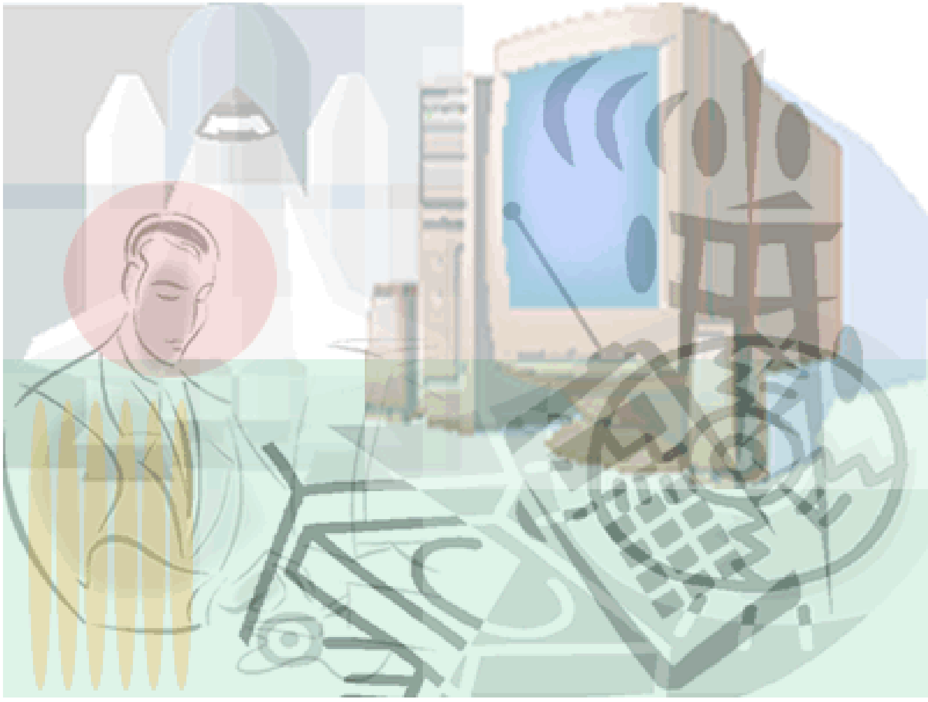
IECI Chapter Japan Series Vol. 4 No. 1, 2002 ISSN 1344-7491

Proceedings of the IECI Japan Workshop 2002

IJW-2002

March 3rd, 2002
Tokyo Institute of Technology



Supported by
Indonesian Society on Electrical, Electronics, Communication and Information (IECI)
Indonesian Students Association (PPI)

Organized by
Indonesian Society on Electrical, Electronics, Communication and Information (IECI) Japan

In Cooperation With
Green Digital Press ™

Toward a Method for Eliciting Software Requirements Using Constraint Natural Language

Romi Satria Wahono

Dept. of Information and Mathematical Sciences, Saitama University
Indonesian Institute of Science (LIPI)

Abstract:
Requirement elicitation is considered as one of the most important activities in software development. Most of the faults found during testing and operation result from poor understanding or misinterpretation of requirements. Although there are many techniques focusing on the requirements elicitation, there are only a few focusing on the formalization of object-oriented features and the methodology for identifying and refining objects. We propose a methodology for eliciting requirements using constraint natural language based on object-oriented paradigms.

Keywords: requirements engineering, requirements elicitation

1. INTRODUCTION

Requirement elicitation is considered as one of the most important activities in software development. Most of the faults found during testing and operation result from poor understanding or misinterpretation of requirements. Until now, there are only a few effective methods and tools to guarantee a complete, consistent, and unambiguous requirement model [Lu00]. In the traditional approach to software analysis, system analyst interviews end-users to capture requirement.

We propose a methodology where user takes an active role in the requirements elicitation using constraint natural language, which is called object-based formal specification (OBFS).

2. THE THREE DIMENSIONS of REQUIREMENTS ENGINEERING

The requirements engineering is the first phase of software engineering process, in which user requirements are collected, understood, and specified. Requirements engineering is recognized as a critical task, since many software failures originate from inconsistent, incomplete or simply incorrect requirements specifications. A correct, consistent and complete way to collect, understand, specify and verify user requirements is important and necessary. The result of the requirements engineering phase is documented in the *requirements specification*. The requirements specification reflects the mutual understanding of the problem to be solved between the analyst and the client. The requirements specification serves as a starting point for the next phase, the design phase. To achieve well-defined document containing the user requirements that satisfies these prerequisites, we can distinguish three processes in requirements engineering [Loucopoulos-95]. These processes involve iteration and feedback (Figure 1).

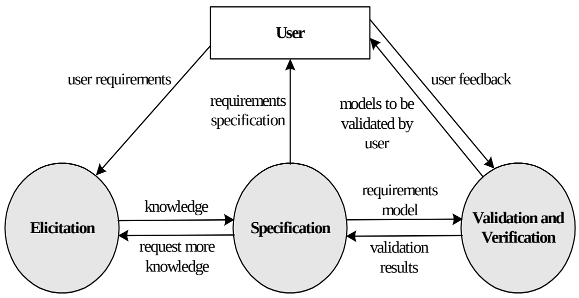


Figure 1. Requirements Engineering Process

- ### Requirements Elicitation

Requirements elicitation is about *understanding* the problem. In general, the requirements analyst is not an expert in the domain being modeled. Through interaction with domain specialists, he has to build himself a sufficiently rich model of that domain. The fact that different disciplines are involved in this process complicates matters. In many cases, the analyst is not a mere outside observer of the domain modeled, simply eliciting facts from domain specialists.

- ### Requirements Specification

Once the problem is understood, it has to be *described* in the requirements specification document. This document describes the product to be delivered, not the process of how it is developed.

- **Requirements Validation and Verification**

Once the problem is described, the different parties involved have to *agree upon* its nature. We have to ascertain that the correct requirements are stated (validation) and that these requirements are stated correctly (verification).

3. The Problems of Requirements Elicitation

Problems of requirements elicitation can be grouped into three categories [Christel-91]:

1. **Problems of scope**, in which the requirements may address too little or too much information.
   - The boundary of the system is ill-defined
   - Unnecessary design information may be given

2. **Problems of understanding**, within groups as well as between groups such as users and developers.
   - Users have incomplete understanding of their needs
   - Users have poor understanding of computer capabilities and limitations
   - Analysts have poor knowledge of problem domain
   - User and analyst speak different languages
   - Ease of omitting "obvious" information
   - Conflicting views of different users
   - Requirements are often vague and untestable, e.g., "user friendly" and "robust"

3. **Problems of volatility**, i.e., the changing nature of requirements.
   - Requirements evolve over time

4. OBFS as a Constraint Natural Language

We propose an approach where end-users take an active role in the analysis by eliciting requirements using OBFS. We use OBFS to guide end-users in describing their problem based on object-oriented paradigm. OBFS is composed of *Description Statements* (*DS*), *Collaborative Statements* (*CS*), *Attributive Statements* (*AS*), *Behavioral Statements* (*BS*), *and Inheritance Statements* (*IS*). OBFS use English natural language based on the *constraint syntax rules*.

4.1. Description Statements (DS)

DS is used to guide the writing of an overview of the system that one wants to build. *DS* is composed from four elements: *Requirement ID*, *Requirement Name*, *Language*, and *Description*. *DS* should specify what is to be done, but not how it is to be done. It should be a statement of needs, not a proposal for a solution.

4.2. Collaborative Statements (CS)

CS is used to identify objects, and associations between the objects. *CS* consists of a set of forms and contains *Subject-Verb-Object (S-V-O)* as well as the English natural language based on *CS syntax rules* (*E*). We use S_{cs}-V_{cs}-O_{cs} notation for describing S-V-O natural language, which is based on *CS syntax rules*. The collaboration between S_{cs} and O_{cs} must be described in the *CS*.

The *CS syntax rules* are listed as follows. *Predicates* are extracted from synonym data dictionary (thesaurus) [Chapman-92].

$$\langle \textbf{ActionSentence(AcS)} \rangle ::= S_{cs} \langle AcSPredicate \rangle O_{cs}$$
$$\langle \textbf{AcSPredicate} \rangle ::= drive|work\ for|maintain|manage|own|execute|serve|use$$
$$\langle \textbf{LocationSentence(LcS)} \rangle ::= S_{cs} \langle LcSPredicate \rangle O_{cs}$$
$$\langle \textbf{LcSPredicate} \rangle ::= next\ to|goto$$
$$\langle \textbf{CommunicationSentence(CmS)} \rangle ::= S_{cs} \langle CmSPredicate \rangle O_{cs}$$
$$\langle \textbf{CmSPredicate} \rangle ::= talk\ to|communicate\ with|refer\ to$$

The objects and its associations can be identified by using the following formulas.

$$\forall CS \in E\ [S_{cs} \Rightarrow OBJ_t]\ and\ \forall CS \in E\ [O_{cs} \Rightarrow OBJ_t] \cdots (1)$$
$$\forall CS \in E\ [V_{cs} \Rightarrow ASS_t] \cdots (2)$$
$$\neg(OBJ_t)_{\text{red}} \Rightarrow OBJ\ and\ \neg(ASS_t)_{\text{red}} \Rightarrow ASS \cdots (3)$$

X_t=tentative X, Y_{red}=redundant Y, OBJ=object, ASS=association

4.3. Attributive Statements (AS)

AS are used to identify the attributes of objects. Attributes are properties of individual objects. Attributes usually correspond to nouns followed by possessive phrases, and sometimes are characterized by adjectives or adverbs. *AS* must contain properties of each object identified at the previous step. *AS* consists of a set of forms and contains S_{as}-V_{as}-O_{as} as well as the English natural language based on *AS syntax rules* (*E*).
The *AS syntax rules* are listed as follows.

$$\langle \textbf{OwnershipSentence(OwS)} \rangle ::= S_{as} \langle OwSPredicate \rangle O_{as}$$
$$\langle \textbf{OwSPredicate} \rangle ::= has\ (properties)|consits\ of|contain\ of$$

The object attributes can be identified by using the following formulas.

$$\forall AS \in E\ [O_{as} \Rightarrow ATT_t]\ and\ \forall AS \in E\ [S_{as} \Rightarrow OBJ] \cdots (4)$$
$$\neg(ATT_t)_{\text{red}} \Rightarrow ATT \cdots (5)$$

X_t=tentative X, Y_{red}=redundant Y, OBJ=object, ATT=attribute

4.4. Behavioral Statements (BS)

BS is used to identify object behaviors. Behavior is how an object acts and reacts, in terms of state changes and message passing. A behavioral statement must contain behaviors of each object identified at the previous step. *BS* consists of a set of forms and contains S_{bs}-V_{bs}-O_{bs} as well as the English natural language based on *BS syntax rules* (*E*).

The *BS syntax rules* are listed as follows.

$$\langle \textbf{CapabilitySentence(CpS)} \rangle ::= S_{bs} \langle CpSPredicate \rangle O_{bs} \mid S_{bs} \langle CpSMinusPredicate \rangle O_{bs}$$
$$\langle \textbf{CpSPredicate} \rangle ::= \textit{has (a capability to)} \mid \textit{has (a capacity for)} \mid \textit{can (capabilities)} \mid \textit{able to (capabilities)}$$
$$\langle \textbf{CpSMinusPredicate} \rangle ::= \textit{has not (a capability to)} \mid \textit{has not (a capacity for)} \mid \textit{can not (capabilities)} \mid \textit{not able to (capabilities)}$$

The object behaviors can be identified by using the following formulas.

$$\forall BS \in E\ [O_{bs} \Rightarrow BEH_t]\ \textit{and}\ \forall BS \in E\ [S_{bs} \Rightarrow OBJ] \cdots (6)$$
$$\neg (BEH_t)_{\text{red}} \Rightarrow BEH \cdots (7)$$

X_t=tentative X, Y_{red}=redundant Y, OBJ=object, BEH=behavior

4.5. Inheritance Statements (IS)

IS is used to organize classes by using inheritance, to share common object attributes and behaviors. *IS* provide sentences that describe *is-a-kind-of* relationship. *IS* consists of a set of forms and contains S_{is}-V_{is}-O_{is} as well as the English natural language based on *IS syntax rules* (*E*).

The *IS syntax rules* are listed as follows.

$$\langle \textbf{InheritanceSentenceA(IhSA)} \rangle ::= S_{is} \langle IhSAPredicate \rangle O_{is}$$
$$\langle \textbf{IhSAPredicate} \rangle ::= \textit{is a kind of} \mid \textit{is spesialization of}$$
$$\langle \textbf{InheritanceSentence B(IhS B)} \rangle ::= O_{is} \langle IhSBPredicate \rangle S_{is}$$
$$\langle \textbf{IhSBPredicate} \rangle ::= \textit{is generalization of}$$

The object and its class hierarchy organization can be refined by using the following formulas.

$$\textbf{IhSA}\ \forall IS \in E\ [O_{is} \Rightarrow SCL_t]\ \textit{and}\ \forall IS \in E\ [S_{is} \Rightarrow OBJ] \cdots (8)$$
$$\textbf{IhSB}\ \forall IS \in E\ [S_{is} \Rightarrow SCL_t]\ \textit{and}\ \forall IS \in E\ [O_{is} \Rightarrow OBJ] \cdots (9)$$
$$\neg (SCL_t)_{\text{red}} \Rightarrow SCL \cdots (10)$$

X_t=tentative X, Y_{red}=redundant Y, OBJ=object, SCL=superclass

5. Concluding Remarks

It is argued that requirements elicitation is an ill-defined task. Although there are many techniques focusing on the requirements elicitation, there are only a few focusing on the formalization of object-oriented features and the methodology for identifying and refining objects. We presented a methodology for eliciting requirements using constraint natural language based on object-oriented paradigms.

6. REFERENCES

[Chapman-92] Robert L. Chapman, Roget's International Thesaurus, HarperCollins Publishers, 1992.

[Christel-91] Michael G. Christel and Kyo C. Kang, Issues in Requirements Elicitation, *Technical Report CMU/SEI-92-TR-12*, ESC-TR-92-012, September 1992.

[IEEE-729] Institute of Electrical and Electronics Engineers. IEEE Standard Glossary of Software Engineering Terminology. *ANSI/IEEE Standard 729-1983*, Institute of Electrical and Electronics Engineers, New York, 1983.

[IEEE-610.12] Institute of Electrical and Electronics Engineers, IEEE Standard Glossary of Software Engineering Technology, *IEEE Std 610.12-1990*, Institute of Electrical and Electronics Engineers, New York, 1990.

[IEEE-830] Institute of Electrical and ElectronicsEngineers, IEEE Recommended Practice for Software Requirements Specifications, *IEEE Std 830-1998*, Institute of Electrical and Electronics Engineers, New York, 1998.

[Loucopoulos-95] P. Loucopoulos and V. Karakostas: Software Requirements Engineering, *McGraw-Hill*, 1995.

[Lu-00] Ruqian Lu and Zhi Jin, Domain Modeling-Based Software Engineering, *Kluwer Academic Publishers*, 2000.

BIOGRAPHY of AUTHOR

Romi Satria Wahono, Received B.Eng. and M.Eng degrees in Information and Computer Sciences in 1999 and 2001, respectively, from Saitama University. He is currently a researcher at the Indonesian Institute of Sciences (LIPI), and a Ph.D. candidate at the Department of Information and Mathematical Sciences, Saitama University. The research fields of his interests are Software Engineering, Requirement Engineering, and Object-Orientation. He is a member of the ACM, IEEE Computer Society, The Institute of Electronics, Information and Communication Engineers (IEICE), Information Processing Society of Japan (IPSJ), Japanese Society for Artificial Intelligence (JSAI), and Indonesian Society on Electrical, Electronics, Communication and Information (IECI).