Development and Application of a Context-free Grammar for Requirements

William Scott
Stephen Cook
Joseph Kasser

Systems Engineering and Evaluation Centre
University of South Australia, Mawson Lakes Campus
Mawson Lakes Boulevard
Mawson Lakes, South Australia, 5095
e-mail: William.Scott@unisa.edu.au

Abstract. Requirements form the basis of the systems engineering life cycle activities but creating a good set of requirements is a difficult task. Some difficulties can be reduced through the application of a context-free grammar for requirements to reduce the complexity of requirements elicitation. Developing the grammar involved a melding of computer science and natural language that yielded useful insights into the nature of requirements. The grammar was originally developed to empower a case-based assessment system for requirements but the value in using the grammar was such that it inspired the creation of a requirements writing tool, BADGER.

Introduction

The requirements engineering (RE) process is an important early stage in the systems engineering life cycle. Poor requirements are often cited as a leading cause of project failure (Standish Report, 1995). RE encompasses two symbiotic parallel processes, the RE activities and requirements management processes. The RE process activities, generically outlined in Figure 1 (Kotonya & Sommerville, 1998), describe the actions performed by engineers to elicit, analyse, document, store and validate requirements.

The goal of RE is to develop and maintain a good set of requirements. Practice has shown that this is difficult (Vencel, 1999; Martin, 2000; Kasser, 2001). The difficulties

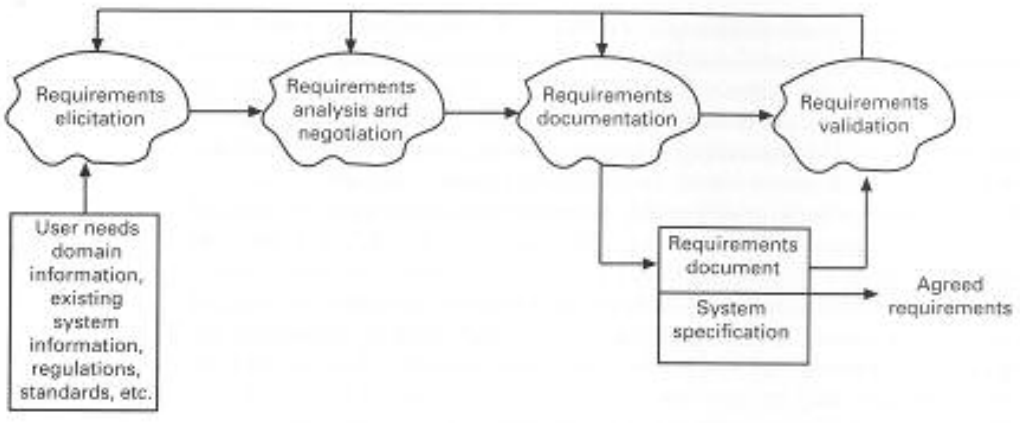

Figure 1: Requirements Activities (Kotonya & Sommerville, 1998).

stem from several sources. Firstly, requirements are commonly written in natural language. Natural language is inherently ambiguous and misunderstanding will often result. To some extent this has been addressed through the adoption of stylized expression (e.g. MIL-STD-490A, MIL-STD-961D) and avoidance of vague words.

Secondly, requirement writers require substantial contextual tacit and explicit knowledge in order to perform the requirements elicitation and analysis tasks well. Engineers use a large amount of assumed contextual knowledge when examining a requirements document drawn from their experience and training. It is through the use of this knowledge that 'comprehensibility', 'feasibility', and 'solution independence' of requirements can be assessed. It is difficult for a computer to apply this contextual information as they lack the ability to implicitly learn and adapt their knowledge.

Thirdly, requirement sets can be large and it is difficult for the author(s) to gain familiarity with the entire set in order to ensure that the set represents a feasible proposition and is internally consistent.

Given the importance of the field, significant work has been performed to address these issues. For example, ontologies have been defined that include the attributes of good requirements and rules for detecting the attributes (INCOSE, 2000; Kasser, 2001; Schneider & Buede, 2000).

Current commercially available tools focus on the requirements management capabilities. Concept demonstrator tools have been developed to assist the RE activities through the application of artificial intelligence. A summary of the concept demonstrators can be found in Scott & Cook (2003) and Kasser et al (2003).

What is needed is an automated means to understand individual requirements (and hence requirement sets) and reason about their quality using a repository of accumulated wisdom. To attempt this, a concept demonstrator is under development to apply case-based reasoning (Leake, 1996; Luger & Stubblefield, 1998) to address these issues from a unique perspective. Solutions are generated by finding previous problems similar to the current dilemma and adapting their outcomes to the current problem. With respect to RE, previous requirements sets are compared to the current set to permit associated information (such as testability and feasibility) to be inferred. More information about the case-based reasoning concept demonstrator can be found in Scott & Cook (2004).

An integral part of the assessment process is the ability to compare requirement statements and identify semantic matches. When comparing requirements between documents it is necessary to be able to compensate for writer style and project specific information, such as system references. To overcome this challenge, a context-free grammar was developed to facilitate the parsing of requirements. Once deconstructed, the components of each requirement can then be readily compared to detect equivalence.

A Grammar for Requirements

When developing a grammar to parse requirements, we can draw on two fields of knowledge. The first field is formal language practice that one encounters in computer science that enables automatic parsing of the requirements. The other field is English grammar structures since the targeted requirements are written in natural language.

Classes of Grammars

An artificial grammar is a set of production rules that define the production of clauses in a language. When developing a structure to represent requirements statements, it is prudent to examine the common types of artificial grammars used. These grammars can be classed in a hierarchical structure as seen in Figure 2 (Aleksander & Hanna, 1976). Natural language is difficult to express using a mathematical structure. The grammatical rules of the language has exceptions too numerous to capture. Subsets of natural language have been developed which can be expressed using mathematical structures.

A subset of natural language is phase-structure grammars, which facilitates the definition of a language's structures. To express the language mathematically, we first need to define some sets. Let us define a vocabulary V that can be broken into two mutually subsets, non-terminal symbols V_n and terminal symbols V_t. We can then define a set of production rules, P, to relate the symbols in the vocabulary. These relations are of the form:

$$\alpha \rightarrow \beta$$

where α and β are both strings in the total vocabulary V. We can also define a set of start symbols in the vocabularyω. In our case, ω is the symbol of a requirement.

A grammar G can be defined as the quadruple:

$$G = < V_n, V_t, P, \omega >$$

So our grammar has a set of start symbols, ω, that can be changed using a series of production rules ($\Rightarrow^*$) into a terminal symbol s where $s \in V_t$. Thus we define the languages using phase-structure grammars as the set L expressed as:

$$L(G) = \{s \mid s \in V_t \text{ and } \omega \Rightarrow^* s\}$$

A subset of the phase-structure grammars is a context-sensitive grammar that restricts the transitions so they are non-expansive. The restriction on the set P is:

$$|\alpha| \leq |\beta| \text{ where } \alpha \rightarrow \beta \in P$$

This creates a calculable upper bound on the number of possible substitutions for any given string.

Context-free grammars are a subset of the context-sensitive grammars that allow substitutions independent of the rest of the structure. The additional restriction for context-free grammars is that the left-hand side of transitions consist of a single symbol and the right-hand side is a non-empty string over the total vocabulary. We can express the resultant set as:

$A \rightarrow \alpha, A \in V_n, \alpha \in V - \{\lambda\}$ where λ is the empty string. Context-free grammars are commonly encountered as they are used to define the syntax for programming languages and enables parsing to extract the semantics.

Finite-state grammars are the ultimate restriction when an explicitly defined set of states can be reached through a limited set of

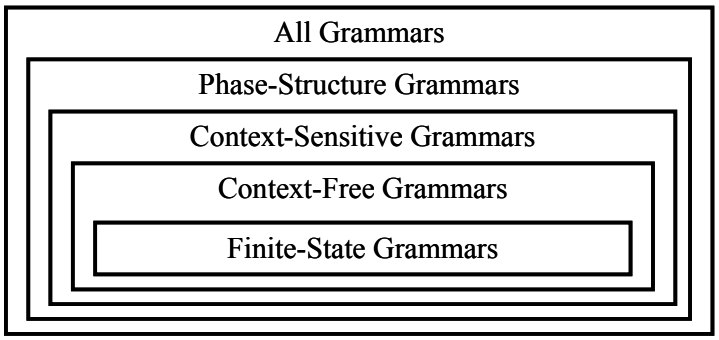

Figure 2: Classes of Grammars (Aleksander & Hanna, 1976)

transitions. The productions are of the form

$$A \rightarrow \alpha B \text{ or } A \rightarrow \alpha \text{ where } A,B \in V_n \text{ and } \alpha \in V_t.$$

It is unrealistic to define a finite-state grammar for even heavily stylised natural language. However, it is possible to construct a context-free grammar specifically for requirements given the restrictions placed upon writing requirements supplemented by common writing habits.

English Grammar Constructs

Sentences are made up of one or more clauses. Identification of the clause type and subsequent semantic analysis of the clause reveals the information contained within.

Clauses are classified as either independent or subordinate. Independent clauses can stand alone syntactically as a full sentence and contain at least one subject and verb. Whereas an independent clause forms the core of the requirement giving the subject, action and targeted object or attribute. Every sentence has an independent clause.

Subordinate clauses are dependent on the information in the rest of the sentence to extract the meaning and thus cannot be viewed entirely separately. Subordinate clauses are also called dependent clauses. There are many subtypes of subordinate clauses and some commonly encountered types are:

Temporal Clause: This clause refers to the time an action occurs. When used in requirements, a temporal clause specifies when the action occurs (e.g. *When the user presses the button,* the system shall…).

Conditional Clause: Clauses of this type make the proposed action dependant on the fulfilment of the subordinate clause. These clauses are typically introduced with the conjunctions "if" or "unless" (e.g. *Unless otherwise specified,* the system shall…).

Relative Clause: These clauses give additional detail to a particular noun. These can be either restrictive or non-restrictive. A non-restrictive relative clause is a relative clause that does not aid in the identification of the referent of the preceding noun (e.g. the student record *in the database*). These can also be referred to as descriptive clauses. The use of non-restrictive relative clauses in requirements adds redundant information that should be trimmed. A restrictive relative clause helps identify the referent of the word modified (e.g. the row of tables *at the front of the room*). In requirements, restrictive relative clauses are often used to define the response and operational envelope criterion.

Other types of subordinate clauses exist but are rarely encountered in requirements.

A Context-free Grammar for Requirements

Thus we use formal language theory to represent the semantics of the English grammar constructs. In developing the grammar it was necessary to balance the language's expressive power against the proving capabilities. The language was designed to apply requirements engineering best practice. As such, restrictions were placed on the grammatical constructs permitted. A restriction was the omission of pronouns from the vocabulary. It is recognised that the use of pronouns introduces ambiguity into requirements and should be inhibited (INCOSE, 2000). Another restriction is the focus on active voice. Using requirements written in active voice allows identification of the relationships captured within the requirement.

The grammar also needed to apply to previous manually written requirements. The grammar had to be designed to compensate for minor mistakes made in writing

requirements. A collection of requirements documents were examined and parsed to determine the applicability of the grammar. The grammar was then adapted to expand the coverage to include common requirement patterns. The result on the grammar was multiple structures for similar semantics to capture variations in writing style. The grammar wasn't adjusted for requirements found to be extremely poor, as it would have a degrading effect on the abilities of the grammar when evaluating good requirements.

Figure 3 summarizes the grammar in Backus Naur Form (Naur, 1960) to an intuitive level. In the figure, TC Clause are Temporal Conditional Clause.

An Example of the Grammar's Application

An example can better explain the use of the grammar. This example was taken from a communication specification. Consider the requirement:

"Organisational Message Traffic shall be transferred with no greater than 1 in 10^3 BER"

The grammar identifies the requirement as containing an independent clause followed by a restrictive relative clause. The independent clause, "Organisational Message Traffic shall be transferred", identifies the

- Subject as "Organisational Message Traffic";
- Auxiliary Verb as "shall";
- Verb as "be"; and
- Target as "transferred".

The restrictive relative clause contains the acceptance criteria of the requirement. This clause identifies the:

- Preposition as "with";
- Criterion Indicator as "no greater than";
- Value's Number as "1 in 10^3"; and
- Value's Units as "BER"

Thus the constituent components of the requirement have been extracted for individual examination. With our decomposed requirements, we can better compare requirements with different writing conventions. For example we can equate "less than" with "no greater than" or "10^3" with "1000". The grammar can be adapted to reflect the practices of the company in this way. Thus we have a mechanism to semantically compare

<Requirement> ::= [<TC Clause> ","] <Independent Clause> [<Restrictive Relative Clause>] |
<Independent Clause> [<Restrictive Relative Clause>] [<TC Clause>]
<Independent Clause> ::= <Subject> <Auxiliary Verb> <Verb> <Noun Phrase>
<Restrictive Relative Clause> ::= [<Preposition>] <Criterion Indicator> <Value>
<TC Clause> ::= <TC Indicator> <Noun Phrase> [<Verb> <Noun Phrase>]
<TC Indicator> ::= <Preposition> | <Condition>
<Value> ::= <Number> <Units>
<Noun Phrase>::= [<determinant>] {<Adjective>} <Noun> [<Preposition> <Noun Phrase>]
[<Conjunction> <Noun Phrase>] |
[<determinant>] < Adjective > {<Adjective>} [<Preposition> <Noun
Phrase>] [<Conjunction> <Noun Phrase>]
<Subject> ::= [<determinant>] {<Adjective>} <Noun>
<Auxiliary Verb> ::= "shall" ["not"]
<Criterion Indicator> ::= "no greater than' | "no less than" | "within"

Figure 3: Context-free Grammar for Requirements

requirements that compensates for writer style. This is crucial for the application of case-based reasoning.

Grammar Application for Requirements Elicitation

While developing and testing the case-based assessment tool, it was observed we could benefit from using the context-free grammar to generate requirements. The grammar was used as the basis for a tool to generate requirements. The tool was implemented as a wizard named BADGER (Built-in Agent using Deterministic Grammar for the Engineering of Requirements) within the tool TIGER (Tool to InGest and Elucidate Requirements), a part of the "Prototype Educational Tools for Systems and Software Engineering" (PETS) project (Kasser et al, 2003). PETS is a suite of concept demonstrator tools to perform SE activities. Each tool is named to facilitate an acronym based on an animal.

BADGER acts by prompting the user for the information relevant to each clause of the requirement grammar. By prompting the user for specific information, the user must make a conscious decision *not* to add the information concerning conditions and performance criterion into a requirement. In traditional free form requirements writing, the converse applies where the writer must actively specify the information.

BADGER ensures that adequate information is obtained for a specific clause before entering the clause into the structure. This reduces ambiguity in the requirement while ensures completeness of the requitement.

With the information for the various clauses, BADGER constructs the requirement for the user using the predefined format. The automatic formatting of the requirement standardises the requirement's structure by removing the author's style. This standardisation can benefit large projects where multiple authors contribute to the requirements.

Through pull-down menus that retain previous entries, the user can enter requirements faster then manually writing the document.

BADGER was also seen as useful in conjunction with the case-based assessment tool as it would enforce structures that may be parsed for the assessment tool to work upon. This improves the efficiency of the case-based assessment.

The tool was introduced to several classes of students being introduced to systems engineering. It was seen that the resulting requirements from the groups with the tool were of a higher quality when compared to previous classes denied the tool. An unexpected consequence of the tool was the focus of discussion in the class. Previous classes focused on the difficulty in expressing the statements but the classes with BADGER focused on the content.

Future Activities

Through the research into the application of the grammar for case-based reasoning and requirements elicitation the grammar is evolving to improve its capability to capture and construct information. The grammar is undergoing refinement for a broader spectrum of requirements as documents become available. The symbiosis of the tools through the grammar is resulting in the tool set becoming better aligned and increasingly practical as each are developed.

Conclusion

To enable automated reasoning of requirements, a grammar is needed to enable parsing. A context-free grammar has been developed to enable the semantic comparison of requirements. This ability facilitates the application of case-based reasoning to requirements. In creating a context-free grammar to structure requirements we have gained a better understanding of the constituent components of a requirement. The process of requirements elicitation can also be improved through an application of the context-free grammar to guide the writer. BADGER provides a mechanism that allows relatively inexperienced requirements writers to generate better requirements.

Acknowledgements

We would like to thank Andrew Stonham, Werner Kolze, BAE SYSTEMS, Australia and the Australian Research Council for their input and support for the SEDM Project

References

Aleksander, I. & Hanna F.H., 1976 *Automata Theory: An Engineering Approach,* Crane, Russak & Company, Inc.

INCOSE, 2000 *Systems Engineering Handbook,* (Online 20 October, 2003), URL: www.incose.org

Kasser, J. E., 2001, 'Writing Requirements For Flexible Systems', *INCOSE UK Spring Symposium, 2001.*

Kasser, J. E., Tran, X-L, Matisons, S. P., 2003 'Prototype Educational Tools for Systems and Software (PETS) Engineering', *Proceedings of AAEE Conference, Brisbane 2003*

Kotonya, G. & Sommerville, I.,1998 *Requirements Engineering: Processes and Techniques,* John Wiley & Sons

Leake, D., 1996, *Case-based Reasoning: Experiences, Lessons, and Future Direction,* Menlo Park: AAAI Press/MIT Press

Luger, G. F., & Stubblefield, W. A., 1998, *Artificial Intelligence, Structures and Strategies For Complex Problem Solving,* Addison Wesley Longman Inc

Martin, J., 2000, 'Requirements Mythology: Shattering Myths About Requirements and the Management Thereof', *Proceedings of the 2000 INCOSE Symposium*

Naur, P., 1960, 'Revised Report on the Algorithmic Language ALGOL 60', *Communications of the ACM,* Vol 3 No 5, pp 299-314

Schneider, R.E. & Buede D.M. 2000, 'Properties of a High Quality Informal Requirements Document', *Proceedings of the 2000 INCOSE Symposium*.

Scott, W. R. & Cook, S. C., 2003, 'An Architecture For An Intelligent Requirements Elicitation And Assessment Assistant', *Proceedings of the 13th Annual INCOSE Symposium, Washington, USA*

Scott, W. R. & Cook, S. C., 2004, 'A Requirements Assessment Architecture That Combines Natural Language Parsing And Artificial Intelligence', *Proceedings of the 14th Annual INCOSE Symposium, Toulouse, France*

Standish Report, 1995, *Chaos Report,* (Online 5 July, 2000), URL: http://stadishgroup.com/visitor/chaos.htm

Vencel, L., 1999, 'Why Is Defining Requirements So Hard? Or Do I Really Need TO Understand Intervention Theory?', *Proceedings of the SETE 1999 Conference*

Biographies

Mr. William Scott graduated in Applied Mathematics and Computer Science at University of Adelaide in 1999, after which he was granted a research scholarship at the System Engineering and Evaluation Centre at the University of South Australia. His time spent at SEEC has been spent in the examination and implementation of the AP-233 standard and the application of artificial intelligence to improve requirements engineering.

Professor Stephen Cook was awarded a bachelor's degree in electronics engineering from the South Australian Institute of Technology in 1977, and completed three postgraduate qualifications part-time whilst building his career: an MSc in computer science from the University of Kent, UK; a Graduate Diploma in electronic systems from the University of South Australia; and a PhD in 1990 in measurement science and systems engineering from the City University, London, UK. He has had a varied career that commenced with over ten years' engineering experience in the telecommunications and aerospace industry after which he joined the Defence Science and Technology Organisation (DSTO) as a research scientist rising to Research Leader Military Information Networks in 1994. In this role he supervised a branch of over 60 staff engaged in all aspects of military communications and networking research for command and control systems. Since 1997 he has been seconded to the University of South Australia as the foundation DSTO Professor of Systems Engineering and was the inaugural Director of the Systems Engineering and Evaluation Centre. He has a wide span of research interests including systems engineering of C2 systems, systems approaches for defence capability development, acquisition modernisation, and theoretical frameworks to support the coherent teaching of systems engineering. He has contributed to three books and has published over seventy refereed journal and conference papers. Prof Cook is a Fellow of the Institution of Electrical Engineers (UK), a Fellow of the Institution of Engineers Australia, and Past President of the Systems Engineering Society of Australia.

Associate Professor Joseph Kasser has been a practising systems engineer for 30 years. He is the author of "*Applying Total Quality Management to Systems Engineering*". He holds a Doctor of Science in Engineering Management from The George Washington University, and is a Certified Manager. He is the DSTO Associate Research Professor at the Systems Engineering and Evaluation Centre at the University of South Australia. He performs research into the nature of systems engineering and the properties of object-oriented requirements. He is a recipient of NASA's Manned Space Flight Awareness Award for quality and technical excellence for performing and directing systems engineering.