

# Automation of Version Management and Change Propagation

V. Koti Reddy  
Department of CSE,  
JNTUA College of Engineering,  
Anantapur, India;

Prof. A. Ananda Rao  
Department of CSE  
JNTUA College of Engineering,  
Anantapur, India;

---

## ABSTRACT

Software systems generally involve a number of phases and tend to evolve over a period of time. Several revisions of individual artifacts which make up the system take place during the evolution process. The revisions and refinements are captured and maintained as different versions using configuration/version management tools. A key issue in the version management of object oriented software system is classification of attributes of an artifact into two categories namely versioning and non-versioning which determines the major and minor functionalities, respectively, of the artifact. In this paper we propose an algorithm for automating the process of above classification. The results of classification are used to predict the type of change as version change or equivalent change required to be made in the related artifacts at the time of evolution due to change propagation. A semantic entity called Unified Representation of an Artifact (URA) is used for representing the artifacts in the software system. The object oriented issues like inheritance, aggregation and association, are also considered for propagating a change in the software system. The role of accessibility of attributes such as private, public and protected in version management is also considered.

**Keywords:** Change Propagation, Equivalent Change, Unified Representation of an Artifact, Version Change, and Version Management.

---

## 1 Introduction

Software systems are developed generally based on an iterative paradigm, where each iteration provides a successive refinement over previous iteration. Refinements in software systems are managed by maintaining different configurations of various artifacts of the systems. User requirements of software systems keep changing. This change leads to

evolution of software system. As the requirements of users changes, software has to support the evolution easily. The changes in an artifact normally require corresponding changes in other dependent artifacts. Therefore there is a need to capture the evolution of related artifacts to keep the system in consistent manner. Capturing the evolution of software system is major issue in software maintenance phase. The concept

of version management is used for managing the evolution of the software system.

A key issue in the version management [1][2] of object oriented software system is classification of attributes of artifacts into two categories namely versioning and non-versioning attributes. Here an attribute is used to mean an instance variable or method of a class. If a change of an artifact leads to change of other related artifacts, then it is versioning attribute, otherwise it is a non-versioning attribute. Versioning attributes determine major functionality of software system and non-versioning attributes determine minor functionality. Version of an artifact is represented in the form: “<major><minor>”. If there is a major change in the functionality of an artifact then it is said to be version changed. This is caused when there is a change in one or more of its versioning attributes. On the other hand if there is a minor change in the functionality of the artifact then it is said to be equivalent change. This is caused when there is a change in one or more of its non versioning attributes.

### 1.1 Introduction to URA

The Unified Representation of an Artifacts (URA) [2] is a meta model entity that represents an artifact of any type or granularity. An artifact is nothing but any logical entity of interest. Artifacts map to physical entities in different ways like classes, sets of classes, sub systems, documents, etc. Fig. 1.1 shows the structure of URA. An URA mainly comprises of three components. The first one extracts the

artifact from the information system. The second component contains the information about the artifact. The third component enforces authentication mechanisms. A set of features are associated with the URA, which allows it to be classified and queried. These features can be either attributes or functionalities of artifacts. The semantic based version information set keeps track of evolution of artifacts. In addition to these, there are labeled links pointing to other URAs, which reflect the relation between the artifacts that the URAs present. A software project is represented as directed graph of URAs. The graph will evolve as changes occur in the project. An artifact in the project is represented as an URA, which is a node in the URA graph. Directed edges in the graph are labeled. The labels are the relationships between the artifacts. Changes occurring in a node are classified into two types. Changes, which create a new version and changes, which create new equivalent. In the URA graph a node is said to change in to new version, if the change affects the semantics of the node. The semantics of a node is said to have changed if there is a change in the functionality or interface of the node. If the change does not affect the node semantics, the node is said to have changed in to new equivalent. The attributes are categorized into versioning attributes and non versioning attributes. Here the attribute is used to mean a feature of an artifact. The labeled links indicate the dependencies between the nodes of the graph and need to propagate the changes. A pivot node in the graph represents the whole project. URA nodes are linked to pivot node by dependency links. Changes are

propagated to this node also. The version of this node is nothing but the version of the Software system.

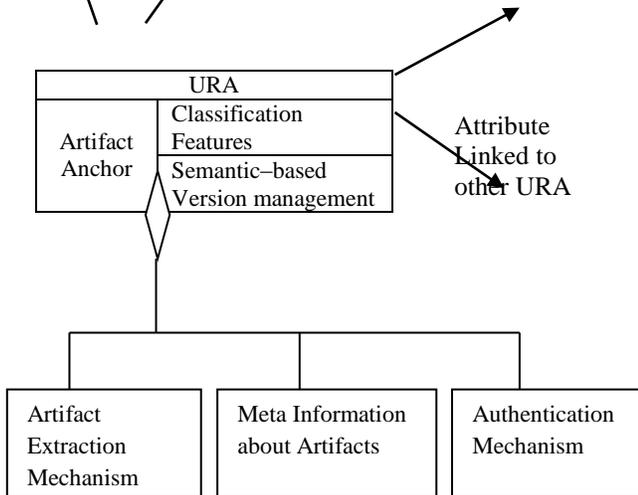


Fig. 1.1 Structure of an URA

## 2 Change Propagation Mechanism

In configuration management, generally whenever a change occurs in an artifact that has to be propagated to the all related artifacts. This propagation preserves the consistency of software system under consideration.

### 2.1 Varieties of change propagation

In object oriented technology classes are considered as basic building blocks of software system. These classes are related using various types of relationships among them such as inheritance, aggregation and association. The classes are represented as artifact in URA graph and relationship among the classes are represented as links. These links are labeled

as cohesive or non-cohesive in the URA graph. In URA graph change is propagated to the related artifacts based on two values which are called as focus of change and property of a link between the artifacts i.e. cohesive or non-cohesive.

The propagations in URA graph are categorized into two categories, one is propagation of equivalent change and the other is propagation of version change. These two categories are tabulated in Tables 2.1 and 2.2 respectively. Whenever a change is propagated in the URA graph. The recommended changes are shown in the following tables.

Table 2.1: Version Propagation table

	Version Focus	
	LOW	HIGH
Cohesive link	V-Change	E-change
Non-Cohesive link	E-change	E-change/ N-change

Change propagation in case of version change of an artifact is as follows.

- If the link is cohesive and version focus is LOW then a version change (V-Change) is recommended to related URAs.
- If the link is cohesive and version focus is HIGH then an equivalent change (E-Change) is recommended to related URAs.
- If the link is Non-cohesive and version focus is LOW then an equivalent change (E-Change) is recommended to related URAs.
- If the link is Non-cohesive and version focus is HIGH then an equivalent change (E-Change) is recommended to related

URAs. If the version focus is too HIGH and cohesion of link is too low then no change is recommended.

Table 2.2: version propagation table

	Equivalent Focus	
	LOW	HIGH
Cohesive link	E-Change	E-Change
Non-Cohesive link	E-Change	N-Change

Change propagation in case of equivalent change of an artifact is as follows.

- If the link is cohesive and focus is LOW, then an equivalent change is recommended to related URAs.
- If the link is cohesive and focus is HIGH, then an equivalent change is recommended to related URAs.
- If the link is non-cohesive and focus is LOW then an equivalent change is recommended to related URAs.
- If the link is non-cohesive and the focus is high then no change (N-change) is recommended to related URAs.

## 2.2 Reasons for Change Propagation

Change propagation can occur because of two reasons:

- If an attribute of an artifact is changed, then change is propagated to related artifacts. Various cases of this reason have been depicted in the table 2.3
- New dependency links will be created when a new artifact is added to the system. These link directions can be to or from the new artifacts. The

[www.ijsea.com](http://www.ijsea.com)

recommended changes of an artifact based on the direction as well as cohesiveness of the link are shown in table 2.3.

Table 2.3 New artifacts cause changes to existing artifacts.

		Direction of Link		
		To the new Artifact	From the new Artifact	Bi-Directional
Property of the link	Cohesive	E-change	V-Change	V-Change
	Non-Cohesive	N-Change	E-Change	E-Change

An artifact moves into a transient state when ever there is a change in an artifact. The artifact is in normal state before the change. Changes in an artifact will lead to chain of change propagatoin. It may also form a cycle. This leads to infinite change recomentdations. This situation is avoided by marking the states of artifacts that is already changed as transient state. In this way the states of an artifact are used in version management. The change management of an artifact has various sets of states. Only three states are considered for the sake of simplicity. These are transient, normal and replace states. There is no need to propagate the changes when ever a defective version of an artifact is replaced. Change propagation can be avoided by marking state of the replacing artifact as replace state.

## 3. Change Management

A class is considered as a basic entity in object oriented systems. Hence each class is treated as an artifact and

denoted as a URA. Links between the URAs shows the relations between the classes such as inheritance, aggregation and association. It is easy to manage the versions through a URA graph. There are two issues of change management. These are version change and version propagation. They are addressed below. When ever a version change occur to an artifact there is a need to propagate the change to other dependent artifacts.

### 3.1 Version change:

Version change of software systems are of two types. One is change in version and the other one is equivalent change. If the changes in software are significant and affect the software system functionality then it is a version change. Otherwise if the changes in software are due to minor improvements and system functionality is not affect much, then it is said to be an equivalent change. Changes can also be categorized as follows. One is internal change of artifacts and other is change propagated from related artifacts. Internal change of an artifact can occur through version or non-versioning attributes. The type of change of an artifact is decided by versioning attributes or non-versioning attributes. Change can occur in two ways. One is change in attributes and the other is addition of new attributes to the artifact. If the attribute is versioning attribute then the type of the change occurring in the class is called as version change (V-change). If the attribute is non-versioning attribute then the type of the change occurring in the class is called as equivalent change (E-change).

### 3.2 Version Propagation

In every software system the changes of the artifact will cause changes of other related artifacts. Thus change propagation mechanism is a major issue in version management. Version change of an artifact will occur if the related artifacts having accessibility to the artifact attributes and functionality. In a class there are three types of access specifiers for an attribute namely public, private and protected. The main aspects of version propagation are focus and cdegree (degree of cohesion). The focus is with respect to change in URA. Each URA in URA graph represents an artifact of the software system. A change in URA has a value called focus [1]. The focus of change is a probability that the change does not impose similar change in other related URAs. Related URA means there exist some dependency links among corresponding

Attribute \ FOCUS	FOCUS
Private	HIGH
Public	LOW
Protected	If (link = Inheritance) LOW Else HIGH

artifacts of URAs. The change

pertaining to an attribute depending on the accessibility is tabulated in table 3. 1

Table 3.1: Focus evaluation table

If an attribute of an artifact is private then change of that attribute may not impose changes in related artifacts. Thus the focus of private attribute is HIGH. Similarly public attribute focus is LOW. If the attribute is protected, the change may affect the related artifacts depending on link between URAs. If the link is inheritance link then focus is LOW, otherwise focus is HIGH. Cdegree of a link is the indicator of the amount of dependency that exists between the two related artifacts [1]. The value of the cdegree has a range [0,1]. The link is said to be “strong”, if the cdegree value is more than the threshold (say 0.5) and this link is called cohesive link. The link is said to be “weak” if the cdegree value is less than threshold value and link is called non-cohesive link.

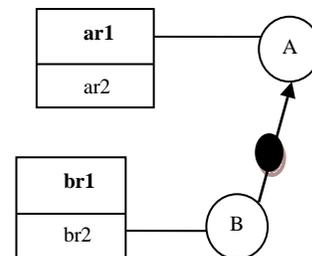
### 3.3 Evaluation of URA graph

The fig. 3.1(a) shows the two categories of versioning and non-versioning attributes of a node. Fig. 3.1(b) shows the changes in the versioning attributes leads to new versions. Fig. 3.1(c) shows the changes in non-versioning attributes create new equivalents. Changes in the cardinality of the sets of versioning attributes create new version. This is shown in fig. 3.1(d). Fig. 3.1(e) shows the changes in the cardinality of set of the non-versioning attributes create new equivalent. Fig. 3.1(f) illustrates the changes in graph semantics due to addition and deletion of links create new versions of the nodes affected.

Change is propagated to other nodes depending on the type of the change, focus of the change and the cdegree of the

links to other nodes. A summary of change propagation is as follows-

- Incase of version change, if the cdegree of the link connecting two nodes is greater than or equal to threshold value, then it is communicated as version change or version change recommendation from a node to its neighboring node. This is illustrated in fig. 3.1(g)
- Incase of version change, if the cdegree of the link connecting two nodes is lesser than threshold value, then it is communicated as equivalent change or equivalent change recommendation from a node to its neighboring node. This is illustrated in fig. 3.1(h)
- Incase of equivalent change, if the cdegree of the link connecting two nodes is greater than or equal to threshold value then it is communicated as equivalent change or equivalent change recommendation from a node to its neighboring node. This is illustrated in fig. 3.1(i).
- Incase of equivalent change, if the cdegree of the link connecting two nodes is lesser than threshold value, then it is considered as equivalent change or equivalent change recommendation from a node to its neighboring node. This is illustrated in fig. 3.1(j).



(e)

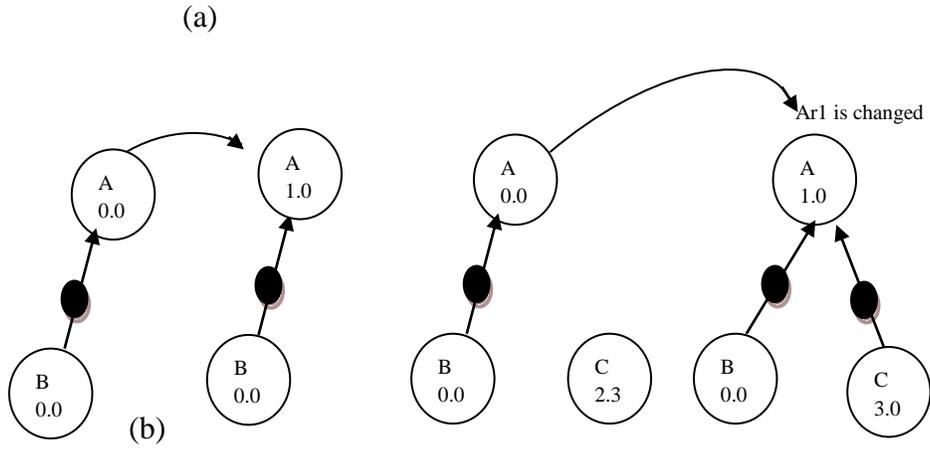
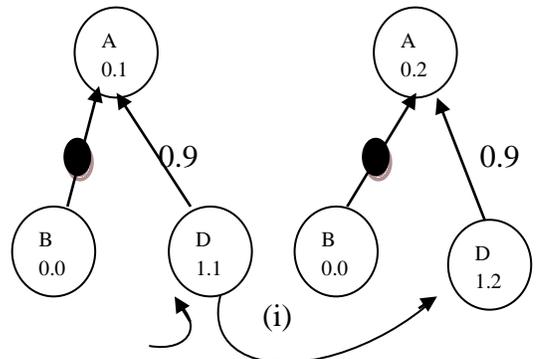
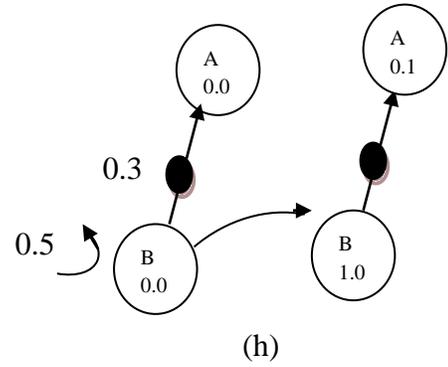
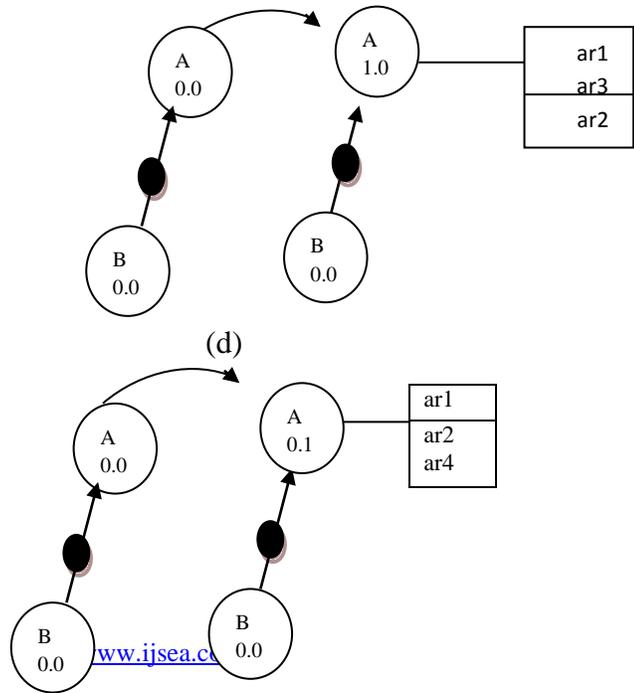
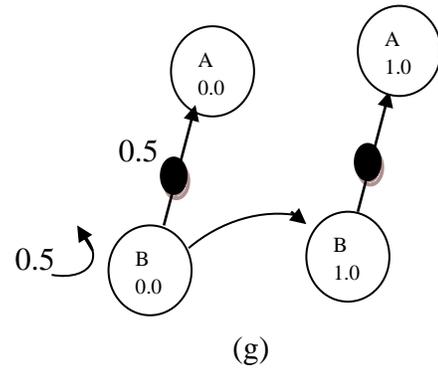
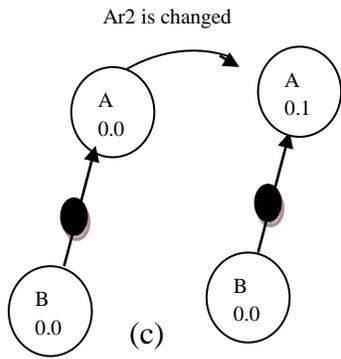
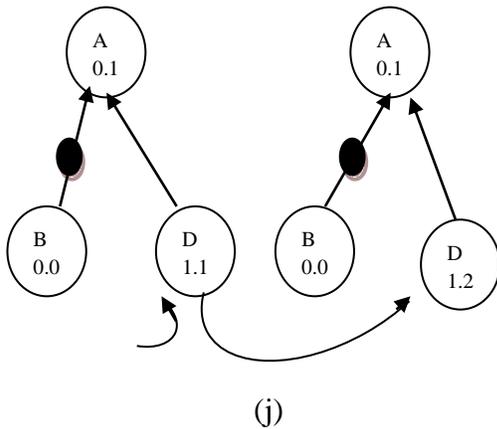


Fig. 3.1 Evaluation of URA Graph  
(f)



representation of UML class diagram with inheritance structure as a URA graph is shown in 0.7 fig 3.2.



### 3.4 Change Propagation (in case of Inheritance, Aggregation and Association)

(a) URA Graph Representation of Inheritance

three different links between the classes in OO systems are inheritance, aggregation and association. These are mapped to URA labeled links i.e., cohesive and non-cohesive links. These mappings as well as change propagation are discussed in this subsection.

**3.4.1 Inheritance:** The unidirectional dependency link between the base class and derived class is called inheritance link. The changes made to base class affect the derived class. Change in a private attribute leads to change with HIGH focus. Change in a public or protected attribute leads to change with LOW focus. The changing attribute can be either versioning attribute or non-versioning attribute. Correspondingly the focus of the change will become version focus or equivalent focus. The

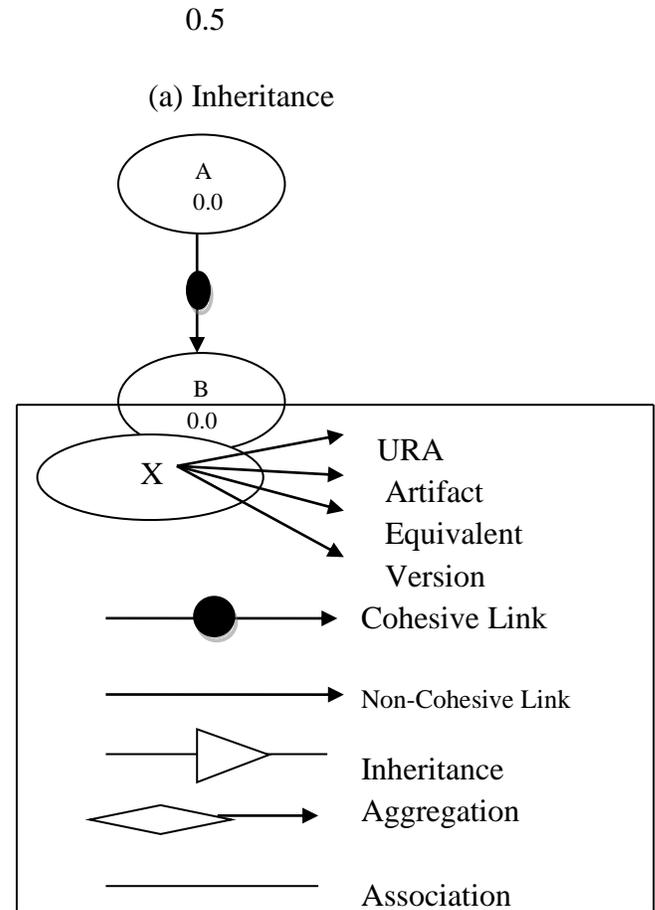
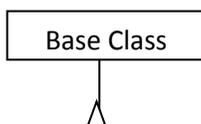


Fig. 3.2 Inheritance and URA Graph.

**3.4.2 Aggregation:** It is a unidirectional dependency link. Fig.3.3 shows the representation of UML class diagram with aggregation structure and its corresponding URA graph. It is a cohesive link because change made to part classes affects the whole class. Change in a private or protected



attribute leads to a change with HIGH focus. Change in a public attribute leads to a change with low focus.

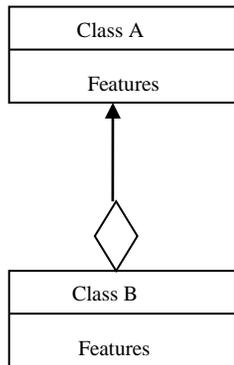
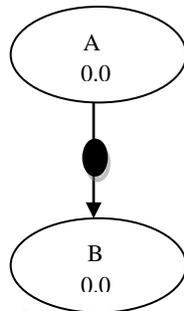


Fig. 3.3 (a) Aggregation



**3.4.3 Association:** It is a bidirectional dependency link in UML class diagram. It can be cohesive or non-cohesive link. The property of link can be found by using cdegree value. Fig 3.4 shows UML class diagram with association structure and its corresponding URA graph.

Fig. 4.1 URA Graph Generator. can be cohesive or non-cohesive link. The property of link can be found by using cdegree value. Fig 3.4 shows UML class diagram with association structure and its corresponding URA graph.

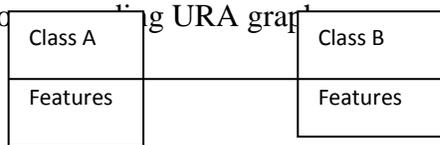


Fig. 3.4(a) Association

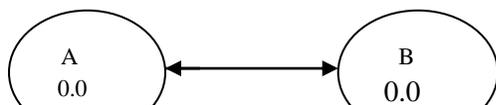


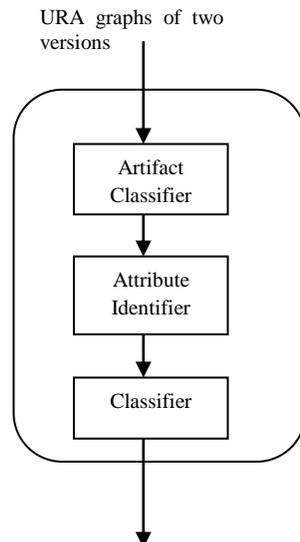
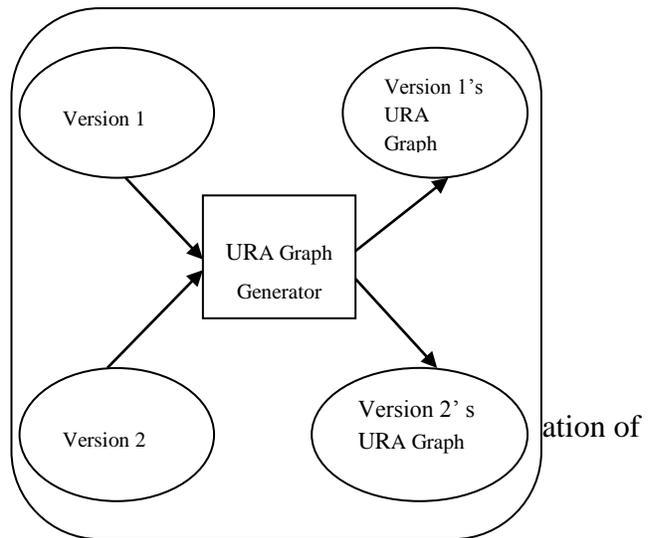
Fig. 3.4(b) URA Graph Association of Association

## 4. System Design

The following section explains URA graph generator, attribute classifier and change propagator.

### 4.1 URA Graph Generator

The Fig. 4.1 shows URA graph generator which takes two versions of software systems as input and generates their corresponding URA graphs.



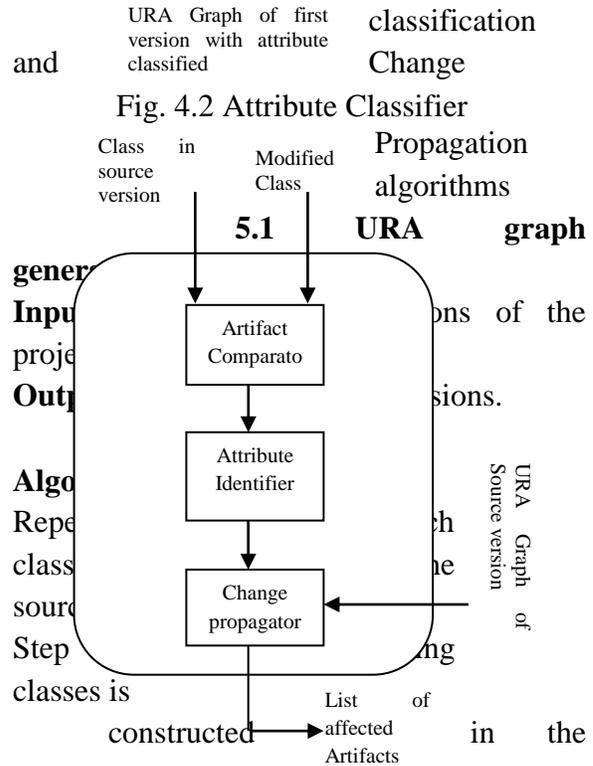


Fig. 4.3 Change Propagator

#### 4.2 Attribute Classifier

The input to attribute classifier is URA graphs of the two versions, which are generated by the URA graph generator. It gives the URA graph of the first version with its attributes classified as versioning, non-versioning or unknown. Fig. 3.2 shows the attribute classifier

#### 4.3 Change Propagator

The input to change propagator are class (artifact) in the source version and the modified class. It generates the list of affected artifacts by using the URA Graph of the source version. Fig 4.3 shows change propagator

#### 5 Algorithms

The following sections explains the URA Graph generation, Attribute

first parse of the source. The attributes and methods are extracted for each class. This forms the URA node of this class. Step 2: The links between the classes are determined in the second parse of the source.

#### 5.2 Attribute classification

**Input:** URA graphs of two versions, generated by the URA graph generator.

**Output:** URA graph of first version, with the attributes of the artifacts classified as versioning, non-versioning and unknown.

**Algorithm:**

Step 1: Consider a particular class from the two versions of system.

Step 2: Determine whether the change is version change or an equivalent change.

Step 3: Determine the attributes, which caused the above change.

Step 4: Accordingly classify the attributes as versioning and non-versioning attributes.

### 5.3 Change Propagation:

**Input:** A class in the source version and its modified form and URA graph of the source version.

**Output:** List of effected classes (Version changed classes and equivalent changed classes).

#### Algorithm:

Step 1: compare the input class of the source version and its modified form.

Step 2: Identify the attributes, which were changed.

Step 3: Identify whether the change is a version change or an equivalent change based on classification status of the variables that were changed.

Step 4: Accordingly mark the class as version changed or equivalent changed class.

Step 5: Propagate the direct and indirect changes using the URA graph and mark the effected classes accordingly.

testing purpose. There is a version change between RFV 1.1 and RFV 2.1. There is an equivalent change between RFV 2.0 and RFV 2.1.

## 6.1 Results

The results are illustrated by the following screen shots of the output. The fig. 6.1 shows the main screen which contains the automated attribute classifier and Change propagator.

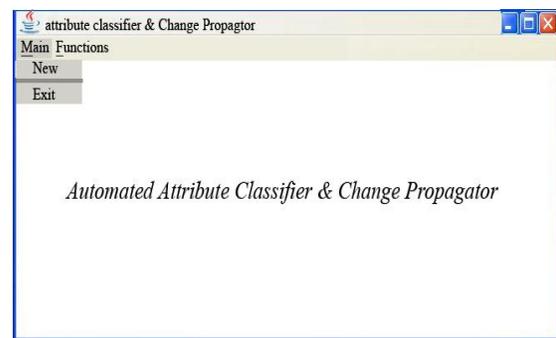


Fig. 6.1: Main Screen

Fig. 6.2 shows the open dialog to select source directory of the version to construct the pivot graph.

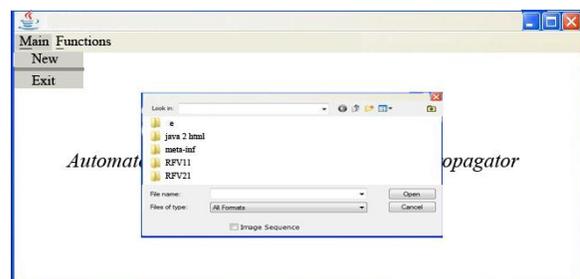


Fig. 6.2: Open Dialog

## 6 Results and conclusions

The tool was tested using professional software by name “Restricted Focus Viewer”. Restricted Focus Viewer (RFV) 1.1, RFV 2.0 and RFV 2.1 are the three different versions considered for the

The attributes of RFV\_FOCUS\_Window class are classified and the classification results are shown in fig. 6.3

Name	Type	ClassificationStatus	AccessModifier	TypeofChange
r3_ver_offset	int	VERSIONING	PUBLIC	EXISTENCE
r1_height	int	VERSIONING	PUBLIC	EXISTENCE
r2_ver_offset	int	VERSIONING	PUBLIC	EXISTENCE
r3_height	int	VERSIONING	PUBLIC	EXISTENCE
r1_hor_offset	int	VERSIONING	PUBLIC	EXISTENCE
r1_ver_offset	int	VERSIONING	PUBLIC	EXISTENCE
r2_height	int	VERSIONING	PUBLIC	EXISTENCE
r1_height	int	VERSIONING	PUBLIC	EXISTENCE
r1_width	int	VERSIONING	PUBLIC	EXISTENCE
r2_hor_offset	int	VERSIONING	PUBLIC	EXISTENCE
r3_width	int	VERSIONING	PUBLIC	EXISTENCE
r1_ver_offset	int	VERSIONING	PUBLIC	EXISTENCE
r3_hor_offset	int	VERSIONING	PUBLIC	EXISTENCE
r1_hor_offset	int	VERSIONING	PUBLIC	EXISTENCE
r1_width	int	VERSIONING	PUBLIC	EXISTENCE
r2_width	int	VERSIONING	PUBLIC	EXISTENCE

Name	Type	ClassificationStatus	AccessModifier	TypeofChange
calculateoffset	void	UNKNOWN	PRIVATE	
validateDimensions	void	UNKNOWN	PRIVATE	
newfocuswindow	void	VERSIONING	PUBLIC	EXISTENCE-ACCESS MODIFIER-
offsetRegions	void	VERSIONING	PUBLIC	

Fig. 6.3 Classification Results

Figure 5.4 Shows the results of propagation function on RFV\_Focus\_Window classes.

E-Changed Classes

- RFV11.RFV
- RFV11.RFV\_Time\_Limit\_Threadd
- RFV11.RFV\_Motion\_Blur\_Threadd

V-Changed Classes

- RFV11.Replayer
- RFV11.RFV\_Replayer\_Threadd

Fig. 6.4 Pragation Results

## 6.2 Conclusions

This process of automation provides a method for finding the versioning and non-versioning attributes of an artifact. This project has been used to collect and analyze the data for a number of applications. Improved measures for calculation of cdegree (cohesion degree) may be adopted. The effect of changes made in an artifact can be determined to a higher degree of precision. This may be achieved by slight improvement in the strategy used for keeping track of the links of an artifact.

## 7. References

- [1] D. Janaki Ram, M. Sreekanth, A. Ananda Rao, "Version Management in Unified Modeling Language", Technical Report IITM-CSE-DOS, IIT Madras, India.
- [2] D. Janaki Ram, S. Sreenath, R. Rama Krishna, "A Generic Model for Semantics- Based Versioning in Projects ", IEEE Transactions on Systems, Man and Cybernetics, vol. 30, No. 2, March 2000.
- [3] S. Srinath, k. Venkatesh, D. Janaki Ram, "An Integratd Solution Based Approach to Software Development using Unified Reuse Artifacts", ACM Software Engineering Notes. July 1997.
- [4] Lucki, "A Graph Model for Software Evolution ", IEEE Transactions on Software Engineering, Vol. 60, No. 8, Aug 1990.
- [5] Chia-Song Ma, Carl K. Chand and Jane Cleland-Huand, "Measuring the Intensity of Object Coupling in C++ Programs" IEEE 2001
- [6] "Versioning in Apache", [Http://www.apache.org/versioning.html](http://www.apache.org/versioning.html).
- [7] Beech D. and B. Mahbod, "Generalized Version Control in an Object Oriented Database", ICDE, PP. 14-22, 1988.
- [8] Zeller A., A Unified Version Model for Configuration Management, SIGSOFT'95: proceedings of 3<sup>rd</sup> ACM SIGSOFT symposium on foundations of software engineering, New yark, NY, USA .
- [9] Babich W. A., software configuration management. Addotion –Wesley, Reading, Massachusetts, 1986.

- [10]Conradi R. and B. Westfechtel, Version Models for Software Configuration Management, ACM Compt. Surv., Vol. 30 , no. 2, pp. 232-282, 1998.
- [11]Beech D. and B. Mahbob, Generalized Version Control in an Object Oriented Database. , ICDE, pp. 14-22, 1988.
- [12]Ahmed R. and S. B. Navathe, Version Menagement of Composite Objects in CAD Database, SIGMOD'91, : proceedings of the 1991 ACM SIGMOD international conference on management of data, (New York, NY, USA).
- [13]Janasen A. R., “ Restructed Focus Viewer Website. ”<http://www.monash.edu.au/tonoj/RFV>
- [14]Clarkson, P. J., Simons, C. and Eckert, C.M, Change Propagation in the Design of Complex Products of the Engineering Design Conference, Brunel University, Uxbridge, UK. 2000.
- [15] Munch, B.P. Conradi, R. “A layered architecture for uniform version management”, IEEE Transactions on Software Engineering, Dec 2001.
- [16] Sebstein Ulewicz et at.,“Software changes in Factory Automation”, IEEE Transactions on Software Engineering, IEEE 2014.
- [17] Chenguang Zhu, “Semantic Slicing of Software Version Histories” IEEE Transactions on Software Engineering, February 2017