SMART: System Model Acquisition from Requirements Text

Dov Dori, Nahum Korda, Avi Soffer, and Shalom Cohen

Technion, Israel institute of Technology

{dori@ie, korda@tx, asoffer@tx, shalom1@tx}.technion.ac.il

Abstract. Modeling of a business system has traditionally been based on free text documents. This work describes an elaborate experiment that constitutes a proof of concept to the idea that a system model can be acquired through an automated process whose input is a corpus of technical free text requirement documents and whose output is an OPM model, expressed both graphically, through a set of Object-Process Diagrams, and textually in equivalent Object-Process Language. Our experiment has yielded a high quality system model that required a much smaller effort than what would have been needed in the traditional approach.

1 Introduction

Architecting systems in general and software systems in particular is a tedious task that consumes significant time and expertise resources. Systematically transforming unstructured, free text business specification and user requirements into precise and formal system specifications is a laborious and complex operation, where instead of focusing on the overall design, one often gets lost in a clutter of details. Automation could be of great assistance here, not only because it can significantly lower the overall effort, but also because it allows system designers to focus on the system overview, get the "big picture" much more quickly, and ultimately maximize the overall efficiency of the system while minimizing its time to market.

While the vision of automating the modeling and architecting processes by extracting semantics from requirements expressed in free text may seem to make a lot of sense, a wide semantic gap stands in the way of such automation. On one side of the gap that we seek to bridge is free natural language text, while on its other side is a formal, machine "understandable" and processable character stream. Documentation that serves as a basis for architecting new systems or improving existing ones, such as business process specifications or user requirements, is formulated in natural language that is not even in a machine-readable, let alone machine-understandable format.

While formalization of freely expressed ideas, concepts, intentions, and desires into rigorous specifications seems to be beyond the reach of current computing technologies, not all hope is lost. The emergence of the Semantic Web and ontology engineering technologies may point the way to eventually bridge the semantic gap obstacle. Although it still seems unrealistic to expect complete automation of the system

design, partial, semi-automatic solutions that operate under human supervision may already be feasible and may prove to be extremely useful.

Our proposed strategy is to start bridging the semantic gap in parallel from its two sides—the formal side and the natural language (NL) side—as follows:

1. On the formal side of the semantic gap, the need is for a paradigm and a tool that is capable of human-oriented intuitive expression of complex system function, structure, and behavior while at the same time being formal to a degree that a machine can unambiguously process it. Object-Process Methodology (OPM) [1] is obviously an excellent candidate for the task at hand, since Object-Process Language (OPL), the textual modality of OPM, utilizes a constrained subset of English, which brings it a significant step closer to the unconstrained natural language that exists on the other side of the gap. The additional advantage of using OPM is that its two semantically equivalent modalities, one graphic (Object-Process Diagram) and the other textual (Object-Process Language), jointly express the same OPM model. Accordingly, every verbal formulation (OPL) is automatically paralleled by its graphic presentation, (OPD), and vice versa, such that complete equivalence between the two presentations is guaranteed at any point in time.
2. On the NL side of the semantic gap, information extraction technologies will be utilized in order to achieve the following benefits:
    - Extracting from unstructured text elements—entities and links—that are key concepts for the domain and the problem at hand,
    - Detecting and mapping alternative formulations of relevant ontological relations, and
    - Deriving a semi-formalized presentation of the underlying documentation that could be manually organized into a rigorous formal model of the required system.

To prove the concept of deriving an OPM model from unstructured technical text, this paper describes an experiment in which we utilized information extraction techniques in order to automatically generate OPL script—a structured subset of natural English—from which the corresponding diagrammatic specification in the form of a set of Object-Process Diagrams (OPDs) was constructed semi-automatically utilizing the OPM-supporting CASE tool (OPCAT) [2]. The automatically derived OPL sentences served as a basis for modeling the initial requirements. The automatically-generated initial specification was elaborated upon by the system architect conferring with the domain expert—the representative of the system beneficiary or user, and obtained the expert's blessing. This initial OPM-based system specification can be further developed into a complete formal system design with OPCAT, and automatically documented, converted into a set of UML diagrams if so desired, and implemented as a set of JAVA classes.

The experiment described in this paper is, to the best of our knowledge, a first successful attempt to construct a system model in a semi-automatic way from the system's free text documentation of the requirements. The experiment was based on GRACE (Grid Search and Categorization Engine), a European Community Information Society Technology (IST) project [3]. This complex software development project combines Grid, ontology engineering, and knowledge management. GRACE was

found to be suitable for our experiment due to its extensive background documentation, which includes user and system requirements. A subset of this documentation corpus served as the free natural language text on which the automatic content extraction and OPM model building was performed.

The rest of the paper is structured as follows: Section 2 includes a review of the state-of-the-art in automating modeling from free text. This is followed by a description of OPM in Section 3 and application of OPM to model the architecture of our SMART system in Section 4. The experiment is described in section 5, and section 6 presents our conclusions.

2 Automating Modeling from Text: State of the Art

Architectures of systems and their underlying software provide high-level abstractions for representing the function, structure, behavior, and key properties of the system. A first and crucial phase in system architecting is eliciting, gathering, analyzing, and engineering the stakeholders' requirements. In spite of the clear and direct relationships between requirements engineering and system architecture modeling, these two activities have traditionally been pursued independently from one another.

2.1 From Requirements to Architecture

System requirements include the customer's expectations and vision of the desired solution of the business problem at hand, and constraints on the solution. The requirements documentation reflects interests of the different system's stakeholders—customers, endusers, developers, and managers [4]. Requirements deal with concepts, intentions (both explicit and implicit), goals, alternatives, conflicts, agreements, and above all—desired functional and non-functional system features and properties.

Architecting a system from its requirements has not yet fully been understood. The task of system architecting from its requirements is difficult due the complex nature of the interdependencies and constraints between architectural elements and requirement elements. A number of techniques have been proposed, though, to assist in this effort-consuming and poorly understood task .For example, the Component Bus System, and Properties (CBSP) approach [5, 6], also supported by tools [7, 8], is an analysis method that operates through classification of system features and properties as reflected in the requirements and altering their representation using an intermediate language.

Techniques that have been proposed so far to bridge the requirements-design gap commonly involve human-driven conceptual analysis of the requirements—an iterative, error-prone, and resource-consuming effort for extracting domain-knowledge related information from the requirements. The CREWS project [9], which makes use of language processing in scenario-based requirements engineering approach [10], promotes guidance of the elicitation and validation of requirements that is based on textual scenarios.

2.2 Working from Business Specification and User Requirements

Another approach to supporting the requirements engineering (RE) process is based on the fact that natural language plays an important role during the requirements

stage. It is argued [11] that acquisition of application domain knowledge is typically achieved through language manipulation, either through document and text analyses or by means of interviews. It has therefore been suggested there that RE should be supported by a CASE tool that is based on a linguistic approach. Such RE support environment would generate the conceptual specification from a description of the problem space provided initially through natural language statements.

A complete and effective RE process, which naturally involves language manipulation, includes the following steps: (1) *acquisition* of domain-dependent knowledge using NL statements, an automated version of which [12] applies NL-processing-based metadata extraction to automatically acquire user preferences, (2) *graphic representation* of the semantic contents of the NL statements, which should be easy to understand and manipulate, and (3) *mapping* of the real-world description to a conceptual schema, or a requirements-level system model. Based on this analysis, an approach for tackling the inherent complexity of the RE process is proposed [13] that is based on a CASE tool for the requirements engineering process. This CASE tool is essentially a rule-based expert system, which is a highly technical environment that requires substantial support in rule generation, adaptation, and checking.

2.3 Natural Language Processing

Industrial practice has shown that NL requirements are easier to evolve, maintain, and discuss with (possibly non-technical) stakeholders. Recognizing the potential role of natural language processing (NLP) in the requirements engineering process, efforts (e.g., [14]) have been made to identify tasks where NLP may be usefully applied. At the same time, however, a note of caution is sounded by noting the limitations of NLP in requirements engineering [15].

A number of experiments have been reported on the use of NLP techniques in the context of systems development. Lexical analysis was used [16] to find abstractions in unstructured and un-interpreted text. Other studies applied NL parsing and understanding techniques to automatic extraction of models from NL requirements [17, 18, 19]. Several specific NLP tools and techniques, including [20, 21], have been introduced for the purpose of analyzing and controlling software requirements. These techniques rely on lexical analysis to extract abstractions from natural language text [22]. The use of NLP has also been reported in analogical reasoning technology for specification reuse and validation [23]. Although the application of NLP techniques to handling system requirements is appealing, it is often difficult to check and prove properties, such as correctness, consistency, and completeness on those requirements [24]. Abstract systems were suggested for detecting such ambiguities and under-specifications [25] as well as requirement redundancies [26].

When moving from early requirement gathering, in which ideas, concepts, and intentions are expressed with NL, to the analysis phase, the freely expressed NL-based requirements need to be formalized. They need to be replaced by rigorous specifications, so coherence, consistency, and feasibility can be reasoned about, at least semi-formally. Lightweight formal methods were used in [27] for partial validation of NL requirement documents. Checking properties of models obtained by shallow parsing of NL-expressed requirements, they concluded that automated analysis of requirements expressed in natural language is both feasible and useful.

The conclusion drawn from current research is that the RE process should be supported by a CASE tool that incorporates a linguistic approach. The tool should facilitate an RE-support environment that generates a conceptual specification from a description of the problem space provided through natural language statements. We distinguish between two different types of NL sentence analyses. One is the syntactic analysis, which is based on finding the parts-of-speech in a sentence, including object, subject, verb, adjective, adverb, etc. A notable method of syntactic analysis of this form is Knowledge Query and Manipulation Language (KQML) language, proposed by ARPA Knowledge Sharing Effort in 1992. It uses Knowledge Interchange Format (KIF) [28] for content description through an ASCII representation of first order predicate logic using a LISP-like syntax [29]. The other sentence analysis type is the semantic approach, in which we seek the deep, underlying meaning of what the sentence expresses in terms of detecting objects in the sentences and relations between them, or a transformation to an object (its generation, consumption, or change of state) that a process causes through its occurrence. These two different types of NL sentence analyses were adopted by [30] to form their Word Class Function Machine aimed at both the syntactic analysis and semantic analysis of NL. Performance of these analyses has been an issue for Samuelsson [31] who optimized the analysis and generation machinery through the use of previously processed training examples [26].

This paper suggests the use of NLP in conjunction with Object-Process Methodology (OPM) [1] and its supporting CASE tool (OPCAT) [2] for acquisition of application domain knowledge. The proposed approach seeks to extract as much semantics as possible automatically from a given corpus of related technical documents, such as requirement documents, and build from this extracted semantics an initial conceptual model in a semi-automatic way using OPM and its OPCAT support environment. We next focus on OPM.

3 Object-Process Methodology

Most interesting and challenging systems are those in which structure and behavior are highly intertwined and hard to separate. Object-Process Methodology (OPM) is a holistic approach to the modeling, study, and development of systems. It integrates the object-oriented and process-oriented paradigms into a single frame of reference. Structure and behavior, the two major aspects that each system exhibits, co-exist in the same graphic-NL bimodal OPM model without highlighting one at the expense of suppressing the other.

The elements of the OPM ontology are entities (things and states) and links. A thing is a generalization of an object and a process—the two basic building blocks of any system expressed in OPM. Objects are (physical or informatical) things that exist, while processes are things that transform objects. In a specific point of time, an object can be exactly in one state, and objects states are changed through occurrences of processes. Links can be structural or procedural. Structural links express static relations between pairs of entities. Aggregation, generalization, characterization, and instantiation are the four fundamental structural relations. Procedural links connect entities (objects, processes, and states) to describe the behavior of a system. The behavior is manifested by processes that interact with objects in three major ways: (1)

processes can transform (generate, consume, or change the state of) objects; (2) objects can enable processes without being transformed by them; and (3) objects can trigger events that invoke processes.

3.1 The Bimodal OPM Model Representation

Two semantically equivalent modalities, one graphic and the other textual, jointly express the same OPM model. A set of inter-related Object-Process Diagrams (OPDs) constitute the graphical, visual OPM formalism. Each OPM element is denoted in an OPD by a symbol, and the OPD syntax specifies correct and consistent ways by which entities can be linked. The Object-Process Language (OPL), a subset of English formally defined by a grammar, is the textual counterpart modality of the graphical OPD-set. OPL is a dual-purpose language, oriented towards humans as well as machines. Catering to human needs, OPL is designed as a constrained subset of English, which serves domain experts and system architects engaged in analyzing and designing a system. Every OPD construct is expressed by a semantically equivalent OPL sentence or phrase. Designed also for machine interpretation through a well-defined set of production rules, OPL has an XML-based notation that provides a solid basis for automatically generating the designed application. This dual representation of OPM increases the processing capability of humans.

3.2 OPM Refinement and Abstraction Mechanisms

Complexity management aims at balancing the tradeoff between two conflicting requirements: completeness and clarity. Completeness requires that the system details be stipulated to the fullest extent possible, while the need for clarity imposes an upper limit on the level of complexity and does not allow for an OPD (or an OPL paragraph) that is too cluttered or overloaded with entities and links among them. The seamless, recursive, and selective refinement-abstraction mechanisms of OPM enable presenting the system at various detail levels without losing the "big picture" and the comprehension of the system as a whole. The three built-in refinement/abstraction mechanisms are: (1) unfolding/folding, which is used for refining/abstracting the structural hierarchy of a thing and is applied by default to objects; (2) in-zooming/out-zooming, which exposes/hides the inner details of a thing within its frame and is applied primarily to processes; and (3) state expressing/suppressing, which exposes/hides the states of an object. Using flexible combinations of these three mechanisms, the achieved OPM models are consistent by definition.

4 OPM Model of the SMART System

OPM is employed in this research at two levels: one is the specification of the System Model Acquisition from Requirements Text (SMART) system, and the other is an example of the GRACE system, which is the outcome of our proof-of-concept experiment. Having introduced the basics of OPM we proceed to utilize it to model the architecture of the SMART system using OPCAT. The SMART system consists of various software tools that operate cooperatively in order to produce SMART's desired output.

Figure 1 shows the System Diagram (SD), i.e., the top-level Object-Process Diagram (OPD) of the SMART system. The diagram depicts the high-level structure of the SMART system, its main process, input and output, and the user, as well as their inter-relations.

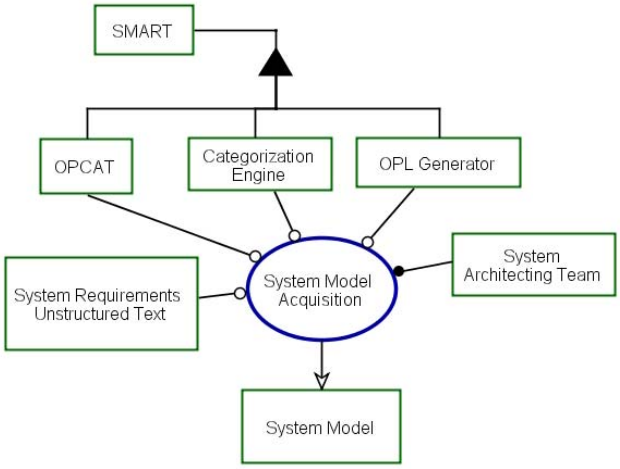

Figure 1 SD – System Diagram (top-level view) of the SMART system OPM model

The graphical description of SMART—the OPD—is backed by corresponding OPL specification, which OPCAT generates automatically in real time in response to the user's graphic input. Table 1 presents the OPL paragraph that describes the OPD in Figure 1.

Table 1 The OPL paragraph describing the SMART system whose OPD is in Figure 1

System Architecting Team handles **System Model Acquisition.**
SMART consists of **Categorization Engine, OPCAT,** and **OPL Generator.**
System Model Acquisition requires **System Requirements Unstructured Text, Categorization Engine, OPCAT,** and **OPL Generator.**
System Model Acquisition yields **System Model.**

The first sentence in the OPL paragraph expresses the fact that the **System Architecting Team** is in charge of, or is involved in the process. As Figure 1 shows, it is connected by an agent link, which triggers the process **System Model Acquisition**. The second sentence expresses the structure of the SMART system. The major components of the system, **Categorization Engine**, **OPCAT**, and **OPL Generator**, are related to the main **System Model Acquisition** process by instrument links. The fourth and last sentence in the OPL paragraph expresses the fact that **System Model Acquisition** generates as a result of its occurrence a new object called **System Model**.

In order to elaborate on the details of the **System Model Acquisition** process depicted in Figure 1 we take advantage of OPM's complexity management capability. Zooming into **System Model Acquisition**, OPCAT creates a new OPD shown in Figure 2, which

is automatically labeled SD1 – System Model Acquisition in-zoomed. SD1 is one level lower than SD in the OPD hierarchy.

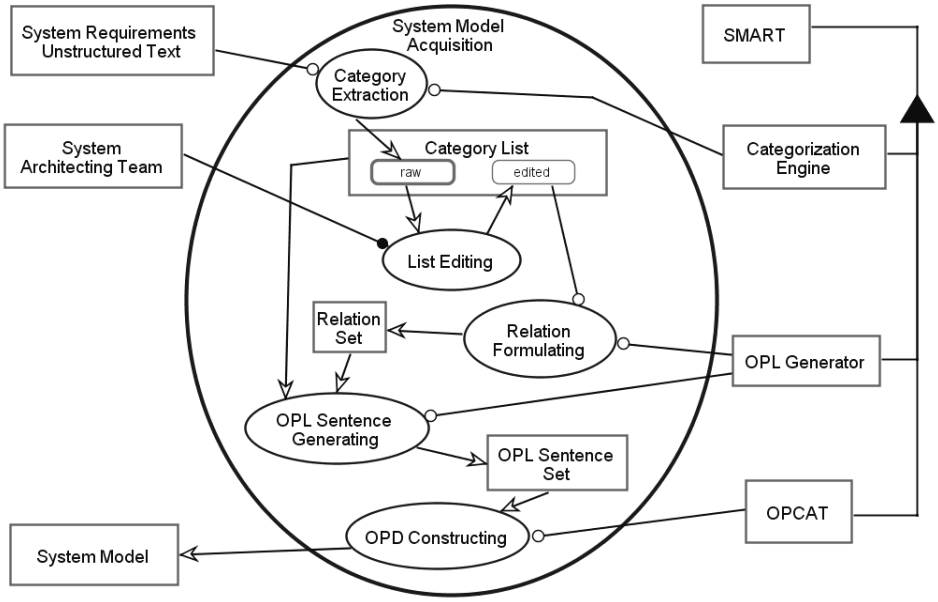

Figure 2 The in-zoomed **System Model Acquisition** process of Figure 1 exposes subprocesses and interim objects

The graphical description of SD1 is backed by another corresponding automatically-generated OPL paragraph. The major subprocesses of SMART, their order of operation (top to bottom), and the interim objects—**Category List**, **Relation Set**, and **OPL Sentence Set**—are obvious from the diagram. The subprocesses and interim objects are also clearly listed in the third sentence in the OPL paragraph, which reads:

> **OPM Model Construction** zooms into **Category Extraction, List Editing, Relation Formulating, OPL Sentence Generating,** and **OPD Constructing,** as well as **OPL Sentence Set, Relation Set,** and **Category List.**

Drilling down into lower levels in the model hierarchy using OPM's abstraction/refinement mechanisms (not shown here due to shortage of space) would reveal further details on the system by showing sub-subprocesses and additional objects' lower level parts and/or attributes. A detailed description of SMART is provided in the next section.

5 The Proof-of-Concept SMART Experiment

Our experiment was aimed to provide proof of concept to the possibility of semi-automatically constructing portions of a model of the system-to-be, as expressed in free text of a corpus of requirement documents. The following is an account of the

experimental settings and procedures. As proof of concept, the experiment proceeded while operating various software programs independently in different phases, rather the attempting to produce a unifying application with a user-friendly graphic interface.

5.1 Automatic Extraction of Categories from Unstructured Text

Our document set of unstructured text consisted of half a dozen free text documents from the GRACE corpus, with a total size of about 0.5 MB. We developed a LISP-based, heuristics-directed categorization engine and utilized it to extract categories from our document set. A category in our context is defined as an idiomatic phrase (word sequence) reflecting the underlying topics in a given corpus of documents. Idioms are expressions whose meaning cannot be deduced from the meaning of its individual constituents, but rather from their consistent use in specific contexts. Table 2 presents a few examples of categories that were automatically extracted from the unstructured GRACE documentation text by our categorization engine.

Table 2 Examples of categories that were automatically extracted from our GRACE documentation

Search Results	**Advanced Searching**	**Knowledge Managing**
Content Sources	**Web Services**	**Query Routing**
Search Engine	**Document Storing**	**Knowledge Sharing**
User Profile	**Document Retrieving**	**Frontend Application**
Web Server	**Content Source Registering**	

Overall, the categorization engine extracted 109 categories, utilizing only its heuristics. Many domains of human knowledge, in particular sciences, have very detailed and precise nomenclatures and dictionaries that could be used for that purpose. We could also calibrate the categorization engine to extract particular categories specified in an external ontology, taxonomy, or thesaurus. Such combination of unconstrained and ontology-guided extraction might generate better results, as the unconstrained categorization could add to the domain vocabulary concepts and expressions that are specific to a document corpus.

5.2 Manual Editing of the Extracted Categories

SMART is intended for use by system engineers with some knowledge domain or previous involvement in similar efforts, since manual category editing requires some domain expertise. The extracted categories were next manually inspected to achieve the following purposes:

1. Selection of those categories that can serve as things (objects or processes) in the OPM model, and classifying them as either object or processes. For example, about half of the extracted things in Table 2 are objects, while the rest are processes. OPM favors processes in the gerund form, i.e., those that end with the "ing" suffix. Indeed, all the processes in the table have this form, but this is not necessarily the case. Fore example, **Document Retrieval** would be classified as a process, synonym with **Document Retrieving**. A counterexample of the word **Building**, means either the object (house) or the process of constructing the house, shows why automatic object-process classification is difficult (but not impossible) to automate.

2. Clustering alternative formulations for the selected OPM things (for example, **Search Results** and **Retrieved Results**) based on their semantic similarity, and
3. Optionally adding OPM things that did not show up among the extracted categories.

An important assistance to the manual editing of categories is the ability of the categorization engine to present all the sentences from the processed corpus in which a particular category appears. Using this feature, a system engineer can focus on the few really relevant instances, in which a particular category occurs, saving the sifting through hundreds of documentation pages. During this inspection, additional categories that were not automatically extracted but are nonetheless relevant for the design may be detected in the text, or may simply come to mind and be manually added.

The system allows *semantic clustering*, i.e., grouping of categories into clusters that share similar a meaning. This caters to the variety of natural language formulations encountered in actual texts. Our experiment has revealed several typical situations in which such clustering is required:

1. Abbreviations and acronyms (e.g., European Data Grid and EDG),
2. Lexical variations (e.g., search results, retrieved documents, retrieved results),
3. Synonyms (e.g., screen, monitor, display),
4. Morphological variations (e.g., registering, registration), and
5. Orthographic variations (e.g., frontend, front-end, front end).

5.3 Automatic Search of OPM Relations

In order to extract OPL sentences from the unstructured text, SMART utilizes a set of configurable, predefined templates. Each template consists of two things and the relation between them, expressed in alternative ways. For example, the **result** relation between a process and an object, expressed in OPL by the verb *yields*, can also occur as *generates*, *results in*, etc. SMART currently utilizes 50 predefined general templates and 20 domain-specific templates that were detected by inspecting various contexts in which the selected categories occurred. These 70 templates were applied to 109 categories organized in 46 clusters. Since not all combinations of things and relations are allowed (for example, the OPM relation **result** cannot exist between two OPM objects from the list of 109 categories, but only between a process and an object, and in this order), the original document corpus was tested against a total of 234,320 templates.

We define *second order regular expressions* as regular expressions, in which the basic unit is a word rather than a character. Instead of comparing character strings, a program that uses second order regular expressions compares word sequences. The program is implemented as a finite-state automaton that operates on suffix-tree index consisting of tokens from the processed text. To guarantee the required expressiveness of the framework, SMART manipulates second order regular expressions, allowing them to be defined on any lexical or grammatical attribute of the processed text, such as part-of-speech, capitalization, and punctuation. The extraction of OPM relations is performed with these templates in two modes:

1. Constrained extraction, which is limited only to the pairs of categories defined as OPM things in the manual editing process, systematically generates couples and attempts to detect any possible relation between them in the text, and

2. Unconstrained extraction, which allows selection of any single OPM thing and seeks all possible relations in which it occurs.

5.4 Automatic Generation of OPL Sentences

Since each template has a corresponding OPL formulation, every extracted natural language sentence can be straight-forwardly translated into an OPL sentence. Nonetheless, at this stage it is also possible to reformulate the outcome in order to better reflect the underlying relations. This transformation is performed in two steps:

Table 3 The OPL paragraph describing the GRACE system whose OPD is in Figure 3.

Search Results consists of **Actual Documents.**
Knowledge Domain consists of **Content Sources.**
EDG Application Layer consists of **Job Management Element** and **Data Management Element.**
Data Management Element retrieves **User Profile.**
Data Management Element and **Document Storage Service** are interfaced**.**
Data Management Element and **Search Engine** are interfaced**.**
NDF Repository consists of **Documents NDF.**
Documents NDF are transferred to **Document Storage Service.**
Document Processing Service processes **Actual Documents.**
Frontend Application transfers **Query Request.**
Web Server and **Frontend Application** are interfaced**.**
Text Indexing requires **Search Engine.**
Query Routing consumes **Query Request, Internal Content Sources,** and **External Content Sources.**
Query Routing yields **Actual Documents.**
Downloading requires **Document Processing Service.**
Downloading consumes **External Content Sources.**
Downloading yields **Actual Documents.**
Cashing consumes **Actual Documents** and **External Content Sources.**
Cashing yields **Content Sources.**
Storing consumes **Actual Documents.**
Storing yields **Internal Content Sources.**
Retrieving requires **Query Request.**
Retrieving yields **Search Results** and **Actual Documents.**
Accessing requires **EDG Application Layer.**
Accessing affects **NDF Repository** and **Document Repositories.**

1. A custom relation is transformed into a process, for example: **cached into** is transformed into **Caching**, and
2. A complex relation, such as **Actual Documents Cached into Document Repositories**, is transformed into two equivalent simple sentences. In our case, (1) **Caching** requires **Actual Documents** and (2) **Caching** yields **Document Repositories**.

These transformations do not modify the underlying semantics of the NL sentences but allow the complex natural language formulations to be and simplified into concise OPL sentences. The output set of the OPL sentences is listed in Table 3.

5.5 Manual Editing of the Results

The OPL sentences were fed into OPCAT one by one to obtain the OPD, which is shown in Figure 3 after manual beatification.

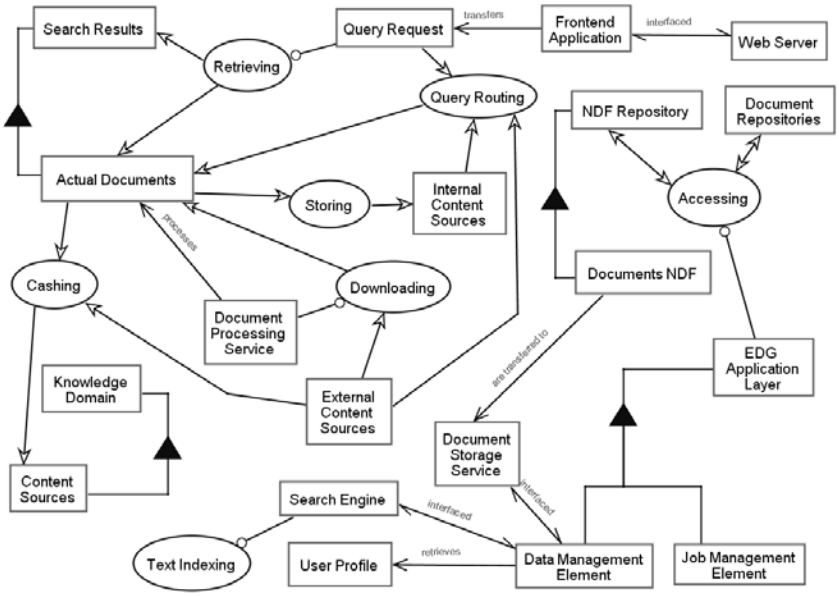

Figure 3 The OPD that represents the OPL sentences generated from GRACE free text

Both the OPL sentence set and the OPD are significantly simpler and more digestible than the hundreds of NL documentation pages from which the model was extracted. OPCAT allows the results to be edited graphically in order to remove the incorrect relations, organize the things and the relations into more complex (multi-layered) structures, add undetected things and relations, etc. The graphic manipulation is much easier than text editing, and this ability is a great advantage of OPCAT. Since complete equivalence between OPD and OPL presentations is granted, every modification in the OPD is automatically reflected in the corresponding OPL sentence(s). Several operations were applied to the results at this final step:

1. ***Corrections***: Some non-semantic corrections were necessary due to the fact that the extraction did not depict all of the existing or implied relations. These corrections fall into the following categories:
   2) Grouping of specialized elements into a general one (e.g., **Internal Content Sources** and **External Content Sources** were grouped into **Content Sources**),
   3) Associating unrelated elements (e.g., **Text Indexing** was associated with the **Document Processing Service**),
   4) Renaming elements (e.g., **Storing** was renamed more specifically as **Grid Publishing**),
   5) Reapplying a relation transitively from a general object to its specialization or from a whole to a part (e.g., transferring the instrument link attached to **Text Indexing** from **Search Engine** to its **Document Processing Service** part).

2. ***Additions and Eliminations***: Unlike corrections, additions and eliminations may semantically modify the original output. Additions aim primarily at improving the detail level and completing the implied structure based on common sense (e.g., by

introducing **User** as the human agent that interacts with the system). Eliminations simplify the results by removing superfluous or unessential detail.
3. ***Scaling***: Scaling was applied in order to simplify the results without losing details. Inspecting the OPD revealed that the documentation implicitly discusses two main processes: (1) storage of documents into content sources and (2) their retrieval on demand. The first process was conveniently renamed **Grid Publishing** and the second—**Information Retrieval**. Figure 4 presents the system diagram (SD)—the top-level view that resulted from abstracting the original results.

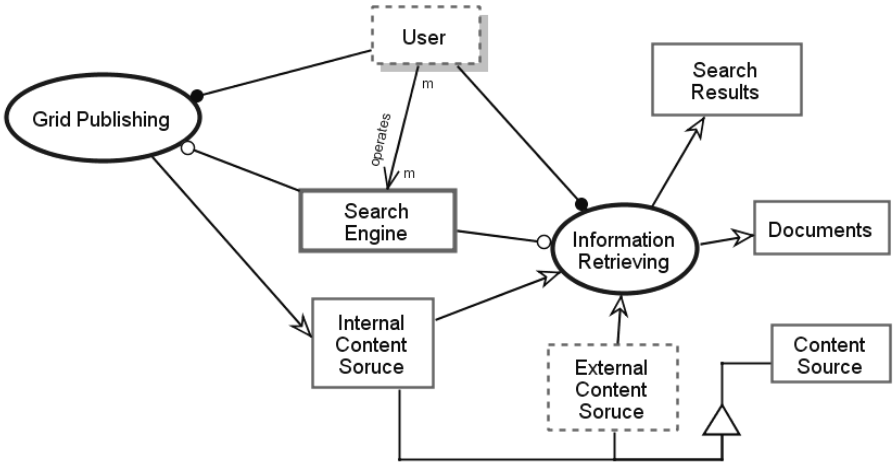

Figure 4 Manually abstracted system diagram of GRACE

From here the editing process that was demonstrated at the top level proceeded mostly through transferring the extracted things and relations to the most appropriate level of detail. The final result consists of seven OPDs at three levels of detail.

6 Summary and Conclusions

The experiment described in this paper demonstrates the feasibility of automating the most critical step in the system engineering process from unstructured business specification and user requirements to precise and formal system specifications. The experiment was designed as a proof-of-concept offering the first hands-on experience required for the development of a future full-scale industrial application. We drew the following be conclusions from the experiment:
1. The proposed methodology significantly reduces the quantity of material that would otherwise need to be processed manually.
2. Translating the original NL sentences into OPL reduces the initial level of conceptual complexity. The variety in which a relation may be expressed in NL may be surprisingly broad, leading to confusion, imprecision, and vagueness. This is typical if the documentation was written by many authors from various professional

backgrounds. OPL, on the other hand, introduces uniformity, which guarantees that the relations are expressed in a concise and unambiguous way.

3. The results depend critically on the quality of the processed documentation. The more architectural information is contained in it, the better the results. Relevant system components were often successfully extracted from the text as categories, but very little information regarding their relations with other system components was actually available. Obviously, no system can extract information that is not there.

4. Even when the results still require significant editing, it is so much easier to understand and manipulate the dual OPM graphic or even textual presentations than to work directly with the NL sources.

5. The quality, accuracy, and conciseness of the system architecture obtained following the SMART process is likely to be higher than that obtained through traditional model construction due to the discipline OPM introduces.

In order to become more useful, SMART needs significant improvements, in particular more sophisticated extraction templates and improved performance. Having provided a proof-of-concept to the viability of automated extraction of system model from free text, future research and development efforts will focus on enhancing the level of automation of SMART and testing it against traditional model construction processes in terms of both model quality and resource expenditure.

References

[1] D. Dori, Object-Process Methodology - A Holistic Systems Paradigm, Springer Verlag, Berlin, Heidelberg, New York, 2002

[2] D. Dori, I. Reinhartz-Berger, and A. Sturm, Developing Complex Systems with Object-Process Methodology using OPCAT. Lecture Notes in Computer Science (2813), pp. 570-572, 2003.

[3] GRACE: Grid Search and Categorization Engine. EU RTD Project in the 2002 Fifth Framework. http://www.grace-ist.org/

[4] B. Nuseibeh and S. Easterbrook. Requirements Engineering: A Roadmap. Proc. Conference on The Future of Software Engineering, Limerick, Ireland, 35-46, 2000.

[5] A. Egyed, P. Grünbacher, and N. Medvidovic. Refinement and Evolution Issues in Bridging Requirements and Architectures - The CBSP Approach. Proc. 1st International Workshops From Requirements to Architecture, co-located with ICSE'01, Toronto, Canada, 2001

[6] P. Grünbacher, A. Egyed, and N. Medvidovic. Reconciling Software Requirements and Architectures: The CBSP Approach. Proc. 5th IEEE International Symposium on Requirements Engineering (RE'01), Toronto, Canada, 2001.

[7] P. Grünbacher, A. Egyed, and N. Medvidovic. Dimensions of Concerns in Requirements Negotiation and Architecture Modeling. The second workshop on multi-dimensional separation of concerns in software engineering, co-located with ICSE'2000, Limerick, Ireland, June, 2000.

[8] W. Robinson, and S. Fickas. Automated Support for Requirements Negotiation. Proc. AAAI-94 Workshop on Models of Conflicts on Conflict Management in Cooperative Problem Solving, 1994.

[9] J. Ralyte, C. Rolland, and V. Plihon. Method Enhancement by Scenario Based Techniques. Proc 11th Conference on Advanced Information Systems Engineering (CAiSE'99), Heidelberg, Germany, 1999.
[10] C. B, Achour, Linguistic Instruments for the Integration of Scenarios in Requirements Engineering. Proc. 3rd International Workshop on Requirements Engineering: Foundations of Software Quality (REFSQ'97), Barcelona, 1997.
[11] C. Rolland and C. Proix. Natural Language Approach for Requirements Engineering. Proc. 4th International Conference on Advanced Information Systems Engineering-CAiSE'92, Springer-Verlag, Manchester, 1992.
[12] W. Paik, S. Yilmazel, E. Brown, M. Poulin, S. Dubon, and C. Amice, Applying Natural Language Processing (NLP) Based Metadata Extraction to Automatically Acquire User Preferences, Knowledge Capture - K-CAP'01, 2001.
[13] S. Si-Said, C. Roland, and G. Grosz. MENTOR: A Computer Aided Requirements Engineering Environment. Proc. 8th International Conference on Advances Information System Engineering, CAiSE'96, Greece, May, 1996 (Lecture Notes in Computer Science, 1080). Springer, 1996
[14] V. Ambriola and V. Gervasi. Processing Natural Language Requirements. Proc. 12th IEEE Conference on Automated Software Engineering (ASE'97). IEEE Press, 1997.
[15] K. Ryan, The Role of Natural Language in Requirements Engineering. Proc. IEEE International Symposium on Requirements Engineering, San Diego, 1993.
[16] L. Goldin, and D.M. Berry. A prototype Natural Language Text Abstraction Finder For Use In Requirements Elicitation. Automated Software Engineering Journal 4, (4) 375-412, 1997.
[17] B. Macias, and SG. Pullman. Natural Language Processing for Requirements Specification. Safety-Critical Systems. Chapman and Hall: London, 57-59, 1993.
[18] A. Fantechi, S. Gnesi, G. Ristori, M. Carenini, M. Vanocchi, and P. Moreschini. Assisting Requirement Formalization by Means of Natural Language Translation. Formal Methods in System Design, 4(3), 243-263. 1994.
[19] N. Juristo, A.M. Moreno, and M. Lopez. How to use Linguistic Instruments for Object-Oriented Analysis. IEEE Software; 17(3), 80-89, 2000.
[20] B. Macias, and S.G. Pullman. A Method for Controlling the Production of Specifications in Natural Language. The Computer Journal, 38(4), 310-318, 1995.
[21] R. Nelken and N. Francez. Automatic translation of natural-language system specifications into temporal logic. Proc. 8th Conference on Computer Aided Verification (CAV'96), Lecture Notes in Computer Science (1102) 360-371, 1996.
[22] M. Jarke, J. Bubenko, C. Rolland, A. Sutcliffe, and Y. Vassiliou. Theories Underlying Requirements Engineering: An Overview of NATURE at Genesis. In: Proc.1st IEEE Symposium on Requirements Engineering, San Diego, 1993.
[23] A.G. Sutcliffe and N.A.M. Maiden. Use of Domain Knowledge for Requirements Validation. Proc.Conference on Information System Development Process, 1993.
[24] F. Fabbrini, M. Fusani, V. Gervasi, S. Gnesi, and S. Ruggieri. Achieving Quality in Natural Language Requirements. Proc. 11th International Software Quality Week, 1998.
[25] C. Huyck and F. Abbas, Natural Language Processing and Requirements Engineering: a Linguistics Perspective, Proc 1st Asia-Pacific Conference on Software Quality, 2000.
[26] J. Natt, B. Regnell, P. Carlshamre, M. Andersson, and J. Karlsson, A Feasibility Study of Automated Natural Language Requirements Analysis in Market-Driven Development, Requirements Engineering 7, 20-33, 2002.
[27] V. Gervasi, and B. Nuseibeh, Lightweight Validation of Natural Language Requirements. Proc. 4th IEEE International Conference on Requirements Engineering (ICRE), Schaumburg, Il, 2000.

[28] M. R. Genesereth, Knowledge Interchange Format (KIF), http://logic.stanford.edu/kif/kif.html, 1998.
[29] T. Finin and R. Fritzson, KQML as an Agent Communication Language, Proc. 3rd International Conference on Information and Knowledge Management (CIKM'94), ACM Press, 1994.
[30] H. Helbig and S. Hartrumpf, Word Class Functions for Syntactic-Semantic Analysis, Proc. 2nd International Conference on Recent Advances in Natural Language Processing, pp. 312-317, 1997.
[31] Samuelsson, C., Optimizing Analysis and Generation in Natural Language Processing, Computational Lingustics – ERCIM, 1996.