An Ameliorated Methodology for the Abstraction of Object Oriented Features from Software Requirements Specification

A. M. Shivarama, Dr. Shivanand M. Handigundb

a,bBangalore Institute of Technology, K. R. Road, V. V. Puram, Bengaluru, 560004, India

Abstract

The Business Process (BP) requirements is specified in the form of software requirements specification (SRS). This SRS serves as a base for software development. The software needs to be developed in a syllogized software development life cycle (SDLC) stages. The SRS which denote the requirements of BP is used as input to the analysis stage from which the paradigm dependent components are to be abstracted. Hither the components are abstracted manually with the exception of hiatuses of semi-automated methods for few of the components. The SRS is construed with reference to the specific paradigm through which the design is to be developed. Unfortunately, no complete automated methodology exists for the abstraction of all paradigm dependent components. The automated methodology eliminates the possible human errors which would have ripple effect to damage the entire software. This paper develops an innovative, unique methodology to resurrect the SRS statements as modular statements. Further develops an automated methodology for abstraction of control and data flow graphs. Further it develops an automated methodology for abstraction of useful components required for the class diagram. The class diagram emphasizes both structural and behavioral aspects. This facility is effectively used to abstract object class attributes, object methods, visibility, signature, return type etc.

Information systems are developed through software projects which use the specific software requirements specification (SRS) provided for them as the base. The SRS contains details of the information system through which appropriate software can be developed. The information systems are also viewed perceptively with different pragmatics like work, work process or usecase. The usecase is one of the prime perspective views whose sum forms the information system. In this paper, an attempt is made to abstract object class, object class attributes, object methods, interrelationships between object classes and starting with/ending with actor from unformatted, unstructured SRS text. This is achieved through our own developed classing and slicing techniques to identify respectively the class structure & object methods. Then usecase is developed through the interrelationships between object methods of different classes and start with or end with the actor. The stages involve moulding the SRS, designing control flow graph for the SRS & data flow table for the SRS statements, developing appropriate classing and slicing criteria, creating actor & actors’ interface attributes table, create slicing criteria for each usecase and then slice relevant statements for each usecase. Here, we have attempted to use Weiser modified algorithm1 to abstract exact usecase. The slicing criterion is obtained through the intersection of actor’s interface attributes and the referenced / defined attributes present between two consecutive actors. Attempts have been made to resolve synonyms & heteronyms present in the SRS. The correctness & completeness of the proposed methodology depends on the realization of actor & actors’ interface attributes.
© 2015 Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Peer-review under responsibility of organizing committee of The 2015 International Conference on Soft Computing and Software Engineering (SCSE 2015)

**Keywords:** Software Requirements Specification (SRS), Control Flow Graph (CFG), Data Flow Graph (DFG), Control Flow Table (CFT), Data Flow Table (DFT), Use Case, Classing Technique, Slicing Technique, Object Method, Class Diagrams.

---

1. Introduction

**Vision:** To develop an automated methodology for the abstraction of usecases from SRS.

**Mission:** To construe the definition of usecase to identify components and develop automated methodology for their abstraction.

**Objectives:**

1. To construe the definition of usecase definition with logical positivism.
2. To develop automated methodology for the abstraction of needy components of usecase from SRS.
3. To eliminate the redundancy with the abstraction of usecase hierarchies from the SRS

**Motivation:**

The SRS is designed by the strategic management through the collection of their requirements. The manual accumulation of requirements has the following disadvantages.

Firstly, the requirements may suffer from incompleteness. Secondly, the presence of synonymous and heteronymous words exists in SRS due to the organizational, multicultural and individual requirements. Finally, enormous presence of syntactics and semantics in the English language enhances flexibility. All the three have compelled abstraction of usecase a human dependent. As a result, the abstraction of components depends on individual human skills. This puts the correctness, completeness & efficiency in cul-de-sac position. Therefore, there is a need to develop an automated methodology for the abstraction of usecases. BP is a complex entity. Usecase is one way of analysing the BP. The different people view the same BP differently. These different views are called perspective views. One of the perspective views is usecase view. Ivar Jacobson developed a usecase which is abstract one, in the UML the three amigos redefined it as complete one but still the definition of usecase is a human skill dependent. As a result the common interaction with different users is a herculean task. There is a need for a standard definition of usecase in this connection Shivanand et al have defined usecase as “A sequence of activities performed by the system for an actor on an occasion”. The present paper attempts to abstract all the components necessary for a usecase and uniformity of this abstraction is automated one and it eliminates the human dependency.

1.1 Literature Survey

Ivar Jacobson has developed usecase as an abstract interaction between the user and the system. Later, three modelists viz. Ivar Jacobson, Grady Booch and James Rumbaugh have developed a unified modelling language comprising of different design diagrams. Amongst them one of the design diagrams is usecase diagram. They defined the usecase as a complete interaction of the user with the system. They have not distinguished the user and the actor. As per their definitions usecase is a complete interaction of the user with the system. Since, they have not distinguished between the different stakeholders, the definition remains vague and without any granulation. This makes difficult for communication and representation. Handigund et al have classified between the inside users as object class and outside users as actors. Again authors have refined the definition of usecase with granulation of the activity.

Information system = $\sum_{i=1}^{\mu} \text{Use case}$ whereas $\text{Use case}_i = \sum_{j=1}^{\nu} \sum_{k=1}^{\omega} A_{ijk}$

Where $j$ take the value from all the work Process and $k$ takes the value from all works and $A_{ijk}$ represents the activity common to $j^{th}$ work process and $k^{th}$ work with $i^{th}$ usecase. These equations make well defined granulation using the...
first cut usecase from the statements following the referenced actor, preceding another consecutive defined actor also preceding the defined actor following the next consecutive actor. This first cut usecase is further refined incorporating the parameters passed to this usecase.

1.2 Taxonomy

This taxonomy is necessitated because, most of the computer scientist may be ignorant about the mathematical empiricism. Moreover the slicing technique needs the data and control flow graphs. Nowadays most cognoscenti people ignore the basic needs and develop highly sensitive technology over weak foundation. Moreover, the au-courant definitions of UML components are vague. In addition, there is no uniqueness in definitions. Therefore we attempted to present the taxonomy here.

- **Noun phrase**: The noun phrase is one, in which the adjective or noun precedes another noun to represent a unique entity (adjective - noun) or (noun-noun).
- **Actor**: File, person or other system which interacts with the system either by providing or by consuming the information to or from the system.
- **CFG**: The graph representing the control flow between the statements of SRS in the realization process
- **Data Flow Graph (DFG)**: It is represented by a table with entries of SRS statements structured with referenced & defined attributes, directly & implicitly relevant attributes organized in the CFG order
- **Defined Items**: The nouns or noun phrases whose values are modified during the realization of the statement.
- **Directly Relevant Variables**: The variables computed from the slicing criterion variables using the formula,

\[ R^0_c(s) = R^0_c(t) \cup \{ v\mid v \in R^0_c(t), v \notin def(s) \} \cup \{ v\mid v \in ref(s), def(s) \cap R^0_c(t) \neq \emptyset \} \]

Where \( t \) is the previously realized statement and \( s \) is the current statement of refined SRS. Initially \( R^0_0(s) = \emptyset \) if \( s \) is not the last statement else \( R^0_0(s) = V \) where \( V \) is the set of attributes of usecase
- **Directly Relevant Statements**: The statements which contain relevant variables and which is abstracted in the first pass of the formation of the slice using the equation,

\[ S^0_c = \{ s\mid (def(s) \cap R^0_c(t) \neq \emptyset), s \rightarrow_{CFG} t \} \]
- **Object Structure**: The attributes set of the class collected through the cohesive property of entity.
- **Referenced Items**: The noun or noun phrase referred in the statement, if it is present in the statement and after realization of the statement if the value remained unaltered.
- **SRS**: It is a document prepared by the client organization involving the detailed requirements of their information system which includes the overview of the system, the functional & nonfunctional requirements of the system, the actor interfaces, the constraints & the prototypes etc. This document is a part of the project charter which leads to the formal commencement of the software development.
- **Slice**: A slice contains statements that affect a specific set of attributes directly or indirectly by the values computed/ Defined at some point of interest. The slice \( S \) be the subset of original program/ text \( P \), that forms a slice with respect to slicing criterion \(<\text{Statement (n)}, \text{set of variables (V)}\>\) satisfying the following properties:

- \( S \) must be a valid program/ text, Whenever \( P \) halts for a given input, \( S \) must also halt for that input, computing the same values for the variable in \( V \) whenever the statement corresponding to the node \( n \) is executed.
- At least one slice exists for any criterion; the program/ text itself.
- The slice itself realizes some independent behavior.
- **Slicing Criterion**: The point of interest in the slice is Referred to as slicing criterion and is specified by a pair \(<\text{Statement (n)}, \text{set of variables (V)}\>\). Here, \( n \) is the statement number along the CFG and \( V \) is the set of variables on which slicing is required.

**Usecase**: A use case is a complete transaction representing a sequence of activities performed by a system for an actor (outside agent) such as a person or another system at an instance of time. The entire set of use cases encompasses the functionality of a system.

**View Elements**: These are the primitive elements that can be abstracted from the SRS for the design of target information system. The view elements include object class name, object state, object role, attribute, attribute domain, method, method signature, interrelationship, visibility, functional dependency, activity, action state, actor, actor interface attribute, event, use-case, message, guard condition, pre condition, post condition and signal.
2. Methodology

2.1 Classing Technique

Input: SRS
Tools used: control flow table (CFT), data flow table (DFT)

1. Initially we create a blank table. As and when the statement is read, the attribute is written in its lexicographic position. For this, knowledge of the all the attributes is not required at the initial stage. As and when the attribute occurs, we insert the attribute in the lexicographic position as in Table-I. This will help to save unnecessary wastage of memory. This information can be stored in B tree file organization. The procedure for creating the CFG from ref / def table is as follows.
   - Read a ref / def entry from the table.
   - Enter the statement number in the appropriate rows of the attributes at the appropriate referenced or defined column.
   - Amongst the newly entered “referenced” columns, if the defined column statement number is already full, enter the defined column statement number in the start column of the CFG and referenced statement number jump1.
   - If referenced column is already full, replace the entry by new statement number in the referenced column of intermediate table as shown in Table-II. If the defined column is already filled, replace the previous statement with the current statement number and erase the reference number of the entry.
   - Sort the control flow graph entries with the jump1 as primary key and start as the secondary key.
   - In the final CFG, cluster the entries of same statement number in the jump1 with the consecutive statement numbers in the start column.
   - In the final CFG, cluster the entries of same statement number in the start with different statement numbers in the jump1 column.
   - Enter the same statement number of start column entries with different statement numbers in jump1 into final CFG as follows. The start entry contains the cluster start entry and first two consecutive entries of cluster jump1 in the alternative jump columns. If the cluster contains more than two jump entries, then in each entry of the final CFG, with the same start statement number, take two consecutive entries in sequence till all entries of the cluster exhausted.

Table-I. Ref/def (Referenced - Defined) table.

<table>
<thead>
<tr>
<th>Stmt. No</th>
<th>Referenced Attribute</th>
<th>Defined Attribute</th>
<th>verb</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Customers</td>
<td>Bank, branch, branch id, branch location</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Bank, overdrafting facility, account, turnover</td>
<td>Customers, shareholders</td>
<td>Provides</td>
</tr>
<tr>
<td>11</td>
<td>Branch, shareholders, cities of Karnataka</td>
<td>bank, board of governors, SID</td>
<td>Elect</td>
</tr>
</tbody>
</table>

Design a table for flow entries with consecutive statement number in start column of each entry. The entries are organized in the control flow order from start column to end column values as shown in Table-III. Here, each entry originally contains three columns. Since, the number of alternate jumps dynamically varies with the number of referenced attributes which is as per the clients writing style. Therefore, we suggest dynamic list memory for each entry.
Table-II. Intermediate table

<table>
<thead>
<tr>
<th>Stmt no</th>
<th>Referenced item (ref(stmt no))</th>
<th>Defined item (def(stmt no))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table-III. Control Flow Table

<table>
<thead>
<tr>
<th>Stmt no</th>
<th>Referenced item (ref(stmt no))</th>
<th>Defined item (def(stmt no))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Read the RDT entry by entry; write the statement number in the appropriate referenced/ defined attribute entry with the following norms.

- If the defined statement number is written after referenced statement number of the attribute then delete the referenced statement number.
- If a specific entry (row) is complete, then transfer the data to the appropriate defined statement number entry of CFT with referenced entry as an alternative jump. Delete the referenced statement number thus transferred.
- Continue the procedure till the last statement of referenced/ defined table.

2. Design data flow table (DFT) in control flow order as depicted in CFT. Where set of all attributes defined in program line is denoted by \( \text{def}(s) \) and set of all attributes referenced in that program line is denoted by \( \text{ref}(s) \) and is shown in table IV. This can be done by reorganizing the entries of \( \text{ref-def} \) entries in control flow order. The format of DFT is as follows.

Table IV. DFT table

<table>
<thead>
<tr>
<th>Stmt no</th>
<th>Referenced item (ref(stmt no))</th>
<th>Defined item (def(stmt no))</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; Stmt no &gt;</td>
<td>Referenced attribute delimited by comma</td>
<td>Defined attribute delimited by comma</td>
</tr>
</tbody>
</table>

3. Organize the attributes of table-IV in the lexicographic order and identify and indicate with special marks the entries in the DFT which contain only referenced attributes (\( \text{ref}(w) \)) of write statement and entries contain only defined attributes \( \text{def}(r) \) of read statement. Cluster these entries of read and write separately based on file name. Take the set union of all \( \text{def}(r) \) of the read and all \( \text{ref}(w) \) of the write statement. Form another table as shown below

Table V. Read-Write table

<table>
<thead>
<tr>
<th>File name</th>
<th>Referenced item (( \text{ref}(w) ))</th>
<th>Defined item (( \text{def}(r) ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; File name &gt;</td>
<td>Referenced attribute delimited by comma</td>
<td>Defined attribute delimited by comma</td>
</tr>
</tbody>
</table>

4. Design a classing criteria with last statement number of the program, the referenced attributes set \( \text{ref} \ (w) \) and file name as gemination ordered triplet of classing criteria <last stmt no, file name, \( \text{ref} \ (w) \)>.
5. Identify the candidate key for each class using the following steps:
   \[ \text{if}(\text{def}(r) - \text{ref}(w) = \emptyset \text{ then candidate key} := \text{def}(r) \cap \text{ref}(w)) \]
   \[ \text{else def}(\text{program line no}) \in \text{ref}(w) \]
   \[ \text{ref}(\text{program line no}) \in \text{def}(r) \text{ then candidate key} := \text{def}(\text{program line no}) \]

6. Identify the defined attributes in the DFT entries where candidate key of ref(w) is referenced then take the set union of candidate key attributes along with defined attributes then take intersection of set union attributes with referenced attributes of write statement that forms the class attribute set. This can be represented as:
   \[ \{\text{Cs: } = \text{ref}(w) \cap (\text{ref}(s) \cup \text{def}(\text{stmt no})); \text{ref}(\text{stmt no}) \in \text{Candidate key of } (\text{ref}(w))\} \]

7. Repeat step 8 for all candidate keys in write statement
   \[ \{\text{Cs: } = \text{Cs} \cup \{\text{ref}(w) \cap (\text{ref}(\text{stmt no}) \cup \text{def}(\text{stmt no})\}); \text{ref}(\text{stmt no}) \in \text{Candidate key of } (\text{ref}(w))\} \]

8. \[ \text{Ci :=Cs} \]

9. Repeat this procedure until ref(w) = \emptyset

10. From Ci study each attribute for the assignability, if the size and type assigned then those set of attributes form the class attributes and if the size and type cannot be assigned then those attribute form the class name.

   We know the functional dependency using identification candidate key. Here we have implicitly implemented the good database design principles partially in the class structure of a program.

2.2 Slicing Technique

Input: all class names and their attributes (all Ci s')
Tool used: structure of the enhanced DFT

1. Extend the original DFT to incorporate the following additional columns with initial values blank in all the entries.

<table>
<thead>
<tr>
<th>Stmt no</th>
<th>Referred attributes</th>
<th>Defined attributes</th>
<th>Directly relevant attributes ( R_c^0 )</th>
<th>Directly relevant statements ( S_c^0 )</th>
<th>Indirectly relevant attributes ( R_c^1 )</th>
<th>Indirectly relevant statements ( S_c^1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>stmt no</td>
<td>ref</td>
<td>def</td>
<td>( R_c^0 )</td>
<td>( S_c^0 )</td>
<td>( R_c^1 )</td>
<td>( S_c^1 )</td>
</tr>
</tbody>
</table>

2. In the fourth column, let the last entry contains \( R_c^0 := \{V_i\} \) where \( V_i \in \text{set of class attributes contained in } C_i \) and all other entries contains null value.

3. For each class carry out the following procedure
   - Each time take the attributes mentioned in the \( C_i \) and go backward direction from DFT where slicing criterion defined as the <last statement, \( C_i >
     \[ R_c^0(s) = R_c^0(s) \cup \{V_i \mid V_i \in R_c^0(t), V_i \notin \text{def}(s)\} \cup \{V_i \mid V_i \in \text{ref}(s), \text{def}(s) \cap R_c^0(t) \neq \emptyset\}
   - Update \( R_c^0(s) \) for all the entries using above equation with initial \( R_c^0(s) \) containing null value for all statements.
   - Step 3 is repeated for all the classes

4. Using this equation, the algorithm determines the directly relevant statements as:
   \[ S_c^0 = \{s \mid \text{def}(s) \cap R_c^0(t) \neq \emptyset, s \rightarrow_{CFG} t\} \]
   Let the range of influence infl(b) of a branch statement b is defined as the set of statements control dependent on b. The set of control dependent branch statements of control predicate b is defined as,
   \[ B_c^0 = \{b \mid s \in S_c^0(s), s \in \text{infl}(b)\} \]
   The definition of the branch statements \( B_c^0 \) that are indirectly relevant due to the \textit{influence} they have on nodes \( s \) in \( S_c^k \) at level k is given by,
   \[ \forall k \geq 0, B_c^k = \{b \mid s \in S_c^k, s \in \text{infl}(b)\} \]
   Next for each level k, the set of indirectly relevant variables \( R_c^{k+1} \) are determined using
   \[ R_c^{k+1}(s) = R_c^k(s) \cup \bigcup_{b \in B_c^k} R_c^{0}(b, ref(b))(s) \]
This computes, in addition to $R^k_c$, the transitive data dependence on the statements $B^k_c$; this is performed using first type of iteration (tracing transitive data dependences) again with respect to slicing criterion $<b, ref(b)>$ where $b \in B^k_c$ is a branch statement.

The indirectly relevant statements in $k+1^{th}$ iteration consist of vertices in $B^k_c$ together with vertices that define the variables $R^{k+1}_c$ relevant to a CFG successor $t$. The indirectly relevant statements in $k+1^{th}$ iteration computed as

$$S^{k+1}_c = B^k_c \cup \{s | \text{def}(s) \cap R^{k+1}_c(t) \neq \emptyset, s \rightarrow \text{CFG}\}$$

The sets $R^{k+1}_c$ and $S^{k+1}_c$ are non-decreasing subsets of the program variables and statements respectively.

5. Identify contiguous set of statements in the slicing table containing the defined attributes of a class. If the defined attributes of the class are in a cluster consecutive statements of the DFT, then that cluster of statements of the programming system form an object method.

2.3 Identification of Usecases

Take slice for each class. Then identify the actor using the following steps.

- Consider the referenced attributes of a class (one class at a time) say $C_i$
- Consider the referenced attributes of other remaining classes in the current slice in ref/def say $C_j$ where $i \neq j$ and $j=1$ to $n$
- Use the following formula to identify the referenced actor $\text{ref} - C_j \cap \text{ref}$ where $j=1$ to $n$

Repeat the above steps for Defined Attributes column to identify the defined actor.

Consider Ref actor in one class say $C_i$ and find association of this class with other classes. This can be achieved by identifying the interrelationship of the other object methods with $C_i$. We can proceed this way in the chained manner until we find a class in which the actor is defined. This sequence of object methods forms the use case for that actor.

Conclusion

Identifying the usecase structure from SRS is a herculean task. As the SRS is format free and human skill dependent the correctness, completeness and robustness of the SRS suffer. However with the sound logic we have devised automated methodology to design the CFG in the form of CFT. This methodology is based on logical positivism the correctness, completeness and robustness improves. Since the read, write and their synonyms implicitly indicate functional dependencies amongst themselves, we have proposed a classing technique to identify the attributes of the class. This implicitly uses good database design principles. The object methods are identified from CFG based on consecutive entries of the DFT for a cluster of statements.

The slice of SRS based on class attributes identifies the object methods as consecutive statements. These identified object methods are referenced based on the parameters present in the signatures of the object methods. Thus, we have developed a suigeneris methodology to identify the usecase.

Acknowledgements

Words are insufficient to express our deep sense of gratitude to Dr. D. B. Phatak, Professor of Dept. of Computer Science & Engineering IIT Bombay, for his inspiration.

References

An Ameliorated Methodology to Establish the Analogy between Business Process Perspective Views and UML Diagrams” has been published in IEEE sponsored International Conference on Advances in Computing, Communications and Informatics(ICACCI-2014), during September 22-29, Delhi.

Program Slicing by Mark Weiser in IEEE international conference in software maintenance, 1981.


