M., M. Lejk, and P. Guy (1993), *SSADM Techniques: An Introduction to Version 4*, Chartwell Bratt, Bromley, UK.
Morgan, G. and C. Smircich (1980), "The Case for Qualitative Research," *Academy of Management Review*, *5*, 4, 491-500.
Morgan, G. (1986), *Images of Organization*, Sage Publications, Newbury Park, CA.
Moore, A. P. (1990), "The Specification and Verified Decomposition of System Requirements using CSP," *IEEE Transactions on Software Engineering*, *16*, 9, 932-948.
Nielsen, J. (1993), *Usability Engineering*, Academic Press Professional, Boston, MA.
Norman, D.A. (1992), *Turn Signals are the Facial Expressions of Automobiles*, Addison-Wesley, Reading, MA.
Potts, C. (1995), "Using Schematic Scenarios to Understand User Needs," In *Proceedings of DIS'95" Designing Interactive Systems*, ACM, New York, NY, pp. 247-256.
Randall, D., J. Hughes, and D. Shapiro (1994), "Steps towards a Partnership: Ethnography and Systems Design," In *Requirements Engineering: Social and Technical Issues*, M. Jirotka and J. Goguen, Eds., Academic Press, London, UK, pp. 241-258.
Robinson, M. (1989), "Double-Level Languages and Cooperative Working," In *Proceedings of the Conference on Support, Society, and Culture: Mutual Uses of Cybernetics and Science*, Software Engineering Research Center, Utrecht, The Netherlands, pp. 79-114.
Ross, D.T. (1977), "Structured Analysis (SA): A Language for Communicating Ideas," *IEEE Transactions on Software Engineering*, *3*, 1, 16-34.
Rubin, K. and A. Goldberg (1992), "Object Behavior Analysis," *Communications of the ACM*, *35*, 9, 48-62.
Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen (1991), *Objecte-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ.
Sommerville, I., T. Rodden, P. Sawyer, R. Bentley, and M. Twidale (1993), "Integrating Ethnography into the Requirements Engineering Process," In *Proceedings of RE'93: International Symposium on Requirements Engineering*, IEEE Computer Society Press, York, UK, pp. 165-173.

Spivey, J.M. (1988), *Understanding Z: A Specification Language and its Formal Semantics*, Cambridge University Press, Cambridge, UK.

Spradley, J. (1979), *The Ethnographic Interview*, Holt Rinehard-Winston, Fort Worth, TX.

Stowell, F. and D. West (1994), *Client-Led Design: A Systematic Approach to Information Systems Definition*, McGraw-Hill, London, UK.

Suchman, L. (1983), *Plans and Situated Action: The Problem of Human-Machine Communication*, Cambridge University Press, Cambridge, UK.

Van Lamsweerde, A., R. Darimont, and P. Massonet (1995), "Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt," In *Proceedings of RE'95: 2nd International Symposium on Requirements Engineering*, IEEE Computer Society Press, York, UK, pp. 194-203.

Vicente, K. and J. Rasmussen (1993), "The Ecology of Human-Machine Systems II: Mediating 'Direct Perception' in Complex work Domains," *Ecological Psychology*, *2*, 3, 191-205.

Yu, E.S.K. and J. Mylopoulos (1994), "Using Goals, Rules, and Methods to Support Reasoning in Business Process Reengineering," In *Proceedings of the 27th Hawaii International Conference on System Sciences, Vol. IV, Information Systems: Collaboration Technology, Organizational Systems and Technology*, IEEE Computer Society Press, Los Alamitos, CA, pp. 234-243.