23

Tool support of orderly transition from informal to formal descriptions in requirements engineering

J. Xu, L. Jin and H. Zhu
Computer Software Institute, Nanjing University,
Nanjing, 210093, China, Tel.:+86 (25)6637551-3283.

Abstract
In the analysis and specification of user requirements, software engineers are often confronted with difficulties due to the complexity of the problem, the communication barriers between peoples of diverse backgrounds, the inconsistency and incompleteness of user's statement of requirements and frequent changes of user's requirements. This paper reports a tool that supports engineers to cope with these difficulties by automatic consistency and completeness checking and automatic generation of functional specifications.

Keywords
Requirements definition, functional specification, transformation, CASE tools, Z language

1 INTRODUCTION

The requirements analysis and specification are concerned with eliciting, clarifying and documenting user's requirements of a computation system and producing the corresponding functional specification. Many studies have shown that errors made at this stage are very costly (even impossible) to rectify. Neglected or only partially completed requirements analysis tends to lead to problems later in development. It is perceived as an area of growing importance. However, in the analysis and specification of user requirements, software engineers are often confronted with difficulties due to the complexity of the problem, the communication barriers between peoples of diverse backgrounds, the inconsistency and incompleteness of information and frequent changes of user's requirements.

To overcome these difficulties, the literature has advanced a number of proposals such as:

N. Terashima et al. (eds.), *Advanced IT Tools*
© Springer Science+Business Media Dordrecht 1996

- integrating multiple views and representations to soothe the communications of people; (System Designers, 1985; Nuseibeh *et al.*, 1994)
- modelling requirements engineering processes in various paradigms to provide guidelines for the development of requirements definitions; (Leonhardt *et al.*, 1995; Finkelstein & Potts, 1986)
- developing methods and software tools to support resolving conflict requirements (Feather & Fickas, 1991), coping with incompleteness and inconsistency (Bell *et al.*, 1977; Heimdahl & Leveson, 1995), automating the transformation of the informal to the formal (Fraser *et al.*, 1991), etc.;
- modelling software systems and their environments and employing domain knowledge and object-oriented methodology to manage requirements changes (Borgida *et al.*, 1985; Diaz and Arango, 1991).

We regard overcoming these difficulties as the major driving force of requirements analysis. They are the most important issues that a CASE tool for requirements engineering should address. Our ultimate research goal is to automate the process of requirements engineering. At present, the researches aim at developing software tools to support requirements analysis with the methods of current state of practice and to link requirements specification to the formal methods by generating formal functional specifications in well established specification languages such as Z (Spivey, 1992).

We can identify two kinds of supports to requirements analysis and specification: (a) the language support, and (b) the tool support. Language supports provide language facilities in which user's requirements are represented, expressed and communicated. They may help software engineers to cope with difficulties due to the complexity of the problem and communication barriers. Tool supports use software tools to perform or help to fulfil various tasks involved in requirements analysis and specification, such as the storage, management and retrieval of user's requirements, the analysis of expressed requirements, and the transformation of one representation into another. They may help to deal with incompleteness and inconsistency and frequent changes of user's requirements.

In the literature, there is a great number of tools and languages to aid in the production of requirements definitions and the generation of functional specifications. Among the most famous are:

- SREM (Bell *et al.*, 1977), which supports management and consistency checking of requirements written in a language RSL based on finite state machines;
- KBRA (Czuchry and Harris, 1988), which offers facilities for reasoning with requirements represented as a knowledge base through inheritance, automatic classification and constraint propagation;
- the work of Fraser *et al.*(1991) on semi-automatic generation of formal functional specifications in VDM from data flow diagrams.

There is a big gap between the informal descriptions in requirements definition and the formal functional specifications. User's initial requirements statement must be in informal or semi-formal representations such as in natural languages and diagrams, whilst any decent analysis of the requirements has to be based on a formal representation. Given the current state of the art of natural language understanding, it seems impossible to build a practical tool to bridge the gap between informal and formal descriptions. Therefore, we take a progressive

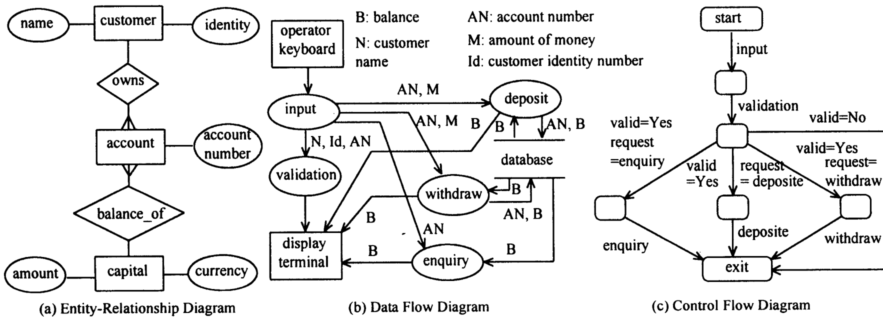
and orderly transition approach to requirements engineering. That is, the process of requirements elicitation, analysis, documentation and specification is divided into a sequence of interacting and iterating phases. It starts with an informal and vague requirements statement, which is then gradually transformed into a formal and consistent functional specification.

In this paper, we report a requirements definition language NDRDL and its requirements analysis support system NDRASS, which support the progressive and orderly transition process of requirements analysis and specification.

2 THE NDRDL LANGUAGE

The NDRDL language (Dong *et al.*, 1995) is based on the classic methods of structured analysis (Yourdon, 1989). A requirements definition in NDRDL has quite standard hierarchical structure, which consists of four parts:

- *an introduction* to the background to user's requirements in natural language;



Data name	Description	Form	Constraint
customer name	the name of a customer	String	
bank database	the database for storing information about customers and accounts.	Set (Record Customer: customer; Account : account_number; Balance: real End)	Balance ≥ 0

(e) Data dictionary

Relationship	Entities	Description	Definition
owns	customer, account	Owns(John, 210093) means that John owns the account 210093.	owns(c,ac) $\Leftrightarrow$ ($\exists$r $\in$ database. (ac = r.account) $\wedge$ (c = r.customer))

(f) Relationship dictionary

Op.	Input	Output	Description	Definition
validate	cn: customer name; cid: customer identity; acn: account number;	valid: Bool	validate the personal information of a customer against the database.	valid = ($\exists$r $\in$ database.((acn = r.account) $\wedge$(cid = r.customer.identity) $\wedge$(cn = r.customer.name))

(g) The operation dictionary

Figure 1 Example of diagrams and dictionaries in NDRDL.

- *a general description* of functions, user characteristics, restrictions and environment;
- *the requirements details,* which are further divided into two parts: one for functional requirements, and the other for non-functional requirements. The former consists of three parts: (a) a list of functions in informal representation; (b) a set of diagrams including an entity-relationship diagram (ERD), a data flow diagram (DFD) and a control flow diagram (CFD); (c) a set of dictionaries including a data dictionary, a relationship dictionary and an operation dictionary to provide definitions of the terminology. Non-functional requirements, such as goals and constraints, are also expressed in natural language.
- *appendix and index,* which are also in informal representation.

Figure 1 gives the diagrams and segments of dictionaries of an example requirements definition in NDRDL. This example will also be used later in the paper to illustrate the transformation from requirements definitions to functional specifications.

Due to the redundancy among the diagrams, inconsistency and incompleteness may occur. The definitions of the data, relationships or operations in the dictionaries may also be inconsistent with their uses in the diagrams. Therefore, some constraints on them are imposed to obtain complete and consistent requirements definitions; see Table 1.

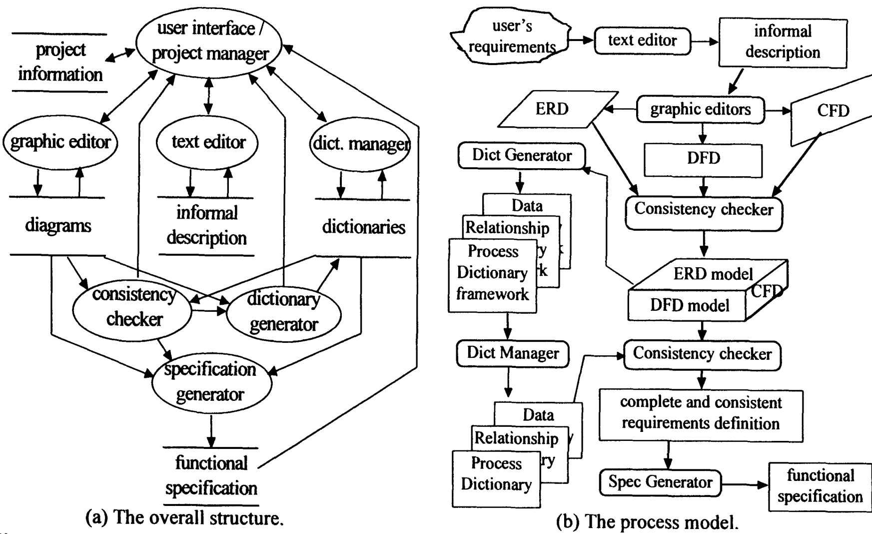
Table 1 Completeness and consistency constraints of NDRDL.

Views	Constraint
DFD/ ERD	The collection of data in a DFD must be the same as the collection of data represented as the entities or their attributes in the corresponding ERD.
CFD/ DFD	The set of processes associated with the arcs in a CFD must be the same as the set of processes in the corresponding DFD.
	Any sequence of events in a CFD must satisfy the permissible condition.
CFD/ ERD	Any data used in a CFD must be contained in the collection of data in the corresponding ERD.
ERD/ DD	Every entity in an ERD must be defined in the data dictionary. If an entity has a set of attributes, the definition of the entity in the data dictionary must also specify the attributes consistently.
ERD/ RD	Every relationship in an ERD must be defined in the relationship dictionary with the same participant entities.
DFD/ PD	Every process in a DFD must be defined in the operation dictionary that the signature of the operation must be consistent with the data flowing inwards and outwards the process node.

3 THE NDRASS SYSTEM

NDRASS system is a requirements analysis support system. As shown in Figure 2(a), it provides the following facilities:

- **A text editor** for the edition and modification of texts in natural language ;
- **Graphic editors** for the edition and modification of various diagrams;
- **Managers of dictionaries** for the edition, modification, browse and management of dictionaries;
- **An automatic checker** for the check of consistency and completeness among dictionaries and diagrams;
- **Two automatic generators**: one for generation of frameworks of dictionaries; the other for the generation of formal functional specifications in Z.



(a) The overall structure.

(b) The process model.

Figure 2 The NDRASS system.

As illustrated in Figure 2(b), a typical requirements analysis process that NDRASS supports starts with production of an informal description of user's requirements. This is supported by a text editor. The second step is the construction of semi-formal models of required system. This is supported by the graphic editors of NDRASS. Once consistence between the diagrams is checked, the dictionary generator can be invoked to generate frameworks of the dictionaries. A framework of data dictionary consists of all the data names used in the diagrams, their data structure according to the ERD. The fields of informal description and constraints are left to be filled by requirements engineer manually. A framework of relationship dictionary consists of the names of the relationships appeared in ERD, and the entities involved in the relationship. The fields of informal description of the relationship and formal definition of the relationship are left to be filled in manually. The framework of operation dictionary consists of the names of the processes appeared in the data flow diagram, and the input, output of the process. The fields of informal description and formal definition of the process are left to be filled in manually. The completion of the dictionaries are supported by the dictionary manager. Once the dictionaries have been completed, the consistence between the diagrams and dictionaries can be checked, and then a formal functional specification in Z can be automatically generated.

4 TRANSFORMATION INTO FUNCTIONAL SPECIFICATION

This section will focus on the automatic generation of functional specifications in Z from consistent and complete requirements definitions in NDRDL. The Z language provides schemas to modular descriptions of the state space and the operations and functions of a software system. Readers are referred to (Spivey, 1992) for the Z notations.

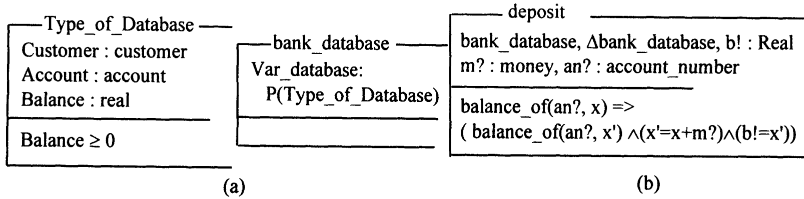


Figure 3 (a) Data store schema; (b) operation schema.

4.1 The generation of state space descriptions

Two types of schemas, *entity schema and relationship schemas,* are generated according to the information contained in ERD, data dictionary and relationship dictionary. For each entity in the ERD, an entity schema is generated such that

(A) the name of the schema is the entity name;

(B) for each attribute *attr* of type *T* of the entity, a declaration *attr:T* is included in the declaration part of the schema;

(C) if the entry of the entity in the data dictionary contains a predicate to describe the restrictions on the values of the entity, the predicate is copied into the predicate part of the schema with some syntactical transformations.

Entity schemas are used as types of state variables, parameters, input and output of functions and operators, and the types of attributes of other entity schemas. The system state space is determined by the data stores contained in the DFD. For each data store DS in the DFD, a schema is generated to define the components of the data store. For example, the schemas (b) in Figure 3 are generated for the *database* in the DFD of Figure 1. A relationship schema defines a relation. It is generated for each relationship R in the ERD. The generation of these schemas is similar to that of entity schemas. Readers are referred to (Xu *et al.*, 1995) for details.

4.2 The generation of function/operation definitions

The definitions of functions and operations are generated according to the information contained in the DFD and operation dictionary. For each process in DFD, an operation schema is generated according to the following rules.

(A) The name of the schema is the name of the process;

(B) For each inward dataflow that does not come out of a data store, "X? : TX" is included in the declaration part for the data X of type TX associated with the flow;

(C) For each outward data flow that does not go into a data store, "Y! : TY" is included in the declaration part for the data Y of type TY associated with the flow;

(D) If a data store DS has data flowing into the process node, the DS schema is included into the operation schema; if there are data flowing from the process node into a data store DS, the DS schema is included with decoration Δ;

(E) The predicate P in the operation dictionary is transformed into P' in Z notation and included in the predicate part. In addition to the syntactical transformations, variables in P must also be systematically decorated according to the following rules:

(a) for each input variable x, if it is associated with an inward data flow coming out from a data store, it is unchanged. Otherwise, x is replaced with x?;
(b) for each output variable y, if it is associated with an outward data flow going to a data store, it is replaced with y'. Otherwise, it is replaced with y!.

For example, Figure 3(b) is the schema for the *deposit* operation.

4.3 The generation of system operation structure

In Z language, a software system is described as a function on the state space. This function will be generated according to the CFD.

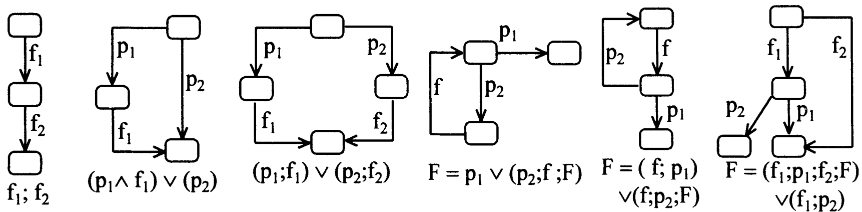


Figure 4 Examples of prime flow graphs.

The generation of system control function is based on Fenton *et al.*'s theory of structured programming (Fenton *et al.*, 1985) to improve the readability of generated Z code. According to the theory, every flow graph can be uniquely decomposed into a set of prime graphs so that it is the composition the prime graphs by the nest and composition operations. Figure 4 gives some examples of prime graph which correspond to control structures. The generation process consists of the following three steps.

(A) A flow graph is normalised so that it contains only one start node and one exit node and every node has at most two outward arcs;
(B) The flow graph is decomposed into prime graphs such that the flow graph is represented as a decomposition tree. Given a flow graph, the decomposition starts with finding prime flow graphs. A prime flow graph is then reduced to an arc from the start node to the exit node of the prime sub-graph. Such a reduction process is recorded and a decomposition tree is constructed;
(C) The Z description of the flow graph is generated according to the decomposition tree.

NDRASS selects a set of prime flow graphs as well structured CFD. Such prime flow graphs have well structured and readable Z descriptions as shown in Figure 4. Prime flow graphs not in the set are considered as not well structured control structures. Once such a prime flow graph is detected, the user is warned and asked if modification will be made. If no modification is made, a recursive Z description of the prime flow graph will be generated. The distinction of well structured from not well structured sub-graphs enables us to control software complexity at requirements engineering stage and helps quality control.

5 CONCLUDING REMARKS

The progressive and orderly transition approach to requirements engineering is characterised by a step by step transition from informal to semi-formal, and finally, into formal descriptions. This approach is supported by the NDRDL language and the NDRASS system. Once a complete and consistent requirements definition is obtained, a formal functional specification in Z can be automatically generated by NDRASS system. NDRASS system has been implemented on Sun Workstation Sparc 490 at the Institute of Computer Software at Nanjing University.

6 REFERENCES

System Designers, (1985) *CORE -- The method*, Systems Designers Scientific, Issue 1.0.

Nuseibeh, B., Kramer, J. and Finkelstein, A. (1994) A framework for expressing the relationships between multiple views in requirements specification, *IEEE TSE*, 20(10), 760~773.

Leonhardt, U., Kramer, J., Nuseibeh, B. and Finkelstein, A. (1995) Decentralised process modelling in a multi-perspective development environment, *Proc. of 17'th ICSE*, 255-264.

Finkelstein, A. and Potts, C. (1986) Structured common sense: the elicitation and formalization of requirements, In *Software Engineering'86* (eds Barnes, D. and Brown, P.), Peter Peregrinus, 236-250.

Bell, T. E. Bixler, D. C. and Dyer, M. E. (1977) An extendible approach to computer-aided software requirements engineering, *IEEE TSE*, SE-3, 849-860.

Fraser, M. D., et al. (1991) Informal and formal requirements specification languages: bridging the gap, *IEEE TSE*, 17(5).

Borgida, A., Greenspan, S. and Mylopoulos, J. (1985) Knowledge representation as the basis for requirements specifications, *IEEE TSE*, 18, 82-91.

Prieto-Diaz, R. and Arango, G. (1991) *Domain Analysis and Software Systems Modelling*, IEEE Computer Society.

Yourdon, E. (1989) *Modern Structured Analysis*, Prentice-Hall, New Jersey.

Czuchry, A.J., and Harris, D.R. (1988) KBRA: a new paradigm for requirements engineering, *IEEE Expert*, 3, 21~35.

Spivey, J. M. (1992) *The Z notation -- A Reference Manual*, Second Edition, Prentice Hall.

Dong, L., Fei, Z. Zhu, H. and Jin, L. (1995) The software requirements definition language NDRDL, *J. Computer Science*. (In Chinese)

Xu, J., Zhu, H., et al. (1995) From requirements definition to formal functional specification -- an automatic transformational approach, *Science in China*, 38(Supp).

Feather, M. S. and Fickas, S. (1991) Coping with requirements freedom, in *Proc. International Workshop on Development of Intelligent Information Systems*, Niagara-on-the-lake, Canada, 42-46.

Heimdahl, M. P. E. and Leveson, N. G.(1995) Completeness and Consistency analysis of state-based requirements, in *Proc. of 17'th ICSE*, 3-14.

Fenton, N. E., Whitty, R. W. and Kaposi, A. A.(1985) A generalised mathematical theory of structured programming, *Theoretical Computer Science*, 36, 145-171 .