Stimulus Response Requirements Specification Technique

Kendra M.L. Cooper*, Jeffrey J. Joyce+, Mabo R. Ito*
+Hughes Aircraft of Canada Limited
*The University of British Columbia

kendrac@ee.ubc.ca, jjoyce@ccgate.hac.com, mito@ee.ubc.ca

Abstract

A new requirements specification technique, Structured Stimulus Response, is presented in this paper. This technique is suitable for the specification of large, software intensive systems with complex data requirements. The Structured Stimulus Response technique may be seen as a synthesis of Structured Analysis and Use Case techniques. From Structured Analysis, this new technique inherits the concept of using a data dictionary to describe the vocabulary for the requirements specification as well as using various structural and notational constraints that may be checked automatically by a software tool. From the Use Case technique, this work inherits the concept of partitioning the specification from the user's perspective. The Structured Stimulus Response technique is presented in this work as a semi-formal requirement specification technique that may potentially serve as a framework to support the selective application of more formal techniques.

1. Introduction

There is a great deal of innovative, interesting and potentially useful research being applied to the development of formal specification techniques based on mathematical notations [Bar97, Hal97, Rus97, Sai96]. Such techniques may be applicable for the description of selected aspects of a large specification, for example, to accurately specify some particularly complex decision logic. However, it is doubtful that a technique based on a notation such as Z [Spi88] could be practically applied comprehensively and uniformly in the specification of the requirements for a large system. Instead of trying to replace natural language with mathematical symbols, the technique described in this paper is the result of imposing a set of constraints on the use of natural language to specify requirements.

The new technique, Structured Stimulus Response (SSR), is intended primarily for the specification of the software requirements for large, software intensive systems that have complex data requirements. This technique shares common elements with both the Structured Analysis (SA) and the Use Case techniques. However, the SSR technique contrasts significantly from these other two techniques in its use of mechanisms to support a "blackbox" description of the requirements, such as a declarative style of requirements specification rather than indirectly specifying requirements by means of building a model.

SSR originates from a technique developed by Hughes Aircraft of Canada Limited (HACL) to specify the software requirements for the Canadian Automated Air Traffic System (CAATS) [Pai93]. As presented here, the SSR technique is a generalization and refinement of the one developed specifically for the CAATS project at HACL. SSR is generalized by eliminating the domain specific details of an air traffic control system, which makes the new technique suitable for a variety of applications. The refinements of the HACL technique are influenced by experience in two areas. The first is the lessons learned from applying the technique on the CAATS and the Canadian Military Automated Air Traffic System (MAATS) projects at HACL. The second area of influence is the familiarity and experience with formal specification techniques.

Although not a formal specification technique, the work presented in this paper is influenced by experience with formal methods. The influence, however, does not extend to the incorporation of mathematical symbols in the technique. The result is that the new technique is not limited for use only by formal methods experts. In its current form, the new technique may serve as a framework for the selective use of formal specification techniques. The use of a formal specification technique may be warranted for the description of safety or security critical functionality in a system, for example. Extensions to the work, which formalize the technique, are also presented in this paper.

The organization of this paper is as follows. The motivation for developing a new requirement specification technique is discussed in Section 2. The distinguishing characteristics and structure of the SSR specification technique are presented in Sections 3 and 4. Sections 5 and 6 compare and contrast the SSR technique with the well established SA and Jacobson's Use Case techniques [Dem78, Jac92]. Section 7 presents the potential for automated checking of and the generation of test cases from a specification written in the SSR technique. Extensions which formalize the SSR technique are discussed in Section 8. The conclusions and future work are presented in Sections 9 and 10.

2. Motivation

Developing a requirement specification for large, software intensive systems with complex data requirements is challenging with the techniques that are currently available. For the purposes of this work, a system is considered large when it has at least several thousand distinct requirements and is documented by a team of 20 or more authors over the course of a year or more. The data complexity may be characterized by a large number of distinct data definitions which may be related by both composition and association relationships, and a large number of stimuli and responses from/to external sources/sinks.

Large, data complex systems may have several hundred distinct data definitions which are related in a data dictionary. The data definitions may be related by composition or association relationships. The composition relationship may be described in Backus-Naur Form (BNF) and is used in the SA technique to achieve a higher level of abstraction in the data definitions. The association relationship is more general than a composition relationship. For example, in a library system, there may be an inherent relationship between a borrower and a book they have signed out. BNF is not well suited for describing this type of relationship.

The large number and variety of stimuli and responses for a system may have requirements that are triggered by a group of stimuli or generate a group of responses. This may occur when the groups consist of both local and external stimuli or responses. Since the stimuli and responses are arriving from or being sent to external interfaces, the format of the message types may be different for each member. The requirement, however, for processing the stimuli or generating the responses may be exactly the same. In systems with a large number of these stimuli and responses, it would be convenient to group them, such that the requirement is only described once. Introducing these groups, however, also needs a supporting mechanism to relate which stimulus belongs with which response. Although the SA technique may use the BNF descriptions to group stimuli and responses, for example, there is no mechanism available to match up the individual stimulus response pairs that need to be specified.

For such large systems, the usefulness as a communication tool, maintainability, consistency and conciseness are important characteristics of a requirement specification to consider. The partitioning of the requirement specification should provide the customer with an overview of the capabilities of the system, as seen from the end users point of view. A maintainable specification is described using techniques that support a “blackbox” description of the system. This helps to avoid describing internal processing which may constrain the design or create additional work to interpret the requirements for the system. Writing conventions in a specification technique promote writing the requirements in a consistent style. This is of particular interest when a large number of authors are developing the specification over a lengthy period of time. The writing constraints also promote describing a common idea with the same phrasing and vocabulary in different parts of the specification. To encourage a concise description of the requirements, a mechanism to group inputs and outputs that trigger the same requirements and are generated by the same requirements is useful for systems with a large number of unique input and outputs. This grouping allows the requirements to be written once, rather than for each member in the group, making the requirements concise.

Currently available techniques, such as SA and Use Case techniques, do not support all of the desired characteristics for describing the large systems with data complexity. SA supports writing constraints through the use of a data dictionary, however it does not promote a “blackbox” view of the system. Use Cases support partitioning the specification from the user’s point of view, but does not provide writing constraints to promote consistency across the requirement specification document. A new technique which draws from both the SA and Use Case techniques may be useful for the description of the large systems under consideration.

3. Distinguishing Characteristics of the SSR Technique

Distinguishing characteristics in the SSR requirements technique include having an “outside-in” partitioning of the requirements, externally visible stimuli and responses, a declarative notational style, a mechanism to group I/O, and employing writing constraints in the technique. These characteristics are discussed below and summarized in Table 3.

3.1. Partitioning of Requirements

Two broad categories for partitioning the software requirements for a system are “top-down” and “outside-in”. The “top-down” decomposition is implementation dependent, as an arbitrary choice of possible decompositions is made [War85].The “outside-in” approach uses the stimuli and responses to and from the system to partition the software requirements. The sources and sinks may be either human or other systems. An advantage of employing the “outside-in” partitioning is that the resulting specification is well suited to use as a communication tool with the end user of the system. The specification is organized around the input and outputs of the system, which corresponds to a user's point of view.

The SSR technique applies the "outside-in" partitioning to structure the software requirement specification document. As a result, the documentation is straightforward to review, especially with respect to the scope of the system's capabilities.

3.2. Visibility of Stimuli and Responses

A "blackbox" description of software requirements describes the behaviour of the system in terms of its external stimuli (inputs) and external responses (outputs). In general, every requirement is specified in terms of a relationship between an externally generated stimulus and an externally visible response. The advantages of using a "blackbox" approach for describing requirements include minimizing the potential for including internal design details in the specification and maximizing the suitability of the specification in testing the system's software. Discouraging the inclusion of design details in the specification decreases the likelihood of overly constraining the design and makes the specification simpler to maintain as the design details may change as the project develops. Minimizing the software's internal processing descriptions simplifies the development of "blackbox" test cases as the test engineers do not have to derive the requirements based test cases from descriptions of internal processing.

The SSR technique is based on describing externally visible stimuli and responses. The requirements are typically described in terms of a direct relationship between a group of externally generated stimuli and a group of externally visible responses. Because of the "blackbox" approach, a specification written in the SSR technique is well suited for use as a working document by the system integration test group.

3.3. Notational Style

Two basic categories of notations include model based and declarative approaches. Model based approaches describe the requirements indirectly, through the development of a representation, or model, of the requirements. The declarative style, on the other hand, does not involve developing a model, and states the requirements directly. A declarative style discourages the inclusion of internal design details in the specification.

The SSR technique uses a declarative style to minimize the inclusion of internal processing (design details) in the specification. In addition to simplifying the interpretation of the requirements, the discouragement of including design details also simplifies the maintenance of the specification.

3.4. Grouping I/O

In terms of maintainability and conciseness, it is convenient to group inputs (stimuli) and outputs (responses) and to be able to refer to these groups by name for systems with a large number of stimuli and responses. For example, if six different stimuli are described that trigger a single requirement, then its group name is used in the SSR technique to refer to the list in a single description of the requirement. The alternative is to repeat the requirement for each of the six different stimuli that triggers the processing requirement.

The SSR technique provides local stimulus names and local response names to represent a list of stimuli (source name, message type pairs) and responses (destination name, message type pairs) respectively. The use of group names supports the development of a specification which is concise and simple to maintain. The specification is concise because a requirement which responds to a list of stimuli and generates a list of responses is only described as a single requirement, rather than describing one requirement for each of the individual stimulus, response events. In terms of maintainability, the specification is simpler to maintain, because the details of the stimuli and the responses are documented in a single place in the specification. For large systems with complex data requirements, the abstraction of the I/O has the advantages of supporting a concise and maintainable specification.

When groupings of I/O are used in a technique, there needs to be a mechanism to pair the appropriate stimulus and response. SSR supports this pairing using writing conventions for the stimulus and response message type names.

3.5. Writing Constraints

Writing constraints in a specification technique, including template phrasing and a reduced vocabulary, are mechanisms to improve the consistency of specifications written by different authors.

Template Phrasing. Template phrasing is one mechanism used to improve the consistency of the requirement specification document. It provides a standard way of describing a problem that is encountered repeatedly. A template based technique is described in [Hen80]. An advantage of using a template based technique is that the authors do not need to reinvent the phrasing to use on descriptions that are re-used, for example the receipt of a stimulus (input). When all of the authors are using the template phrasing, the specification is consistent and, as a result, straightforward to review.

The SSR technique uses template phrasing within each of the sections to improve the consistency of the specification, written by a large group of authors, as a whole. For example, one of the standard template phrases for a requirement statement is:

> **Upon receipt of a [** local stimulus name **], (if** condition**, then) the system shall** respond_with **a [** local response name **] .**

The bolded words or characters represent the standard part of the phrase. The plain type words are the parts of the sentence that are filled in by the author. The parts that are filled in by the author are also subject to the reduced vocabulary constraints described below.

Reduced Vocabulary. Reducing the vocabulary is another mechanism to improve the consistency of the requirement specification document. A reduced vocabulary means that authors specifying the requirements are restricted in their choice of words to describe the behaviour of the system. A well maintained data model, for example, is one mechanism to reduce the variety of terms used in a project. The data model should only describe unique data items that are used in the requirement specification. Ideally, items are re-used by different authors, rather than each author creating duplicate entries in the data model. This improves the consistency of the specification because all authors use the same terminology to describe the data in the requirements.

The SSR technique uses a local data dictionary concept, in which only the data that is used by a specification unit, or task as viewed by an end user, is described. From these local dictionaries, the project dictionary may be extracted. For large projects, a centrally maintained data dictionary is difficult to maintain, such that the entries are necessary and complete. Contributing factors to this problem include the large number of authors requesting updates to the data dictionary and the large size of the repository.

In addition to the local data dictionary, the SSR technique also restricts the verbs that are used in the responses to a set of five action verbs. Again, the purpose of the reduced vocabulary is to ensure consistency among the requirements written by different authors. A second benefit of this writing constraint is that it discourages the inclusion of computer human interface (CHI) details. The verbs do not include CHI related terminology such as “display” or “enter”. The result is that the system is described in terms of a virtual environment, which isolates the requirements from the CHI design and the interface technology used to implement the CHI design. As CHI requirements may change frequently, the removal of the CHI details from the specification makes the document simpler to maintain.

3.6. Other Useful Characteristics of SSR

Tabular Descriptions. Tabular formats for describing complex, logical conditions are described in the literature [Par92, Lev94, Day97]. The advantages of providing a tabular format for describing logic include conciseness and readability.

The format of the tables supported in the SSR technique are a variation of an AND/OR table [Day97]. Predicate tables have the following format:

Label	Case 1	Case 2	...	Case n
Label 1	predicate	predicate	...	predicate
Label 2	predicate	predicate	...	predicate
...	...	...	...	predicate
Label m	predicate	predicate	predicate	predicate
Title				

The label of each row in the predicate table is an expression. The cells of each row are predicates that are applied to the row’s label. Each column represents a single case, where the conjunction of the cells in the column is true. Any other cases not described in the table explicitly are assumed to be false. The table represents an “if, else if” structure. The predicate table’s name is documented in the last row of the table.

The inclusion of a table format for describing complex conditions makes the SSR technique flexible. The specification for requirements such as aircraft separation rules in air traffic control systems have been concisely described using this tabular format [Day97].

4. Structure of Stimulus Response Requirements Specification Unit

The specification unit is highly structured into the following sections: title, overview, stimuli, responses, requirements, declarations, and performance (refer to Figure 1). The purpose of

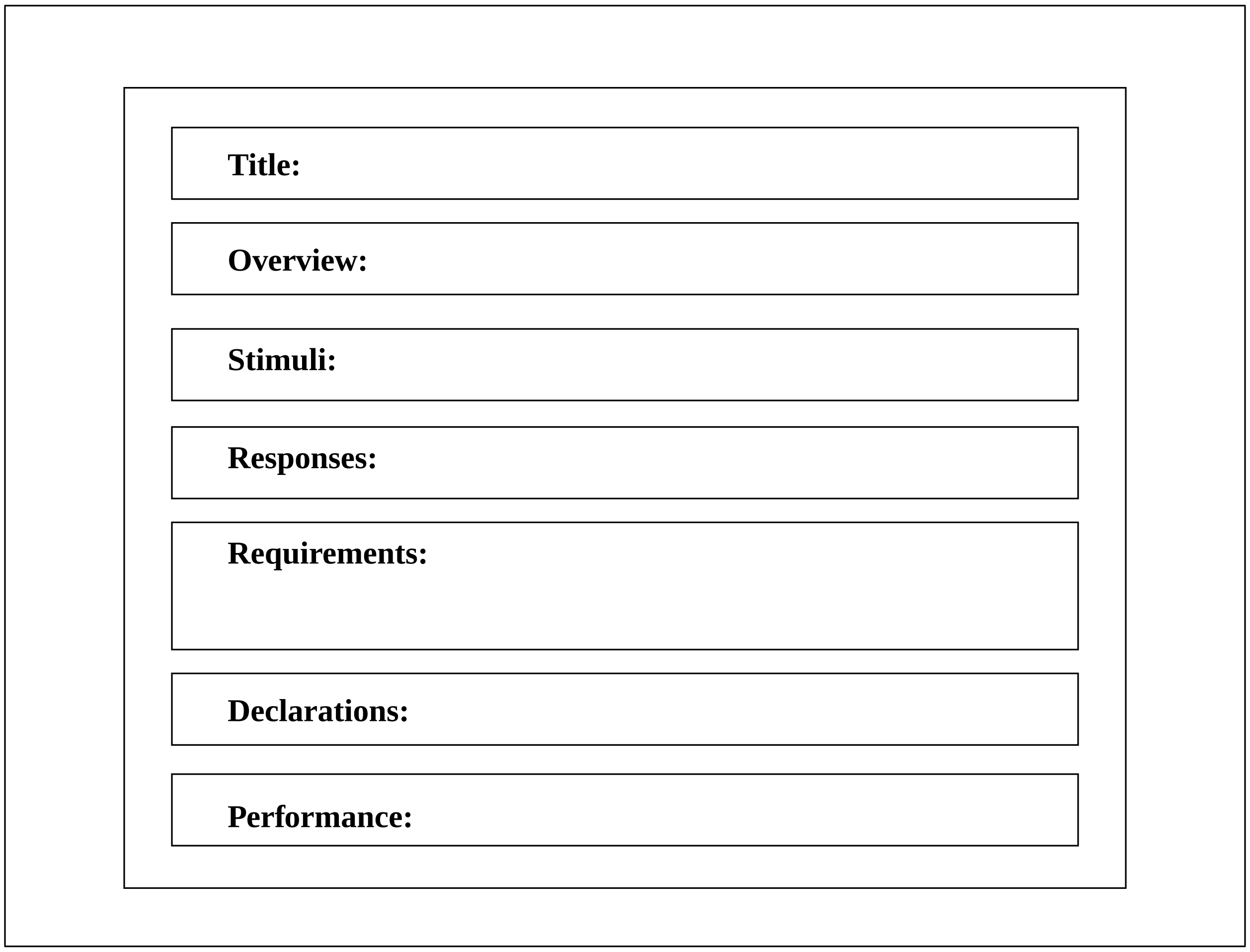

Figure 1: Stimulus Response Specification Unit Structure

each section is described below and illustrated using a library system example.

Having a highly structured specification unit provides a common framework for the specifiers, reviewers, and users of the specification unit to follow. This increases the consistency of the specification units in a SRS and makes the specification units simpler to write and review.

4.1. Title Section.

The title section uniquely identifies the specification unit with a meaningful name. It provides a quick overview of what the specification unit does. For example, a library system has requirements that users can search the catalogue for an item. The title of the specification unit describing these requirements is:

Title:
\ Search Catalogue \

The customer may use the list of titles in a requirement specification as summary of the capabilities the specification describes.

4.2. Overview Section.

The overview section provides a high level description of the processing the specification unit provides. The section summarizes the acceptance and rejection processing for the specification unit. For example, the overview section for the \ Search Catalogue \ specification unit is:

Overview:
The Search Catalogue specification unit describes the processing performed when an operator or external library requests to search for an item. The request may search for an item using the author name, title, call number, or subject.

4.3. Stimuli Section.

The stimuli section describes all of the events that trigger processing in the specification unit. Each stimulus is composed of a source name and a message type. The stimuli are grouped under local names such that each group of stimuli is processed in the same way in the requirements section of the specification unit. The local stimulus name provides a convenient shorthand notation for use in the requirements section to refer to the stimuli in the group.The advantage of these group names in large systems is that the processing for multiple stimuli may be exactly the same and only need to be described once. For example, the stimuli section for the \ Search Catalogue \ specification unit contains:

Stimuli:

1) The library system shall satisfy the requirements described below upon receipt of a [search request] from:

a) Operator <search request>;

b) Remote Operator <search request>;

c) VPL <search request>;

d) SFU <search request>;

e) UVIC <search request>;

f) EPL <search request>.

4.4. Responses Section.

The responses section describes all of the outputs that are the result of processing done in the specification unit. Each response is composed of a destination name and a message type. The events are grouped under local names, such that each group of responses is generated in the same way in the requirements section of the specification unit. The local response name provides a convenient shorthand notation for use in the requirements section to refer to the responses in the group. In large systems, the responses that are generated in response to a stimulus may be exactly the same, and only need to be described once using a group name. For example, the responses section for the \ Search Catalogue \ specification unit contains:

Responses:

1) As specified by the requirements described below, the library system shall return a [search response] to:

a) Operator <search response>;

b) Remote Operator <search response>;

c) VPL <search response>;

d) SFU <search response>;

e) UVIC <search response>;

f) EPL <search response>.

4.5. Requirements Section.

The requirements section describes what processing the specification is responsible for when it is triggered by different groups of stimuli. For each stimulus, the minimum requirements include a response of some type. A requirement refers to its stimuli by the local stimulus name. Each requirement generates one or more responses, which are referred to by the local response names. For example, the requirements section for the \ Search Catalogue \ specification unit contains:

Requirements:
1) Upon receipt of a [search request], if the [search request] is not rejected, then the library system shall return an [acceptance].

4.6. Declaration Section.

The declaration section provides the underlying foundation for the specification unit in terms of the data it requires and the relationships among the data. The data declarations are the "glue" that tie together the requirement specification processing requirements, and are a critical component of the specification.

To provide structure to the declaration section, seven sub-sections are defined. These are summarized in Table 2.

Declaration Sub-section	Purpose
Type Names	Declare the data types needed
Type Aliases	Declare alternate names for the data types
Static Associations	Declare relationships among the data types
Constants	Declare named objects of a specific data type
Adaptation Data	Declare named objects of a particular type that may be modified on a per installation basis
Stimulus and Response Components	Declare the source names, destination names, and message types for the stimuli and responses
Other Specification Units Used	Declare the other specification name titles referred to for re-using requirement descriptions

Table 2: The Purpose of the Declaration Sub-sections

For example, type names used in the \ Search Catalogue \ include:

Declarations:

Type Names:
1) :<search key>.

Constants used include:

Constants:
1) <author name>:<search key>.
2) <title>:<search key>.
3) <call number>:<search key>.
4) <subject>:<search key>.

4.7. Performance Section.

The performance section describes the response time categories for each stimulus-response pair in the specification unit. For example, the performance section of the \Search Catalogue\ contains:

The mean amount of time to process searching for an item is 3.5 seconds.
The maximum amount of time to process searching for an item is 6.5 seconds.

5. Comparison and Contrast with Structured Analysis

SA is a well established technique used to describe a functional model of a system [Dem78]. It is a model based technique that uses a “top-down”, functional decomposition. The SA technique combines both graphic diagrams and textual descriptions to model the requirements.

The characteristics described in section 3 are used as a basis to compare and contrast SA with SSR in this section. The results are summarized in Table 3.

Goal for the Specification	Characteristic to Support the Goal	SSR	SA	Use Case
Communication Tool with Customer’s End User	“Outside-in” partitioning	yes	no	yes
Maintainable Specification	Externally visible stimuli, responses	yes	mixed	mixed
Maintainable Specification	Declarative notational style	yes	no	no
Concise Specification	Grouping I/O	yes	no	no
Consistent Specification	Template phrasing	yes	no	no
Consistent Specification	Reduced vocabulary	yes	yes	no

Table 3: Characteristics of SSR, SA, and Use Case Techniques

5.1. Partitioning of Requirements

The SA technique partitions requirements using a functional, “top-down” technique. The basic partitioning tool in SA is the data flow diagram. The partitioning is achieved by decomposing functional transformation centers on a diagram until the processing for a single center can be described in one page or less of text. The lowest level of partitioning may not provide a good overview of the capabilities of the system, from a customer’s point of view. In contrast, the SSR technique uses an “outside-in” partitioning which partitions the specification from an external view, and is useful as a communication tool to use with the customer.

5.2. Visibility of Stimuli and Responses

The mapping between the external sources and sinks of stimuli and responses is summarized on the context diagram of the SA model [Dem78]. At the context level, the stimuli and responses are all externally visible. As the model is decomposed, however, and more details are added to describe the system, internally visible flows are added to the model. At the primitive, process specification level, the model of the requirements may not be partitioned such that every process specification is responding to externally visible stimuli and generating externally visible responses. Stimuli and responses described at this level which are not externally visible represent internal processing, or design details. The SSR technique, however, partitions a specification such that every specification unit is responding to and generating externally visible events.

5.3. Grouping I/O

The SA technique supports the description of data flows in the data dictionary using the BNF notation. Stimuli and responses may be grouped together using the selection notation in BNF, however, there are no mechanisms in the SA technique to pair up appropriate stimulus response pairs. In the process specification, the stimulus response pairs are described separately in the requirements. In contrast, the SSR technique supports grouping I/O in the stimuli and responses sections and pairing the events.

5.4. Writing Constraints

Writing constraints supported in the SA technique include enforcing a restricted vocabulary and a restricted syntax.The structured English consists only of imperative verbs, terms defined in the data dictionary, and reserved words for logic formulation. The data dictionary promotes the re-use of data elements in the specification, as does the SSR technique. The significant difference of the SA technique with respect to the reduced vocabulary used in SSR is that the SA technique does not restrict which verbs may be used. The syntax of the sentences permitted in the SA structured English include simple declarative sentences, decision constructs, and repetition constructs. This syntax is much more flexible than a template phrasing approach. The flexibility may lead to more differences among the authors writing the specifications and reduce the consistency of the document.

5.5. Notational Style

The SA technique is a modelling technique. The model is based on transforming data in the system and is described using a set of data flow diagrams and a data dictionary. The model is built as a decomposition of the context diagram, which describes the boundary of the system and the external sources and sinks of data, into a set of data flow diagrams. The data flow diagrams are composed of transformation centers, data flows, and data stores. As the model is decomposed, additional details are added to describe the system, by describing transformation centers and additional internal flows and stores. In contrast with the declarative style of the SSR technique, the SA technique does not discourage the description of internal processing details. Subsequently, the SA model of the requirements may take additional time to interpret by its users and inadvertently constrain the design.

6. Comparison and Contrast with Use Cases

Use Cases are a newer, popular technique for describing requirements that uses natural language to describe a model of the user requirements. Each Use Case describes a way to use the system. The set of Use Cases for a system represents everything users can do with the system [Jac94]. Use Cases are compared and contrasted with the SSR technique in this section. The results are summarized in Table 3.

6.1. Partitioning of Requirements

The Use Case technique partitions the software requirements from the user’s perspective. Each task the actor needs to do becomes a Use Case in the software requirement specification.

The requirements specification is straightforward to review by the customer, as the partitioning clearly describes the scope of the system. This is the same partitioning mechanism used in the SSR technique.

6.2. Visibility of Stimuli and Responses

The Use Case technique combines two views of the system into the Use Case descriptions: the user's point of view and the developer's point of view. The result is a mixed "blackbox" and "whitebox" description of software requirements. The "blackbox" component of the Use Case describes the dialogue between the user and the system. The "whitebox" components describe internal processing which may constrain the design and/or implementation.

6.3. Grouping I/O

The Use Case technique does not provide a mechanism to group I/O, as the SSR technique does. In the Use Case technique, each stimulus is described separately in the Use Case description. The SSR technique has an advantage for describing systems with a large number and variety of stimuli and responses, in comparison to the Use Case technique.

6.4. Writing Constraints

The Use Case technique does not provide writing constraints for template phrasing or for using a reduced vocabulary as the SSR technique does. The Use Case technique is extremely flexible, however, it may be difficult to use on projects with a large number of authors. Ensuring a consistent style across the document without writing constraints may be difficult, as each author would have their own style and that style is likely to evolve over the duration of writing the specifications.

6.5. Notational Style

A Use Case is not described using a declarative style. Instead, a Use Case can be modelled as a state machine [Jac94]. An instance of a Use Case traverses states of this machine during its lifetime. A stimulus that is received from an actor causes the Use Case to leave its current state and perform a transaction. The transaction performed depends on the state-stimulus combination. The transaction includes the manipulation of internal attributes of the Use Case and externally visible outputs to actors. The transaction is finished when the Use Case has entered a state (possibly the same one) and is awaiting another stimulus from an actor.

In contrast to the SSR technique, which has a declarative style, the Use Case technique has the authors building a model of the requirements. This modelling technique does not discourage the description of internal processing.

7. Potential for Automated Checking and Generation of Test Cases

The SSR technique uses writing constraints that support scanning and parsing the natural language text for data dictionary elements (enclosed within '<' and '>') and local stimulus and response names (enclosed within '[' and ']'). With scanning and parsing support, simple checks may be automated including:

- is a local stimulus name declared in the stimuli section before being used in the requirements

section?
- is each local stimulus name used one or more times in the requirements section?
- is each stimulus message used in the stimuli section declared in the declaration section?
- is a local response name declared in the responses section before being used in the requirements section?
- is each local response name used one or more times in requirements section?
- is each response message used in the responses section declared in the declaration section?
- is each data element used in the requirements section declared in the declaration section?

8. Formalization

A reduced vocabulary and decentralized data dictionary provide the infrastructure to support formalization of the SSR technique. There are two possible directions to take when formalizing the requirements (refer to Figure 3). The first option involves interleaving a formal notation and natural language descriptions of the requirements. The second option involves formalizing the template phrases.

The first option structures the requirements such that a Z like modelling technique is supported. The formalized version interleaves the Z schema with informal natural language. An advantage of this option is that a reader may choose which sections to read (either the formal or the informal). Significant drawbacks to this option include the difficulty in maintaining consistency between the informal and formal components. The maintenance is required on two specifications, rather than just one. Secondly, if or when the two documents are out of synchronization, the question which must be answered is which of the two specifications is the *real* specification of the requirements.

The second option extends the template phrasing concept such that the entire specification is written using a notation that reads like English, has a defined syntax and semantics, and is completely machine scannable and parseable. This makes the specification amenable to automated scanning, parsing, typechecking, analysis, and transformation. Automated checking and analysis may be used to reduce the cost of reviewing specification documents. The potential to automate the generation of test cases by transforming requirements specifications is described in [Don97].

9. Conclusions

The new SSR requirements specification technique is designed for describing the requirements of large, software intensive systems with complex data requirements. The SSR technique is intended to describe requirements specification documents which are useful as a communication tool with the customer, concise, and maintainable.

To support the specification of the large systems, the SSR technique draws ideas from the well established SA and Use Case techniques and contributes new ideas as well. SSR borrows the “outside in” partitioning of the requirement specification document from the Use Case technique and the use of a data dictionary from the SA technique. To these ideas, grouping the I/O, an optional table format, and multiple levels of writing constraints are added.

Multiple levels of constraints are used in the SSR technique to improve the consistency of

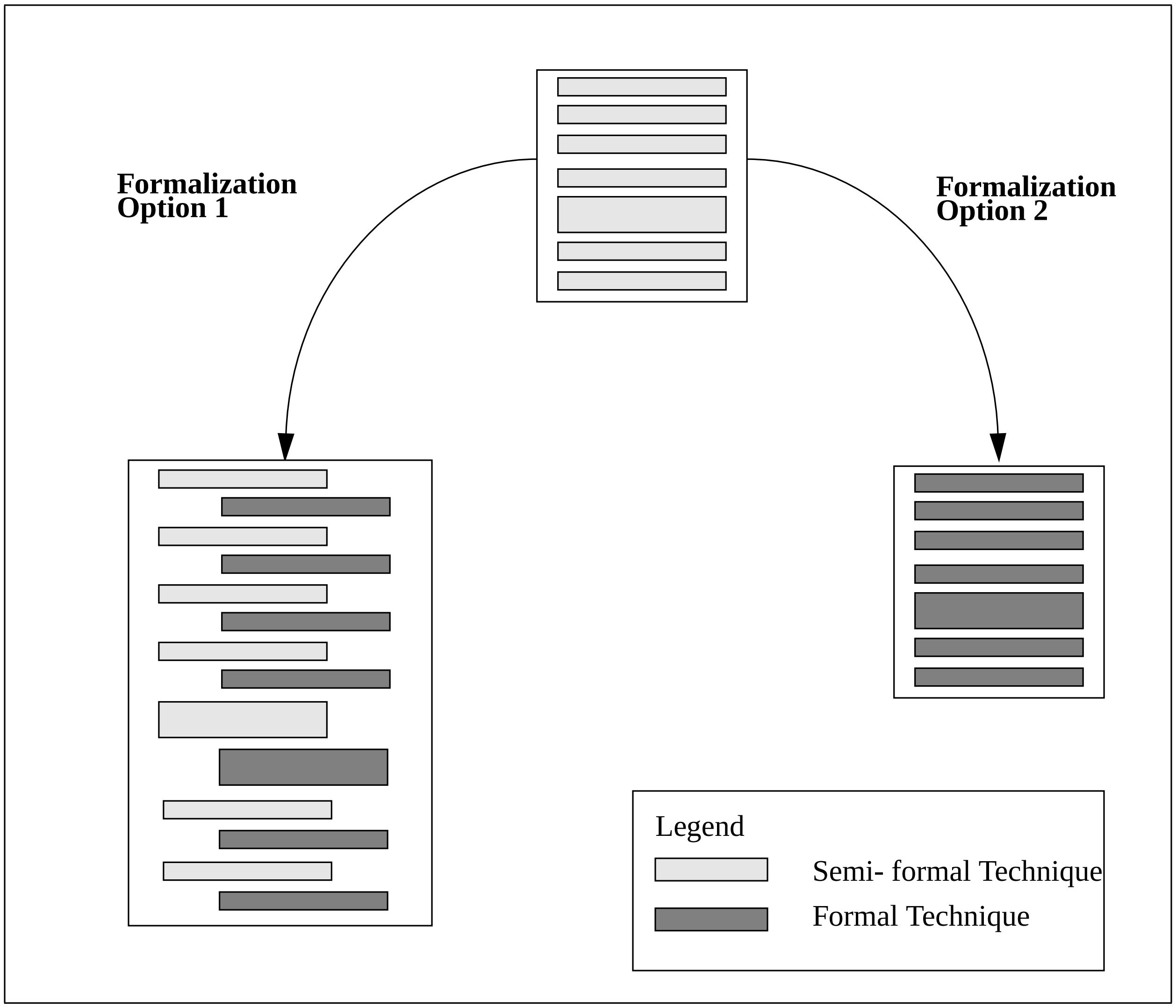

Figure 3: SSR Formalization Options

the specifications written by multiple authors. Improving the consistency of the specification units reduces writing, reviewing, and maintenance time for the specification.

Because of the declarative style used in the SSR technique, it is not suitable for describing requirements with state based behaviour. For example, process control systems are well suited for the SA or Use Case techniques, but not the SSR technique.

The automated scanning, parsing, and analysis checks are limited in the SSR technique. A formalized version of the SSR technique would allow typechecking and more sophisticated analysis checks to be run on the specification. This may improve the consistency of the requirements while reducing the cost of reviewing the document. The cost of using a formalized technique may include additional training time in the specification technique and the tools that support it.

10. Future Work

Future work in this research is focused on formalizing the SSR technique. The formalization work is proposed to use the second option described in Section 8. The template phrasing concept is to have a defined syntax and semantics, yet still read like English. After defining the language, tool support to scan, parse, typecheck, analyze, and transform a specification in the notation are to be built. The transformation is intended to be a suitable input for the automatic test case generation work under investigation by [Don97].

In order to evaluate the usefulness of formalizing the SSR technique, an experiment is proposed to objectively evaluate the semi-formal SSR technique to the formalized version. Measurements are to include defect detection rates and the effort involved in generating, reviewing, correcting, and translating the specification to test cases.

11. References

Bar97 Luciano Baresi, Alessandro Orso, and Mauro Pezze, "Introducing Formal Specification Method in Industrial Practice", *Proceedings of the 19th International Conference on Software Engineering*, May 17-23, 1997, Boston, Massachusetts, USA, 1997, pp. 56-66.

Day97 Nancy A. Day, Jeffrey J. Joyce, and Gerry Pelletier, "Formalization and Analysis of the Separation Minima for Aircraft in the North Atlantic Region", *4th NASA LaRC Formal Methods Workshop*, Hampton Virginia, USA, September 10-12 1997 (to appear).

Dem78 Tom Demarco, Structured Analysis and System Specification, Prentice-Hall, Inc., USA, 1978.

Don97 Michael R. Donat, "Automating formal specification-based testing", In Michel Bidoit and Max Dauchet, editors, *TAPSOFT '97: Theory and Practice of Software Development, 7th International Joint Conference CAAP/FASE*, volume 1214 of Lecture Notes in Computer Science, Springer-Verlag, April 1997.

Hal97 Anthony Hall, "What's the Use of Requirements Engineering?", *ISRE '97, Third IEEE International Symposium on Requirements Engineering*, Jan. 6-10 1997, Annapolis, Maryland, USA, 1997, pp. 2-3.

Hen80 K. Heninger, "Specifying Software Requirements for Complex Systems: New Techniques and Their Application", *IEEE Transactions on Software Engineering*, Vol. SE-6, No. 1, January 1980, pp. 2-13.

Jac92 Ivar Jacobson, Object-Oriented Software Engineering a Use Case driven approach, Addison Wesley Longman Ltd., England, 1992.

Jac94	Ivar Jacobson, “Basic Use Case Modeling”, *Report on object analysis and design*, July-August, 1994.
Lev94	Nancy Leveson, Mats Heimdahl, Holly Hildreth, and Jon Reese, “Requirements Specification for Process-Control Systems”, IEEE Transactions on Software Engineering, Vol. 20, no. 9, September, 1994, pp. 684-707.
Pai93	T. Paine, P. Krutchen, and K. Toth, “Modernizing ATC Through Modern Software Methods”, 38th Annual Air Traffic Control Association Convention, Nashville, Tennessee, October, 1993.
Rus97	John Rushby, “Calculating with Requirements”, *ISRE ‘97, Third IEEE International Symposium on Requirements Engineering*, Jan. 6-10 1997, Annapolis, Maryland, USA, 1997, pp. 144-146.
Sai96	Hossein Saidian, “An Invitation to Formal Methods”, *IEEE Computer*, April 1996, pp. 16-30.
Spi88	J.M. Spivey, Understanding Z, Cambridge University Press, Great Britain, 1988.
War85	Paul Ward and Stephen Mellor, Structured Development for Real-Time Systems, Volumes 1,2,3, Yourdon Press, USA, 1985.

12. Appendix A. Library Specification Unit Example

Title:
\ Search Catalogue \

Overview:
The Search Catalogue specification unit describes the processing performed when an operator or external library requests to search for an item. The request may search for an item using the author name, title, call number, or subject. The response is sorted according to the search key used in the search request.

Stimuli:
1) The library system shall satisfy the requirements described below upon receipt of a [search request] from:

a) Operator <search request>;

b) Remote Operator <search request>;

c) VPL <search request>;

d) SFU <search request>;

e) UVIC <search request>;

f) EPL <search request>.

Responses:

1) As specified by the requirements described below, the library system shall return an [acceptance] to:

    a) Operator <acceptance>.

2) As specified by the requirements described below, the library system shall send a [search response] to:

    a) Operator <search response>;

    b) Remote Operator <search response>;

    c) VPL <search response>;

    d) SFU <search response>;

    e) UVIC <search response>;

    f) EPL <search response>.

Requirements:

1) Upon receipt of a [search request], if the [search request] is not rejected, then the library system shall return an [acceptance].

2) Upon receipt of a [search request], if the [search request] is not rejected, then the library system shall send a [search response].

    a) If the <search key> in the input is by <author name>, then the sent [search response] shall be sorted by <author name>.

    b) If the <search key> in the input is by <title>, then the sent [search response] shall be sorted by <title>.

    c) If the <search key> in the input is by <call number>, then the sent [search response] shall be sorted by <call number>.

    d) If the <search key> in the input is by <subject>, then the sent [search response] shall be sorted by <subject>.

Declarations:
Type Names:
1) :<search key>.

2) :<boolean operator>.

Type Aliases:

Static Associations:
1) A <search request> is associated with:

a) one or more <search key>s;

b) zero or more <boolean operator>s.

Constants:
1) <author name>:<search key>.

2) <title>:<search key>.

3) <call number>:<search key>.

3) <subject>:<search key>.

Adaptation Data:

Stimulus and Response Components
1) <search request>:<request message>.

2) <search response>:<response message>.

Other Specification Units Used:

Performance:
The mean amount of time to process searching for an item is 3.5 seconds.
The maximum amount of time to process searching for an item is 6.5 seconds.