Formalizing a Derivation Strategy for Formal Specifications from Natural Language Requirements Models

Maria Virginia Mauco, Maria Carmen Leonardi, Daniel Riesco, German Montejano, Narayan Debnath

2005 IEEE International Symposium on Signal Processing and Information Technology

This paper tries to bridge the gap between LEL specification (natural language) and RAISE specification (formal requirement specification language). It discusses the LEL (Language Extended Lexicon) method for specifying requirements in natural language. LEL consists of entries for frequently used words in the pertaining domain. It assigns a set of notions and behavioral responses to every entry. Notion represents the meaning/sense of the word and behavioral response indicates the use of the word in the system. On the other hand, RAISE (Rigorous approach to Industrial Software Engineering) specification is a formal specification method which offers features similar to OO paradigm such as encapsulation, polymorphism etc. There are tools available for writing specifications in RSL (RAISE specification language), performing justifications, translating into imperative languages etc. But it is hard to write specifications in RSL than in natural language such as LEL. This paper proposes a set of rules to convert LEL specifications into RSL. These rules are specified in RSL itself. The transformation also provides 'trace' relationships which helps in validating the inferred formal specification. The main aim of the proposed translation is to automatically define RSL types from LEL. This can be considered as the primary step in conversion of informal requirements into formal specifications.

The major difference between this approach and our approach is the use of LEL as input. LEL itself is hard to write and hence requires some effort from the business analyst. There are not many specifications which are currently written in LEL. Thus extracting the static and dynamic behavior of a system from its natural language specification will be more useful than parsing LEL.

Transformation of SBVR Business Design to UML Models

Amit Raj, T.V. Prabhakar, Stan Hendryx

Proceedings of the 1st conference on India software engineering conference

This paper describes a technique to transform SBVR (Semantics of Business Vocabulary and Rules) designs written in structured English (controlled language) into UML models. The methodology uses logical formulation of SBVR rules. In this method, after categorizing the rules as operative or structural, a set of rules which contribute to activity diagram are identified first. Each fact type with a transitive verb is objectified into an activity. Activity diagrams are then generated by using an algorithm which also adds guard conditions and actions. The activity diagram creation technique addresses activity groups, fork and join operations etc. Rule sequencing engine establishes order among the activities. After creating the activity diagram, sequence diagrams are generated using an algorithm. It identifies messages and determines source and destination for each of them. Messages are then ordered to create sequence diagrams, but this ordering algorithm has some limitations such as unable to detect actor or unable to process activities immediately preceding end state. The SBVR specification provides guidelines to transform the logical formulations into class diagrams but this paper adds features like operations of classes, multiplicities and association ends. In addition to the presented technique, a tool has been developed in Java to create, update and validate SBVR designs. The tool can then be used to create logical formulation trees and the UML diagrams such as activity diagram, sequence diagram and class diagram.

Though SBVR is a standard developed by OMG for writing business rules, this approach still requires logical formulations of the business rules which are hard to write. Cogito approach involves building the business ontology before writing the use-case specifications, where as Dowser is not concerned with the ontological elements.

Information Extraction, Automatic

H. Cunningham, Encyclopedia of Language & Linguistics, Second Edition (2006), volume 5, pages 665-677.

The survey paper introduces Information Extraction (IE) and its relationship with Information Retrieval (IR). It talks about the parameters such as complexity and specificity that govern the performance of IE. Various application scenarios such as financial analysis, marketing strategies, PR working, media analysis etc. are discussed. According to the MUC program, IE system is considered to be composed of five primary tasks / components viz. Named entity recognition (NE), Coreference resolution (CO), Template element construction (TE), Template relation construction (TR) and Scenario template production (ST). Authors give examples to illustrate the tasks in detail. Empirical data about the accuracies of each task is presented. Later, the paper also discusses the new ACE (Automatic Content Extraction) program which consists of phases such as Entity detection and tracking, Relationship detection and tracking, event detection and characterization. Ontology based IE is also illustrated with an example of ontology creation from natural language text. Two types of OBIE (Ontology based IE) are explained viz. 'identification of instances from the ontology' where the application tries to determine the instances of ontological elements in the given text and 'automatic population of ontology' in which instances from text are automatically added to the ontology in the right place. At the end, examples of IE system are given which offer IE facilities in different forms such as add-ons, APIs etc.

GATE, General Architecture for Text Engineering

H. Cunningham, Computers and Humanities, Volume 36, pages 223-254, 2002.

This paper discusses GATE, from requirements to design and the actual implementation. GATE defines two types of resources, LE (Language Resource) which denotes data-only resources and PE (Processing Resource) which denotes algorithmic or programmatic components. GATE provides implementation of certain common algorithms and data structures used in text engineering. The paper identifies primary requirements for SALE (Software Architecture for Language Engineering) and describes common set of use-cases for that. In the design and implementation section, the three main components of GATE are described viz. GDM (GATE Document Manager), CREOLE (Collection of REusable Objects for Language Engineering) and GGI (GATE Graphical Interface). GDM is the component which manages the underlying data. CREOLE is the collection of LE components integrated with the system which can be used to perform various tasks on the data. GGI is a development environment which allows access to services of other components and visualization and debugging tools. For representation of corpora/documents in GDM, two candidates were evaluated viz. TIPSER and SGML. TIPSER was chosen for its better time/space profile, ease of random access, and ability to easily represent non-contiguous text structures. However, GATE also supports import and export of SGML. GDM acts like a central repository which stores all the information an LE system generates. It provides a uniform API to access the stored data. Text analysis in GATE is done by CREOLE modules. CREOLE interface defines a standard API which allows access to GGI and GDM. GDM imposes input/output format restrictions on CREOLE modules. Persistence is achieved using TIPSER database model in which any object can be reached from the collection to which it belongs. The paper then describes several ways to integrate modules into GATE such as dynamic coupling, loose coupling and static coupling. Users can also specify metadata for components in a configuration file which defines pre-conditions, post-conditions and viewers to be used for the generated results. Graph of CREOLE can be constructed using dependencies caused by pre and post conditions. User can then select a sub-graph (called as task-graph) to be executed. For few example tasks, working of GGI (GATE Graphical Interface) is described. Different forms of data visualizations are offered in GATE such as single-span annotations, multi-span annotations and trees. GATE also allows data comparison similar to diff viewers and data editing for manual annotation. In the evaluation section, the paper describes effectiveness of GATE in the context of usage of GATE in various fields, code reuse and a reflection on requirements against the release version. The paper concludes with list of major advantages and drawbacks of GATE (version 1).

Object-Oriented Analysis: Getting help from robust computational linguistic tools

Sylvain Delisle, Ken Barker, Ismail Biskri

This paper vouches the use of advanced computational linguistic tools called DIPETT (Domain Independent Parser of English Technical Texts) and HAIKU (Semi-Automatic Semantic analysis tool) for providing assistance during initial phases of object-oriented analysis. It describes Conger's OO analysis methodology as a representative technique which consists of four steps viz. 'developing summary paragraph', 'identifying objects of interest', 'identifying processes', 'Defining attributes of objects'. Last two steps are considered for automation using DIPETT and HAIKU. DIPETT is the parser tool used to parse given text. It parses individual sentences into main clause, sub-clause structure. In case of ambiguity, user is required to choose one of the parse tree choices. After this phase, HAIKU is used for semantic analysis which processes individual clauses and determines semantic relationships between clauses. A sample specification of video rental system was used to evaluate the effectiveness of the technique. For measuring the accuracy, results produced by automation are compared against those created by human experts. The results are said to be encouraging because of the considerable accuracy that was attained. Moreover, by using text analysis systems which were not specifically designed for computer-assisted OO analysis and for texts which were not adapted or modified for making suitable for the system.

In this approach generalized text analysis tools are used to assist OO analysis process, whereas Dowser uses technique which is specific to requirement analysis. I can be argued that since a specialized technique is used for analyzing requirements, more details can be gathered automatically than using generalized analysis tools and manually inferring the model.

Attempto Controlled English (ACE)

Norbert E. Fuchs, Rolf Schwitter, CLAW 96, The First International Workshop on Controlled Language applications, Katholieke Universiteit Leuven, 26-27 March 1996

This paper explains the ACE (Attempto Controlled English) language in detail. ACE is a computer processable subset of English language and it does not require expert knowledge in formal methods or computational linguistics. These two features make it a better technique than completely informal or completely formal specification. ACE acts like a textual view of formal specification in predicate logic. Attempto accepts specification texts and translates them into discourse representation structures (DRS). DRS is a structural representation which is created for every sentence. It can also convert the generated DRS into prolog if desired. Attempto uses deterministic parsing and has several restrictions on usage of noun, verbs, adjectives etc. It uses a lexicon which is specified by user. After interpreting the structure of the sentence, Attempto provides a paraphrase for the sentence which user needs to validate. User can then either accept the interpreted structure or rephrase the specification. Moreover, this specification can be executed by using instance data.

ACE approach is very close to the Cogito and Dowser approach. ACE also uses lexicon specified by user which is similar to Cogito's ontology building. However, ACE does not use dictionary like WordNet. ACE can execute the specification if it can be converted into first order logic. But it does not create any static or dynamic model of the system which can be used in further phases of software development such as architecture design / high level design. ACE also has several limitations for using verbs, nouns etc. which could be overcome using a link grammar parser approach as in Dowser.

Automated Conversion from Requirements Documentation to an Object-Oriented Formal Specification Language

Beum-Seuk Lee and Barrett R. Bryant, SAC '02: Proceedings of the 2002 ACM symposium on applied computing*, pages 932-936, New York, NY, USA, 2002*

In this paper, the authors explain a technique which uses Contextual Natural language processing (CNLP) and Two-Level Grammar (TLG) to construct a bridge between NL requirements and formal specifications. The authors argue that restrictions on the requirement specification language can cause loss of information and also user has to remember all the restrictions. Hence they are using CNLP for disambiguation and TLG for bridging the formalism level gap. First, the requirements specification is converted to XML form. In the XML format, the specification is divided into paragraphs and sentences. The XML document is then converted to knowledge base (KB) using CNLP. Bottom-up parsing is used for part-of-speech recognition and top-down parsing is used for part-of-sentence recognition. Huge corpora of statistically ordered part-of-speeches are used to resolve syntactic ambiguities. It also does anaphora resolution by using recency constraint i.e. by assigning higher priority to recently used words and co-referenced words. In the knowledge base, each sentence is stored in a tree-like structure which determines the context of that sentence. Once KB is constructed it can be displayed in XML format or queried in natural language. KB is then converted into TLG by removing contextual dependency. TLG is an intermediate representation which defines classes that contain data and rules. Root of each context tree becomes a class in TLG and rules for that class are created from sub-contexts. TLG is then converted to VDM++ specification. VDM++ is a formal specification language for which VDM toolkit is available. VDM++ document can then be translated into implementation language such as Java / C++ using code generator that VDM toolkit provides.

This approach creates intermediate representations from requirement specifications and tries to generate code at the end. Dowser and Cogito aim to provide analysis / design level view of the system. Although the Contextual Natural Language Processing technique seems to perform better, it is still limited by the part-of-speech tagger and also corpora used for resolving syntactic ambiguity. It does not use any generic dictionary and it also does not provide any extension mechanism.