ScenIC

Guidebook for the Scenario-Based Inquiry Cycle Method

Colin Potts
Georgia Tech, College of Computing
March 9, 1997

-

I .Introduction

A .Purpose

This document describes the ScenIC methodology as used to support the evolution of legacy software in the MORALE project (Mission-Oriented Requirements Analysis and Legacy Evolution). The emphasis of this document is the forward engineering of requirements for new systems or new versions of existing systems. Other guidebooks in the MORALE series deal with architectural evaluation and reverse engineering.

ScenIC builds on a generic model of requirements and design activities known as the Inquiry Cycle. The Inquiry Cycle is a process model and an associated prescriptive framework for acquiring, analyzing and refining requirements descriptions and design specifications for systems. The Inquiry Cycle is method-neutral. Specializing the Inquiry Cycle process for a specific method or collection of techniques involves selecting a set of standard representations for describing the system and a set of discussion guidelines to direct the refinement process.

ScenIC, as the acronym suggests, is a specialization of the Inquiry Cycle (IC) that emphasizes the invention of scenarios and the refinement of requirements and early designs based on an analysis of these scenarios. ScenIC provides the following features:

b Descriptions of system purpose in terms of goals and their dependencies.

b Scripts: descriptions of scenarios (concrete examples of envisaged system behavior).

b Guidelines for identifying and refining goals and for realizing goals in terms of behaviors of system components.

b Representations of the rationale underlying these refinement decisions.

ScenIC is an abstract refinement method and does not require the adoption of a particular project organization or team practices. For example, ScenIC can be used in conjunction with contractual development projects in which the requirements are elicited by staff of a contracting organization interviewing customer representatives; equally, however, it can be used to focus discussions in team-based decision-making procedures, such as Joint Application Design sessions or Participatory Design "future workshops." The insertion of ScenIC guidelines into project organizations and team practices is not considered in this guidebook.

The remainder of this section introduces ScenIC and its context in the evolution of legacy systems. We start by explaining the philosophy of requirements analysis and design as the incremental development of descriptions and what ScenIC descriptions are about and how they are structured. We then provide a quick summary of the ScenIC method and an outline example. In Section II, we describe the method in more detail, concentrating on the knowledge or information that is required when applying the method, the information that it produces, a step-by-step guide that emphasizes the refinements of this information, and a set of evaluation criteria for reviewing ScenIC descriptions. Section III provides an example in more detail, the addition of some functionality to a web browser. At the end of the guidebook are a glossary of terms and an annotated bibliography of research and practice.

A .ScenIC as a Strategy for Incremental Description

1 .Software as Description

In the Inquiry Cycle and its specializations like ScenIC, we take seriously Michael Jacksonõs thesis that software engineering amounts to the production of descriptions.

> Description is at the heart of software development. It is not an exaggeration to say that the whole business of software development is making descriptions. Programs are descriptions of machines; requirements are descriptions of the application domain and the problems to be solved there; and specifications are descriptions of the interface between the machine and the application domain. Every product of development is a description of some part or aspect or property of the problem or of the machine or of the application domain.
>
> [Jackson 1995 p. 3]

ScenIC, then, is a method for producing descriptions of a systemõs requirements. As such, it involves recommendations for expressing information in these descriptions, analyzing and criticizing descriptions, and refining them so that they better capture requirements.

a)Subject of ScenIC Descriptions

The subject of a description is whatever the description describes. Requirements are usually thought of as being descriptions of what a system should do (and not how it should accomplish its requirements). We agree with Jackson, though, that a requirements description usually includes a lot of factual information about the application domain as well as information about the problem to be solved. In ScenIC, therefore, requirements descriptions may have application knowledge, the current situation or problem, envisaged or hypothetical solutions, and desired goals or system behaviors as their subjects. To avoid confusion, these subjects tend to be described in different kinds of description, but for now it is only necessary to be aware of the generality of subject matter in ScenIC. Moreover, because ScenIC is used to help guide the evolution of legacy systems, the subject matter of a ScenIC description may legitimately be the current architecture of the system (that is òhowó as well as òwható information).

It is not necessary in ScenIC to distinguish between software systems and systems in general. In aerospace, Defense and industrial applications, this distinction is sometimes held to be important because it reflects organizational boundaries and responsibilities. There is no need to abandon such a deeply rooted distinction when using ScenIC, and in this guidebook, we will be concerned exclusively with software requirements, but the concepts of goal, agency, responsibility and behavior are general enough that they could be used to describe other types of system, and as you will see even software-only requirements can be understood only in the broader context of the physical and organizational environment of the system.

Distinguishing between a system and its environment reflects a division of responsibility between the designer and others. For example, in designing an interactive system, the designer is responsible for the system but not for the behavior of the users. The designer cannot implement users, even if the design of the system influences their behaviors, but the designer does implement the system and is responsible for its qualities.

Do not confuse this distinction between system and environment with another distinction: that between an actual situation and a desired situation. The actual situation in the case of an evolving system includes both system and environmental parts, as does the desired situation reflected in the requirements.

a)Moods of ScenIC Descriptions

Jackson distinguishes between two subjects of descriptions in terms of the grammatical mood of the verbs that are used. A description of an actual situation is a collection of statements of fact in the indicative moode. The real test of such a description is whether the claims it embodies are actually true. If a description states that overtime is always billed at time-and-a-half, or that only department managers may authorize travel plans, or that the architecture of the system currently has the database updates being performed on a central server, then these are all statements of fact that are either true or false. Whether they are desirable or undesirable is not the issue; the issue is whether they are true. We refer to this type of description as a *factual description.*

Whereas descriptions of the current situation employ the indicative mood and are judged by their accuracy, descriptions of a desired situation employ the optative mood and are judged by whether the situation described really meets the readerõs values. Thus, a requirement to reduce travel authorization time by fifty percent or to decentralize and replicate mission data are neither true nor false: but they may be more or less desirable. We refer to this type of description as an *ideal description*.

A third type of description, unrecognized by Jackson, refers to hypothetical situations and tends to employ the subjunctive mood. Thus an imagined situation or behavioral scenario describes a possible state of affairs to which the customer and designer are not yet committed. Such a situation does not exist, so the description cannot be said to be true of false; and it is not yet committed to, so it is not desirable or not. To the extent that hypothetical descriptions are like factual descriptions, we judge them by the plausibility of their assumptions and the feasibility of their occurrence, and to the extent that they are like ideal descriptions, we judge them by the desirability of their consequences and the insights that they provide about the requirements. We refer to this third type of description as a *hypothetical description*.

The following table summarizes the differences between the three types of description. We need all three in ScenIC. Ignoring the ideal descriptions is obviously not appropriate in a method aimed at clarifying requirements; ignoring factual descriptions is a poor idea also, because the desirability of requirements can often only be judged against assumptions of fact; and finally, to ignore hypothetical descriptions would be to assume that statements of need can be derived and analyzed in a purely rational manner without imagining or illustrating what the **Formality of ScenIC Descriptions**

	Factual descriptions	*Ideal descriptions*	*Hypothetical descriptions*
Mnemonic	òAs isó description	òTo beó description	òMaybeó description
Examples	Domain models Architecture of system	Statements of need Goals or objectives Software/system requirements	Cases Scenarios
Grammatical mood	Indicative	Optative	Subjunctive
ScenIC representations	Domain model Actors in the system (components) and its environment (agents).	Teleological model Behavioral specifications	Scenario scripts

Descriptions may be expressed with varying degrees of formality. At one extreme is informal oral or written text, while at the other are languages with formal semantics that admit the possibility of formal inference. In between are schemes that provide for some structuring of descriptions so that quality checks may be carried out without the description itself being processed. ScenIC includes informal and semi-formal representations for descriptions of the factual, ideal and hypothetical. Formal representations akin to KAOS [REF] and a ScenIC interface to SAAM [REF] and SBF [REF] are under investigation.

a)Language of Descriptions

The ScenIC philosophy is that the process of developing descriptions (that is, expressing, criticizing and refining them) is more important than the nature of the representations used to express the descriptions. ScenIC provides a number of sub-languages for expressing factual, ideal and hypothetical descriptions, but it is not necessary to use these sub-languages to use ScenIC. Natural language, other textual languages, and graphical languages may also be used. For example, goal hierarchies and networks are quite common in artificial intelligence research, management science and human factors task analyses. The ScenIC representation for teleological models can easily be replaced by any other representation of goal hierarchies without losing the methodological support of the method. Or the representation for semi-structured critical rationale may be replaced by IBIS [REF] or Design Space Analysis [REF] if the user prefers.

1 .The Generic Inquiry Cycle Framework

In the Inquiry Cycle (IC) framework, the process of developing a description or set of related descriptions involves three intertwined and continually executed activities: expression, criticism and refinement. IC is a generic framework for describing and supporting design and planning methods, not just the production of system requirements. During the application of any method, there will be large volumes of informal and formal information that describe the status of the evolving design, plan or requirements specification. We call this information the "corpus" of descriptions. One way of thinking of the corpus is as an integrated model of the current, proposed and envisaged systems and their environments, together with process information such as who proposed what and when, why a decision was made, etc. Such an integrated model presupposes an automated repository and update, access and analysis tools, and it presupposes methodological policies for maintaining the consistency of the model. But another way of thinking of the corpus is as a collection of documents (whether on paper or maintained electronically does not matter much) that collectively represent all the information. The integrated model view of the corpus obviously has great promise in supporting a complex development process, but the required tools are not all available, and the costs in routinizing the representation of descriptions and the processes for expressing, criticizing and refining them would probably be too onerous. In the remainder of the guidebook, therefore, we will assume that the corpus consists of many independent (though conceptually related) documents of varying levels of formality.

In this section, we will discuss IC in the context of requirements acquisition and elaboration. To illustrate the repeated application of the canonical IC step of expressing some information, criticizing it and refining the description accordingly, consider how a single mission-oriented goal might be transformed into a functional requirement. First, the mission-oriented goal is written down (i.e. expressed in the corpus). Then, a discussion may ensue that explores several issues: What does the goal really mean? How can the goal be fulfilled by system functions and user actions? How does the goal interact with others? What could happen in the environment that would block the fulfillment of the goal, and how should the system be specified so that it stops this from happening or recovers when it does? Each of these discussions may refer to the contents of the corpus or add auxiliary information to it (such as scenarios or documentation of trade-off analyses). Finally, the resolution of each of these issues may lead to a refinement of the corpus. This refinement may take the form of a change to the element of the corpus that initiated the discussion (i.e. a refinement of the goal) or a change to another element (i.e. a refinement of another goal) or the addition of a new entity in the corpus (e.g. the introduction of a functional requirement that traces from the goal).

ScenIC is a scenario-based method for acquiring, analyzing and elaborating requirements. It is a specialization of the IC, because it consists of several representations for expressing information in the corpus, several standard discussion templates that are triggered by patterns in the corpus, and standard refinements that modify the content of the corpus. The objective of ScenIC is to transform a corpus consisting of information mainly about the architecture of an existing system and mission-oriented goals for a new version of the system into a corpus consisting of requirements for the new system and rationale for those requirements. The distinguishing features of ScenIC as a requirements analysis method are the heavy and systematic emphasis on mission-oriented goals and their refinement, the pervasive use of scenarios at varying levels of detail to help stakeholders understand the implications of possible requirements, and rich analysis heuristics for identifying patterns in the corpus that deserve stakeholder discussion and scenario-based exploration.

A .Overview of the ScenIC Method

The ScenIC framework is depicted in Figure <ScenIC>. Inputs to the process come from the customer representative and the "Discover Architecture" component of the MORALE architecture. Outputs go to the "Evaluate Architecture" and "Implement Architecture" components of the MORALE architecture.

>>> Figure <ScenIC> about here.

1 .The overall MORALE architecture

-- This part is common across guidebooks. Include Figure <MORALE>.

1 .The ScenIC architecture

a)Inputs to ScenIC

The inputs to ScenIC consist of mission-oriented goals (from the customer), existing requirements (from the customer), and architectural constraints (from "Discover Architecture").Mission-oriented goals and existing requirements may be introduced in several forms. The most easily understood and managed is in the form of informal documentation. Goals and requirements are unlikely to be clearly differentiated in customer-generated documentation. Some guidelines for separating them are given below, as are guidelines for inferring goals that may lie behind documented requirements.

Other goals and requirements may come interactively and incrementally through interviews or meetings. As explained above, we do not present guidelines on conducting interviews or team-based requirements gathering meetings. However, we do present some guidelines on gathering goals and requirements during interviews and meetings.

Mission-oriented goals and existing requirements are purely informal statements either embedded in documents or summarized from interviews and meetings. Goals and requirements produced by the ScenIC method have a semi-formal expression that is described in Section <III.B.???>.

Architectural constraints are relevant for all evolving systems. Some decisions that are made during the requirements determination process involve the allocation of operational requirements to system components. A system may only implement requirements that are feasible in a given architecture. If the architecture is constrained to an existing structure, as it usually is in an evolving system, some requirements may be feasible whereas others are not. Architectural constraints are therefore an important influence on the ScenIC process.

The format of architectural constraints is expanded on in Section <III.B.???>.

a)Outputs from ScenIC

The outputs of ScenIC consist of requirements specifications and requirements rationale.

Requirements specifications describe the required system in enough detail for implementation. They consist of functional requirements for the required system, quality requirements for the required system, and environmental assumptions.

Functional and requirements for the required system are statements that must be true of the required system once implemented. Functional requirements specify what the required system must do in specific situations. Quality requirements specify the degree to which the required system must meet specified quality criteria. Environmental assumptions are assertions about the environment that may be assumed to be true.

We shall refer to all three types of statement, and not just the functional and quality requirements for the required system, as "requirements".

Functional requirements, quality requirements and environmental assumptions may all be attributed to components of the required system or environment. In the case of an evolving system, the feasibility and detailed refinement of many functional and quality requirements depends on the architecture of the current version of the system. Even in the case of a new system, there may be obvious architectural constraints on the structure of the required system (perhaps suggested by the suitability of commercial off-the-shelf or other non-delivered components). In these cases, the functional and quality requirements may be further attributed to architectural components, and not just attributed to the required system as a whole. (We will leave to others the debate as to whether this constitutes premature design.)

Requirements rationale is a collection of representations that explain why the requirements take the form they do. Rationale includes pre-requirements traces, scenarios, and semi-structured discussion records.

Pre-requirements traces are traces back from the requirements to originating statements of various kinds. The most systematically preserved form of pre-requirements traceability in ScenIC is traceability back to mission-oriented goals. Other requirements may be traced back to statements made by stakeholders in meetings or documents.

Scenarios are descriptions of interesting usage patterns for a system. Scenarios may be constructed for the current version of the system, the required system once specified, or for an intermediate (partly specified and putative) system. Scenarios are used to help elaborate the requirements in such a way that they fulfill the mission goals and are feasible in the context of an architecture.

Scenarios may be elaborated for typical, anomalous or otherwise significant usage patterns.

Semi-structured discussion records are records of discussions about the requirements that preserve the major points of the discussion, separating those points from the text of the discussion summary itself. Such points include the stakeholders contributing to the discussion, the major issues or alternatives discussed, the reasons put forth in support of an alternative, and other descriptions referred to during the discussion (e.g. scenarios, mission-oriented goals).

1 .Overview of ScenIC

Because ScenIC can be thought of as a method for transforming one kind of corpus into another, we shall describe ScenIC in terms of its start and end corpus states and the refinements that collectively effect the transformation between them.

The ScenIC corpus consists of two kinds of information: information that cannot be formally manipulated ("unstructured" information), and information that can be formally manipulated ("structured" information). Unstructured information normally takes the form of natural language text (oral as well as written) and sketches. Structured information is expressed in one of the ScenIC sub-languages or expression formats. The start state of ScenIC consists of unstructured input from stakeholders about their requirements for a new version of a system and structured representations of architectural information about the current version.

a)Ideal Descriptions in ScenIC: The Teleological Model and Requirements

ScenIC does not consist of a series of steps that the analyst must perform in order. Rather, it consists of a collection of expression/discussion/refinement cycles that collectively transform the initial information into an acceptable requirements specification. We shall describe these cycles and their constituent representations in more detail in Section II. Here, briefly, is an overview of the major refinements.

Goal identification: A mission-oriented goal is identified in the unstructured information or is rationally reconstructed from the actions performed by components of the existing architecture.

Goal expansion: A goal is expanded into conjunctive or disjunctive sub-goals, and dependencies among goals are identified and recorded.

Obstacle identification: The obstacles that can thwart or block a goal or action are identified and are evaluated to see whether the risk of their occurring outweighs the cost and inconvenience of dealing with them.

Obstacle defense: In the case of obstacles that have been determined to be worth defending against, subsidiary defensive goals or actions are identified to prevent or reduce the likelihood of the obstacle occurring.

Obstacle mitigation: In the case of obstacles that have been determined to be worth mitigating, subsidiary mitigation goals or actions are identified that recover from or reduce the harm done by an obstacle that occurs.

Responsibility allocation: A goal or action is allocated to the required system, an architectural component of the required system, or an actor in the environment.

Goal realization: One or more actions are identified that fulfill a goal.

a)Hypothetical Descriptions in ScenIC: Scenarios

ScenIC gets its name from the extensive role played by scenario analysis in the discussions that lead to most of these types of refinement. Below are the uses of scenarios in each of the types of pre-refinement discussion.

Goal identification: Analysts and stakeholders can easily miss goals that pervasive goals (those that typically map to quality). When exploring scenarios it is common to find something wrong with the envisaged system that violates a hitherto unrecognized goal.

Goal expansion: Similarly, goals may have interdependencies that are only recognized when they are explored in the context of concrete scenarios.

Obstacle identification: The consequences of an obstacle occurring may be explored in a scenario to see whether it is worth developing a defensive or mitigation strategy to deal with it.

Obstacle defense and mitigation: As well as normal cases, scenarios are constructed to evaluate the effectiveness of obstacle defense and mitigation strategies, particularly in cases where significant obstacles co-occur.

Responsibility allocation: Alternative allocations of responsibility (specifically, alternative allocations of actions to architectural components or actors in the environment) may be evaluated by comparing scenarios that explore the different allocations.

Goal realization: Scenarios can only be used to explore goals that are realized by actions. Early analysis of scenarios therefore requires a tentative commitment to an allocation strategy that can be explicitly questioned by analyzing the scenarios that result.

a)Factual Descriptions in ScenIC

- No commitment to any kind of representation.

- Mapping of agency onto architectural components

1 .Rationale Capture

In ScenIC, rationale takes three forms:

1 .Pre-requirements traces from goals and originating statements to requirements. These goals and originating statements explain why requirements are present by explaining their origin or higher level expression in mission-oriented terms.

2 .Scenarios that illustrate desirable behaviors implied by the requirements or undesirable behaviors prohibited by the requirements. Scenarios explain why requirements are present by illustrating desirable or undesirable behaviors.

3 .Semi-structured records of IC discussions. These records are structured in that they have a well-defined

organization, but are unstructured in that the contents of the records are informal text as written or recorded during or after the discussion. Rationale records explain why requirements are or are not present by justifying the reasoning that contributed to the decision.

1 .Evaluation of ScenIC Specifications

A ScenIC-produced requirements specification is judged against the following criteria:

b Do the goals in the corpus from which the requirements are derived accurately record the stakeholders' mission-oriented intent for the system? Goal accuracy is important, because a system developed to fulfill the wrong goals is very unlikely to be effective. Goal accuracy is evaluated by careful feedback-eliciting questions to stakeholders and exploring scenarios with them.

b Do the obstacles in the corpus from which auxiliary goals and required actions are derived correspond with the likely environmental contingencies that the system will face? Obstacle accuracy is important, because obstacle defense and mitigation strategies may lead to many auxiliary requirements in the corpus. If these requirements are not necessary, the system will be avoidably expensive to implement. Obstacle accuracy is evaluated by eliciting feedback from domain experts and performing root-cause analyses on likely problems.

b Are the requirements complete? Completeness refers to the extent to which the stakeholders' intentions are reflected in the requirements. This is an informal criterion, because the requirements are structured whereas the stakeholders' intentions are not only unstructured but unexpressed. However, the requirements can be checked exhaustively against the originating goals and possible obstacles. With goal accuracy and obstacle accuracy assumed, therefore, requirements completeness can be assessed systematically by assessing whether a stated goal is fulfilled by the requirements or explicitly allocated to the environment, and by assessing whether an obstacle is adequately defended against or mitigated by auxiliary goals and system behaviors.

b Do the requirements exclude all unnecessary requirements ("gold plating")? Gold plating is known to be one of the primary risk factors affecting software development, so it is important to find it and weed it out as early as possible. Because an unnecessary requirement is one that is not needed for the system to meet stakeholder intentions, it cannot be assessed directly. In this respect, gold plating is the converse of completeness, and, as in the case of completeness, the requirements can be checked exhaustively against the originating goals and possible obstacles. With goal accuracy and obstacle accuracy assumed, some gold plating can be identified with precision. A necessary but insufficient condition for a requirement being gold plating is for there to be no trace back to any goal or obstacle. More subtle forms of gold plating are dealt with below.

b Do the requirements reflect an optimal realization of the goals in the corpus? There are many possible systems that could meet a set of mission-oriented goals. These systems exhibit different behaviors, different degrees of automation, and different allocations of responsibilities to environmental actors. Part of the evaluation of alternative realizations of a set of goals is whether each realization is architecturally feasible and which are superior with respect to some measurable criteria. These architecturally-aware evaluations are discussed in the SAAM guidebook.

  Another part of the evaluation of realizations, however, involves the degree to which each fits into the environment of the system. Scenarios and feedback-eliciting questions about skills or properties of users, interfacing systems, organizations and devices are used to ascertain the quality of a goal realization.

  A third form of evaluation of the realization of a goal is to ask whether it is as simple as it needs to be. Goals that are fulfilled with unnecessary complexity are a second form of gold plating. This form of gold plating is less easily detected than requirements that do not trace back to goals or obstacles at all and can be identified only with the help of domain and design expertise.

b Are the scenarios sufficiently diverse to induce confidence in their use to analyze the requirements? ScenIC is predicated on the belief that scenarios are valuable tools in the understanding of complex systems. They can be misleading, however, when they fail to explore the space of design alternatives and environmental contingencies with sufficient coverage. Insufficient coverage means that many requirements defects are not detected because the situations are never explored that would make the defects apparent. To ensure greater coverage, ScenIC includes some (admittedly weak) coverage heuristics for generating scenarios. For example, the fulfillment of every goal should be illustrated in at least one scenario; and the occurrence of every combination of obstacles and their defensive or mitigation strategies should also occur in at least one scenario. Significant combinations of obstacles may also be checked, but what makes a combination significant depends on the domain and can only be determined by an expert.

b Are the scenarios sufficiently detailed to induce confidence in their use to analyze the requirements? As with the coverage issue, scenario detail is vital for developing a good understanding of requirements-related issues. Several techniques can help to make scenarios detailed enough. One is to name all actors and data variables mentioned in the scenario. (Thus, in a library system, one would not explore the "Patron returns book late" scenario, but the "Colin returns 'War and Peace' two weeks late" scenario.) Another is to manage scenario scripts at varying levels of detail that are appropriate for different types of requirements investigations. (Thus, one version of the scenario above would describe what happens in terms of the library's business processes and its fine policies; a second would describe what would happen in terms of time-ordered operations performed by the participating actors; and a still more detailed version would involve events at the level of user interface dialogs and device interactions.)

b Will the system as specified fit into its broader context? There is always a possible residual source of requirements defects that is not removed even when the questions listed above have been answered. Sometimes a system does not "fit", it cannot be used effectively, and it just does not seem useful. In principle, we could predict and avoid this happening by accurate modeling of the stakeholders' goals and careful refinement of goals into requirements. In practice, however, this is not possible, because all the activities in requirements engineering must be undertaken in a state of imperfect and constantly changing knowledge. In particular, compatibility of the proposed system with concrete features of the current context of use may play a disproportionate role in determining whether the proposed system works effectively. These features can be identified by careful application of naturalistic data gathering and interpretive techniques (e.g. those from ethnographic practice). Applying naturalistic techniques in the context of goal refinement is the topic of a separate investigation (Potts and Hsi, 1996) and is not considered further here.

A .Example usage

We will now consider briefly a development scenario in which ScenIC is used in the requirements engineering of a new version of an existing system. This will help you make more sense out of the details in Section II. A more fully worked illustration is contained in Section III.

--- *t.b.w.*

I .Detailed Methodology Description

Section I presented an overview of ScenIC. In this section, we describe ScenIC in more procedural detail. Sub-section A deals with the format of inputs to the method, and Sub-section B with the format of outputs. The Sub-sections in C describe each of the ScenIC requirements refinements. Finally, in Sub-section D we revisit the discussion of evaluation, this time giving specific heuristics and techniques for evaluating requirements.

Figure <ScenIC> shows the position that ScenIC occupies in the MORALE framework for evolving systems.

A .Input formats

ScenIC makes use of three types of input information: mission-oriented requirements (externally, from the stakeholders), the previous requirements specification (externally, from the stakeholders) and architectural responsibilities (internally to MORALE, from "Discover Architecture").

1 .Format of mission-oriented requirements

Mission-oriented requirements are a type of unstructured information about the proposed version of the system. We make no assumptions about the form that they take and assume that they are expressed informally in natural language. It is even possible for non-critical systems that some mission-oriented requirements are obtained orally, not in writing.

In the refinement descriptions in Sub-section C, we assume that the mission-oriented requirements exist in writing and that they are expressed in natural language. The description of ScenIC procedures is essentially similar if requirements are obtained through interviews or meetings or if they are partially expressed in the form of informal diagrams.

Mission-oriented requirements are not essential for ScenIC. We obviously cannot analyze the requirements for a system if we have none. However, in some situations (e.g. the elaboration of requirements for consumer software products), it may be best to conduct product visualization using the ScenIC procedures and start with the goal identification activities. In this guidebook, we assume that mission-oriented requirements have already been written.

A typical example of an input mission-oriented requirement for a new version of a meeting scheduler might read as follows:

> "The system should allow resources such as rooms to be scheduled"

1 .Format of previous requirements specification

Existing requirements are requirements for the current system. They are of interest in ScenIC for several reasons. First, the requirements for the existing version of the system, if they have been kept up to date, may tell us much of value about the rationale for the functionality of the current system.

In addition to this, we may wish to use the existing requirements expression as a template for elaboration when expressing the requirements for the new version of the system. Not only is it unnecessary to rewrite all the requirements from scratch, but we are sure to miss something.

Finally, the existing requirements document may provide a form and organization that we can use for the new requirements.

It is not necessary to have previously documented requirements to use ScenIC. In the special case of requirements for a new system there is no previous version of the system. And in many cases the requirements for the previous system may have been poorly documented or be unusable for other reasons.

1 .Format of architectural responsibilities

Architectural responsibilities in the current version of the system may constrain the expression of new requirements. For this reason, it is useful to have a specification of the current architecture.

The representation of architectural information is described in the SAAM and SBF guidebooks. In ScenIC we are interested only in high-level features of architecture. For our purposes, an architectural specification can be considered as a set of component/action pairs. Components are pieces, subsystems or modules of the system. Actions are requirements-relevant operations, usually externally visible, that have well-defined triggering conditions, intervals of execution, preconditions and effects.

```
ArchSpec == (Comp X Action X Allocation X ActionDef)
```

where

- b Comp is a set of component names (including environmental actors).
- b Action is a set of action names.
- b Allocation is a set of component/action pairs in which each component in Comp and each action in Action are mentioned at least once, and each action in Action is allocated to exactly one component.
- b ActionDef is a set of definitions for actions, in which each action is defined exactly once. An action definition consists of an optional triggering condition, an optional precondition, a post-condition and an optional duration specification.

The simplified architectural sub-language of SBF that is to be used as the input to ScenIC will be defined later. However, the following illustrates how a fragment of an architectural specification for a meeting scheduler might be represented:

```
Action: Schedule-meeting

Allocation: Automated-scheduler

Trigger: OCCUR(Drop-dead-date) OR RECEIVE(Last-reply)

Assumption: Meeting-called AND Can-schedule-given-replies

Effect: NEW Time(Meeting)
```

Components and actions should be defined at compatible levels of detail. Although a full architectural specification may contain layers or hierarchies of architectural descriptions, in ScenIC these are flattened at whatever level of detail seems appropriate from the stakeholders' external viewpoint. We envisage an activity in SAAM that simplifies an SBF specification into a ScenIC architectural specification.

As with the requirements for the previous version of the system, there does not have to be architectural specification. In the special case of a new system, there obviously cannot be such a specification.

A .Output formats

The outputs of ScenIC consist of requirements specifications and requirements rationale.

1 .Format of requirements specification

A ScenIC output requirements specification is similar to an input architectural specification (see sub-section A.3 above) with the addition of quality requirements.

The quality requirements specification sub-language will be defined later. An illustration of how a requirement for a meeting scheduler might be specified is given below:

```
Action: Schedule-room

Allocation: Automated-scheduler

Trigger: DONE(Scheduled-meeting-time)

Assumption: Some-room-available

Effect: NEW Room(Meeting)

Performance: t.b.d.
```

1 .Format of requirements rationale

As explained in Section II, there are three kinds of requirements rationale in ScenIC: a trace from a requirement back to the goals that originated it, a scenario that illustrates the need for the requirement (or the lack of need for a plausible alternative), and a structured record of a discussion that gave rise to the requirement (or failed to give rise to a plausible alternative).

A trace rationale is illustrated by the following:

```
Action: Schedule-room

Rationale: FULFILLS "Improve room utilization" STAKEHOLDER Facilities-
manager
```

A scenario rationale is illustrated by the following:

```
Action: Schedule-room

Rationale: OTHERWISE SCENARIO "Nowhere to hold scheduled meeting" EX-
PLORED-BY November JAD team
```

Finally, a structured record of a discussion is illustrated by the following:

```
Action: Schedule-room

Rationale: RESOLVES ISSUE "Are additional resources needed to hold a
meeting, and if so can the

Automated-scheduler seize them?" POSTED-BY Analyst-Fred AGREED-BY No-
vember JAD team.
```

An ongoing activity within MORALE is concerned with integrating the rationale capture and scenario exploration aspects of ScenIC with the Win-Win requirements negotiation method (Boehm & In, 1996).

A .Refinements

ScenIC does not consist of a series of steps that the analyst must perform in order. Rather, it consists of a collection of expression/discussion/refinement cycles that collectively transform the initial information into an acceptable requirements specification.

1 .Goal identification

```
Start-corpus: unstructured information

End-corpus: Start-corpus + Goal + Actor
```

Typically, unstructured requirements information contains statements at many incompatible levels of abstraction. Even though a requirements statement sometimes is expressed at a detailed level, it is still important to take all the requirements information and rationally reconstruct a higher-level statement of mission-oriented goals.

A goal description consists of a goal name, a goal category and an optional textual description. For example,

```
GOAL-231:

NAME "Improve-resource-utilization"

CATEGORY Improvement

DESCRIPTION "Improve utilization of such resources as space and high-
value equipment."
```

There are three major categories of goals: improvement goals, maintenance goals, and achievement goals. The first two types of goals are pervasive goals, because they pervade the requirements for the system. The third type is easier to localize and therefore trace forward to the requirements, because the achievement of a goal is generally the responsibility of a single actor or component. We find it useful to indicate the category of a goal in its name by using a small repertoire of category-suggesting verbs.

Improvement goals are concerned with improvement in some business or mission factor. Category-suggesting verbs that can be used to name improvement goals include "improve", "speed up", "reduce", "increase", etc.

Maintenance goals are concerned with the continuance of some desirable mission-oriented condition or, conversely, the avoidance of some undesirable mission-oriented condition. Category-suggesting verbs that can be used to name maintenance goals include "keep", "preserve", "maintain", "prevent", "avoid", etc. Achievement goals are concerned with the achievement of some desirable mission-oriented condition, such as a physical state or the knowledge of some information. Category-suggesting verbs that can be used to name achievement goals include "make", "know", "become", "have", etc. We recommend avoiding names for achievement goals that suggest a specific action for fulfilling the goal. Thus "Know room's schedule" is a better name for a goal than "Look up room's schedule" because it denotes a condition to be achieved, not an action to be performed.

-- *How goals can be identified.*

Closely related to goal identification is actor identification. Actors include entities in the environment with which the system interacts and components of the system architecture that are assumed to exist in the proposed system (usually because they exist in the previous version). Actors are named and categorized. Naming and categorizing actors is generally unproblematic. Environmental entities exist in the discourse of stakeholders and are therefore easy to name; architectural components generally have names derived from the design documentation for the previous system. We recognize the following categories of actors: End-user, Indirect-user, Organization, System, Device, and Component, where all but Component exist in the system's environment. The category of an actor is relevant when looking for obstacles later.

1 .Goal expansion

-- *improvement goals into maintenance goals*

-- *maintenance goals into achievement goals*

-- *conjunctive sub-goals*

-- *disjunctive sub-goals*

-- *supporting goals*

-- *pervasive effects of improvement and maintenance goals*

-- *scenarios for normative cases*

-- *episodic scenarios*

1 .Obstacle identification

-- *achievement goals*

-- *obstacles not just negation of goals but causes thereof*

-- *many obstacles arise because of allocation decisions*

-- *categories of obstacle*

-- *actor failure obstacles*

-- *medium failure obstacles*

-- *resource contention obstacles*

-- *replicate confusion obstacles*

-- *risk analysis*

1 .Obstacle defense

-- *deciding whether to defend against obstacles*

-- *feasibility of defensive strategies*

-- *defending against actor and medium failure*

-- *defending against resource contention*

-- *defending against replicate confusion*

-- *role of scenarios in exploring defensive strategies*

1 .Obstacle mitigation

-- *deciding whether to mitigate obstacle effects*

-- *feasibility of mitigation strategies*

-- *mitigating actor and medium failure*

-- *mitigating resource contention*

-- *mitigating replicate confusion*

-- *role of scenarios in exploring mitigation strategies*

1 .Responsibility allocation

-- *allocation of goals to actors*

-- *allocation of actions to actors*

-- *responsibility allocation goes hand in hand with realization*

1 .Goal realization

-- *realization of achievement goals*

-- *realization of defensive strategies*

-- *realization of mitigation strategies*

-- *role of scenarios in allocation and realization*

1 .Scenario exploration

-- *not a refinement, but relevant to most refinements*

-- *scenarios as named situations*

-- *scripts*

-- *scripts as event traces*

-- *scripts as episodic stories*

-- *scenario coverage heuristics*

-- *recommendations about scenario details*

A .Evaluation

1 .Evaluation of goal accuracy

-- *techniques for assessing accuracy of goals from stakeholders*

-- *techniques for resolving inter-stakeholder goal conflicts*

1 .Evaluation of obstacle accuracy

-- *techniques for assessing accuracy of obstacles*

1 .Evaluation of requirements completeness

-- *techniques for assessing fulfillment of goals*

-- *techniques for assessing adequacy of obstacle defenses*

-- *techniques for assessing adequacy of obstacle mitigation*

1 .Evaluation of absence of gold plating

-- *techniques for isolating unnecessary requirements*

-- *techniques for isolating unnecessary defense*

-- *techniques for isolating unnecessary mitigation*

-- *techniques for isolating unnecessarily complex realizations of goals*

1 .Evaluation of goal realizations

-- *techniques for proposing realizations*

-- *the role of architecture analysis in realization*

-- *techniques for evaluating alternative realizations*

1 .Evaluation of scenario coverage

-- *measurement of goal coverage*

-- *measurement of defensive coverage*

-- *measurement of mitigation coverage*

-- *proposed measures of co-occurrence coverage*

1 .Evaluation of scenario detail

-- *instances in scenarios*

-- *three levels of detail*

-- *sharing of scenario episodes*

1 .Evaluation of contextual fit

-- *not here but refer to Potts & Hsi*

I .Tutorial Example

A .Introduction to example

-- Description of simple web browsing example

-- Envisionment of social context of example

A .Rational analysis of example

-- I: What are the high-level goals?

-- R: Goal identification

-- I: What subgoals and dependencies?

-- R: Expansion of goals and goal dependencies

-- I: What idealizations?

-- R: Identification of intrinsic obstacles

-- I: What obstacle defenses?

-- R: Identification of defensive goals

-- I: What alternative realizations?

-- R: Responsibility allocations

-- R: Operationalization of software-allocated goals

-- I: What exceptions?

-- R: Identification of realization obstacles

-- I: What realization obstacle defenses?

-- I: What realization obstacle mitigation strategies?

-- R: Identification of realization-based defensive and mitigation goals

-- R: Identification of realization-based defensive and mitigation operations

A .Concrete envisionment of example

-- I: What are the normal-case uses?

-- R: Normative scenarios

-- I: What is supposed to happen?

-- R: Scripts for normative scenarios

-- I: What are the exceptional cases?

-- R: Single-obstacle and combined-obstacle scenarios

-- I: What

I .Glossary

-- Part of this should be common across the guidebooks

Allocated requirement: A requirement that has been allocated to a total system (usually, automated system) component.

Achievement goal: A goal that is described in terms of a desiredstate of affairs that does not yet hold.

Architect: A developer who is responsible for analyzing the requirements for the automated system, and proposing and analyzing a solution architecture or modifications to an existing architecture. (Note: This definition cuts across traditional distinctions between systems and software engineers and between analysts and software designers.)

Architectural constraint: A fact about the architecture of the legacy system that probably limits the requirements and putative architecture of the target system.

Architectural scenario: A scenario in which the roles are users and automated system components.

Architectural script: A script for an architectural scenario.

Architecture: An organization of automated system components that meets the requirements for the automated system and the affordances provided by its environment.

Automated system: A system in which one or more components are devices, software configuration items or other automated systems.

Canonical refinement: A standard refinement pattern.

Component: A part of a system. A component may be regarded as a system in its own right or as an atomic agent.

Context: Concrete aspects of the system's environment that affect its likely success.

Customer: A stakeholder who represents the organization that is intended to benefit most from the total system. (Note: the contractual client may not be a customer in this sense).

Defensive goal or requirement: A goal or requirement that exists to reduce the likelihood of an obstacle occurring. The limiting case is a preventative goal or requirement.

Description: The activity of representing and documenting some aspect of a system.

Design Journal: The repository containing the current requirements and architecture of the target system, the history of refinements that led to it, and the rationale.

Developer: A stakeholder who is a member of an organization responsible for the development and evolution of the automated system.

Discussion: The activity of discussing, analyzing or reviewing a description.

Elaboration: A refinement that converts an idealized or simplified goal or requirement into one that accounts for obstacles in the target system or its environment.

Environment: The totality of things not in a system that interact with or influence it.

Expansion: A refinement that converts a goal into interdependent subgoals.

Fact: An assumed or verified statement about the world that affects the system requirements and which the system must be designed to accommodate but which is not a requirement on the system itself.

Goal: A desired purpose of a system or component.

Goal web: A description (probably graphical) of a collection of goals and their dependencies.

Human-Activity System (HAS): A system in which the components are people, teams or organizations. These system components may use automated tools to help them meet their goals, but the system does not contain such tools as principal components.

IBIS: Issue-based IS. A framework for conducting and representing design deliberations and for recording design rationale.

Inquiry Cycle: A collection of three kinds of activity involved in designing and planning a system, viz.: description, discussion and refinement.

Legacy system: An existing software system that is determined by a customer to be in need of improvement.

Maintenance goal: A goal described in terms of a desired state of affairs that already holds.

Mission-oriented goals & requirements: Goals and requirements that stem from the mission or context in which the target system is to be used, rather than those arising out of purely technical inadequacies of the legacy system.

Mission-oriented scenario: A scenario in which the structure of the scenario is related to the goals of the system or components in question.

Mission-oriented script: A script for a mission-oriented scenario.

Mitigation goal or requirement: A goal or requirement that exists to reduce the harm caused by an obstacle when it occurs. The limiting case is a recovery goal or requirement.

Obstacle: A state of affairs that can impede the fulfilment of a goal.

Operational requirement: A requirement that has been allocated to a software component and that is described in enough detail to be implemented in source code.

Owner: A stakeholder who defines the success or otherwise of the total system and has the authority to cancel its development.

Procedural representation: A representation of requirements or system architecture that emphasizes temporal behavior rather than system purpose.

Putative architecture: An architecture put forward by an architect as a viable design to meet the mission-oriented requirements or as an explanatory structure for the source code of a legacy system.

Quality goal: A goal described in terms of a trajectory of improvement in some criterion.

QOC: Questions/options/criteria. A framework for conducting and representing design deliberations and for recording design rationale.

Rationale: The reason or reasons for a decision, or the reasoning surrounding an issue that has not yet been decided.

Realization: The refinement process that converts goals into allocated operational requirements.

Refinement: The revision of one or more descriptions as a result of discussion. Refinements include the creation of new descriptions that are either subordinate to existing descriptions or otherwise dependent on them.

Requirement: A property of an automated system needed by a customer stakeholder.

Responsibility: An allocation of a goal or an operational requirement to a component.

SAAM: Software architecture assessment methodology(?). A method that uses scenarios to evaluate putative architectures.

SBF: Structure/Behavior/Function architecture modeling framework.

Scenario: An imagined set of concrete system behaviors in response to a state of affairs.

Script: A description of a scenario in which the component's actions are represented in time-ordered sequence.

Software requirement: A requirement that has been allocated to a software system or component.

Software system: A system that is to be implemented in software components.

Stakeholder: A person, group of people or with a stake in the success of the total system.

Synchronized refinement: An expectation-driven method for analyzing source code and documentation to recover a software architecture.

System: A collection of active components whose coordinated behavior meets a set of goals.

Target system: A system that is to be implemented in software and hardware components.

Teleological representation: A representation of requirements or scenarios that emphasizes purpose, goals or objectives, rather than process.

Total System: A large, complex system including one or more human-activity systems and one or more automated systems.

User: A stakeholder who is a member of a human-activity system in the environment of the automated system who directly interacts with the automated system. Direct interaction may take the form of operating a computer worksation, but could equally involve interaction with ubiquitous or embedded automated system components or the consumption of reports or other information digests generated by the automated system. A user may perform a collection of activities that have no direct counterpart in the human-activity system that exists prior to the implementation of the automated system.

I .Annotated References to the Literature

How this method relates to other methods.

Discriminators

for each of:

b -- Process models for requirements analysis

b -- Coherent methodologies (SADT, SSADM,...)

b -- Goal refinement methods (including SSM)

b -- Scenario-based design

b -- Design rationale models

Links to further information

-- *Previous papers*