International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:8, No:8, 2014

Heterogeneous Artifacts Construction for Software Evolution Control

Mounir Zekkaoui, Abdelhadi Fennan

***Abstract*—**The software evolution control requires a deep understanding of the changes and their impact on different system heterogeneous artifacts. And an understanding of descriptive knowledge of the developed software artifacts is a prerequisite condition for the success of the evolutionary process.

The implementation of an evolutionary process is to make changes more or less important to many heterogeneous software artifacts such as source code, analysis and design models, unit testing, XML deployment descriptors, user guides, and others. These changes can be a source of degradation in functional, qualitative or behavioral terms of modified software. Hence the need for a unified approach for extraction and representation of different heterogeneous artifacts in order to ensure a unified and detailed description of heterogeneous software artifacts, exploitable by several software tools and allowing to responsible for the evolution of carry out the reasoning change concerned.

***Keywords*—**Heterogeneous software artifacts, Software evolution control, Unified approach, Meta Model, Software Architecture.

I. Introduction

IN computerized applications related to all activities areas, software evolution is the major process that ensures innovation and continuous improvement methods, techniques and tools of information processing. This explains why the dedicated cost is the most important of all IT activities. Be successful the software evolution by optimizing the effort and high cost is became a primary goal of software engineering. The optimization and the success of the evolutionary process require an understanding and a descriptive unification of the developed software artifacts, allowing better control of the underlying activities of the evolution. This knowledge includes in first the individual description of each artifact and its interdependence against the rest of the software. The architectural description adopted of software artifacts and the partial views graphs of relationships (inheritance, communications, call, import, etc...) Between artifacts are parts of the body of knowledge to be made available during the various situations of reasoning which can stake the process of evolution.

We consider an application is described by a set of heterogeneous and distributed software artifacts. The artifacts can refer each other explicitly (e.g. a hyperlink in an HTML document) or implicit manner (e.g.: a Java class is referenced by name in an XML deployment descriptor). All artifacts can evolve over time (artifacts can be removed others can be added). Each artifact may change over time. These changes can be a source of degradation in functional, qualitative, or behavioral terms of modified software. Hence the need for a meta-model of unified representation of heterogeneous artifacts and an expression formalism of all artifacts. The formalism of description shall be uniform and will allow the description of heterogeneous artifacts allowing to the responsible for development to use a unified description of the artifacts to lead better software control.

It is important to emphasize that the current traditional approaches to understanding the evolution of the software include the use of the history of software systems to analyze their current situation and predict future changes complemented by refactoring approaches, or existing reverse engineering [1]-[4]. These approaches extract the data either by browsing all artifacts with a listening on developer construction operations then stored in a database [5], or through the conversion of all artifacts into XML before applying consistency rules [6]. This requires the handling and storage of a large data set of software description. Approaches traditionally adopted for these empirical studies require costly data collection (often manual) and complex statistical analysis. In the case of our approach, allowing us to users (who are often responsible for evolution) to extract the levels of artifacts that need and stored in the database to lead better control of the software (example: Classes, methods, attributes, beans deployment descriptor).

It is also important to note that these approaches generally allow that to work on homogeneous artifacts (documents specific objects e.g.: UML models) [7]-[9], or using pivots formats (such as XML) to hide the heterogeneity [6], [10]. In addition, they generally cannot cope with the evolution of different heterogeneous artifacts.

Under the control of software evolution, this paper proposes an extraction and representation of the different software artifacts in order to ensure a unified and detailed description of heterogeneous software artifacts, exploitable by several IT tools and allowing responsible for the evolution to complete the adopted reasoning to the concerned change.

The approach is validated by performing a specific platform for the collection of knowledge and a semi-automatic mechanism to facilitate the extraction of all types of heterogeneous artifacts. Validation is performed mainly in the source code to validate the extraction of software artifacts and their dependencies in distributed applications. The rest of the paper is organized as follows:

Section II presents the foundations of our approach,

M. Zekkaoui is with the Faculty of Science and Technology, Tangier, Morocco (phone: +212676013932; e-mail: zekkaoui@gmail.com).

A. Fennan is with the Faculty of Science and Technology, Tangier, Morocco (e-mail: fennan@fstt.ac.ma).

showing how we attacked the problem, the methods and tools we used and presents the results of our approach, the meta-model construction, unified formalism of software artifacts extraction. Section III presents the web platform that we have developed to validate our approach and we conclude in the last section.

II. Unified Meta-Model of Artifacts

A. The Artifacts Classification

Several approaches of software artifacts building have been implemented. They generally use systems queries on graphs and graph rewriting systems. For decades compilers have used tree structures to represent the programs. However, trees are limited in the representation of different software artifacts. The concept of graph, more particularly that of typed and attributed graphs [11], is a generic way, semi-automatic and most suitable than trees to represent the structure of artifacts as well as additional information about themselves (through attributes).

Classification or hierarchy of artifacts proceeds a strategy of simplification the artifacts extraction process. We retain two main classification criteria allowing us to define two levels (or dimensions). These are the abstraction and granularity levels [12]. It is important to note that artifacts can have the same abstraction level without the same granularity level. Within the paradigm object, for example, an instance variable (field of a class) and a method have the same abstraction level, but not the same granularity level. Similarly, artifacts can have the same granularity level but different abstraction level. Thus, a class has the same granularity level as a template or class template but not the same level of abstraction (a template to a level higher than that of a class abstraction).

For a better understanding of software applications, we considered that all artifacts, regardless of their type, from different file types and are similar to typed graphs [11]. An Artifact is then composed of elements (artifacts). Each element is typed. It can carry values for attributes and can reference other elements. All attributes and references are also typed. The artifacts are interconnected to different abstro-granular levels [12]. All artifacts can be represented by the hierarchy levels in the triple $<\Sigma_{Ft}, \Sigma_{Lv}, \Sigma_{art}>$ where Σ_{Ft} is the set of all file types, Σ_{Lv} is the set of all levels. The k^{th} artifact of j^{th} level of the i^{th} type of file is represented by the triple $<Ft_i, Lv_j, art_k>$. For example, $<Ft_{Java}, Lv_{Class}$, Calculator> denotes that Calculator is an artifact belonging to the class level in a file Java type.

Modeling the inter-relationships artifacts is a complex and very important task. We consider three types of relationships (Fig. 1) [13]. They are:

- **Inter-files relationships**: These relationships connect artifacts belonging to two different file systems. This is the case for example of the relationship between a UML class and a Java class that implements or the relationship between java class and the deployment descriptor (xml file).
- **Horizontal relationships**: They represent different kinds of semantic links in the same file and linking artifacts of the same granular level. This is particularly the case of the call relationship between two methods or the inheritance relationship between two classes, ...
- **Vertical relationships**: They connect two artifacts belonging to the same file at different granular levels. An example of this type of relationship is the one between a class of these attributes, a method body or block the instructions that compose it...

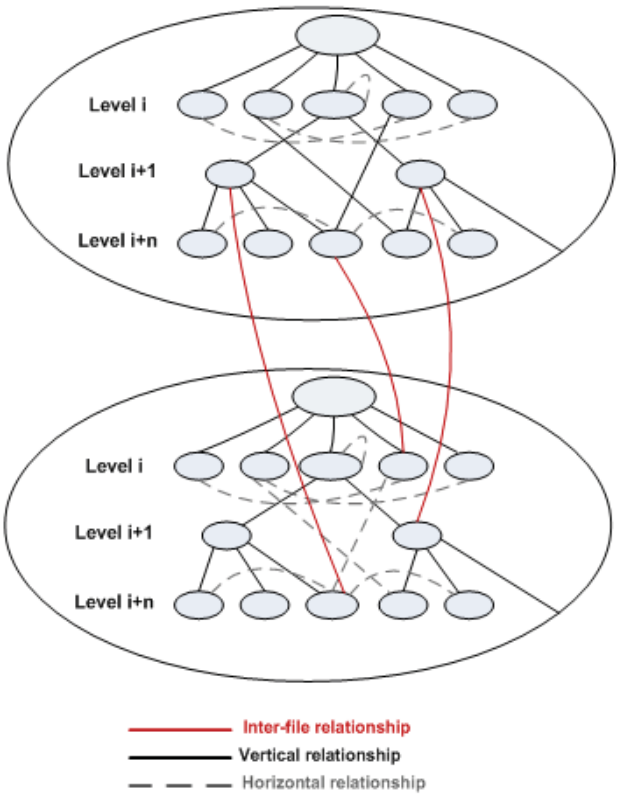

Fig. 1 Software artifacts classification by file type

B. U2MHA: The Construction Meta-Model

The platform that we have developed analyzes and extracts the heterogeneous artifacts by different kinds of parsers that we defined by file type and level of artifact and allows a representation of these artifacts homogeneously in the meta-model U2MHA proposed in Fig. 2.

To improve the performance of our platform, we propose a unified approach to extract only the artifacts selected by evolution expert, it means that the developer responsible for the evolution specifies the different artifacts to extract by level and it is through a web form (web interface) choosing file type and artifact levels to extract. For more complex artifacts such as compounds artifacts, text files and documents specifying business rules, we adopt an approach that is to define them manually by the expert of evolution. This means that artifacts are extracted manually (example above) or by specific algorithms implemented by file type and level of artifact such as java and xml files.

Fig. 2 shows the meta-model U2MHA that we proposed to describe the structure of uniform representation of artifacts [13]. Specifics algorithms have defined by file type and level of artifacts and other manually added by experts following the adopted approach. Listeners adapted by level are listening different construction operations (create, modify, delete) to monitor each artifact in the meta-model of construction, respecting the temporal order of the different operations (versioning) for better control of software evolution.

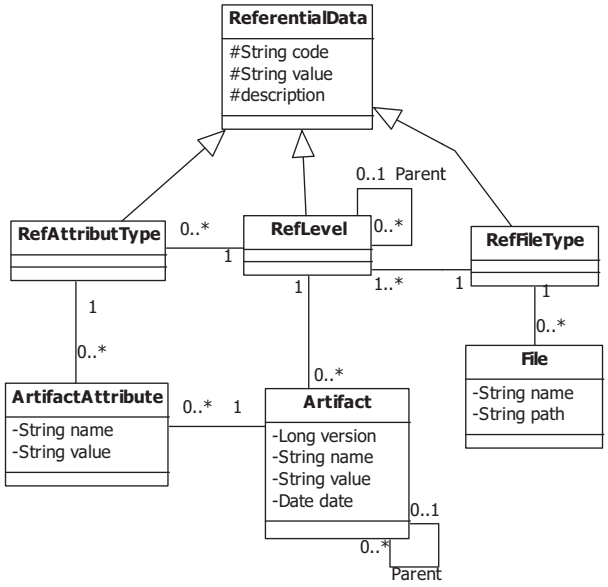

Fig. 2 (U2MHA) Unified meta-model of heterogeneous artifacts

We tried to provide a simple and scalable meta-model to represent all types of artifacts. We considered that all artifacts are stored in files. Each file type must have abstro-granular hierarchical levels (e.g.: java file, the class level that contains methods which can find the parameters). The artifacts may also be represented in a hierarchical manner, an artifact can have elements (artifacts), and each element may have other elements (artifacts), and so on. Each artifact belongs to a level and may have other attributes by level set manually by experts.

III. Extraction Mechanism (Jobs)

A job is a recording of artifacts extraction parameters by file type and abstro-granular level, perform by developers through web application developed (forms). A job can be started manually by the user of the application as it can be launched through listeners or programmed batch.

Fig. 3 shows the process of creating and updating extraction jobs.

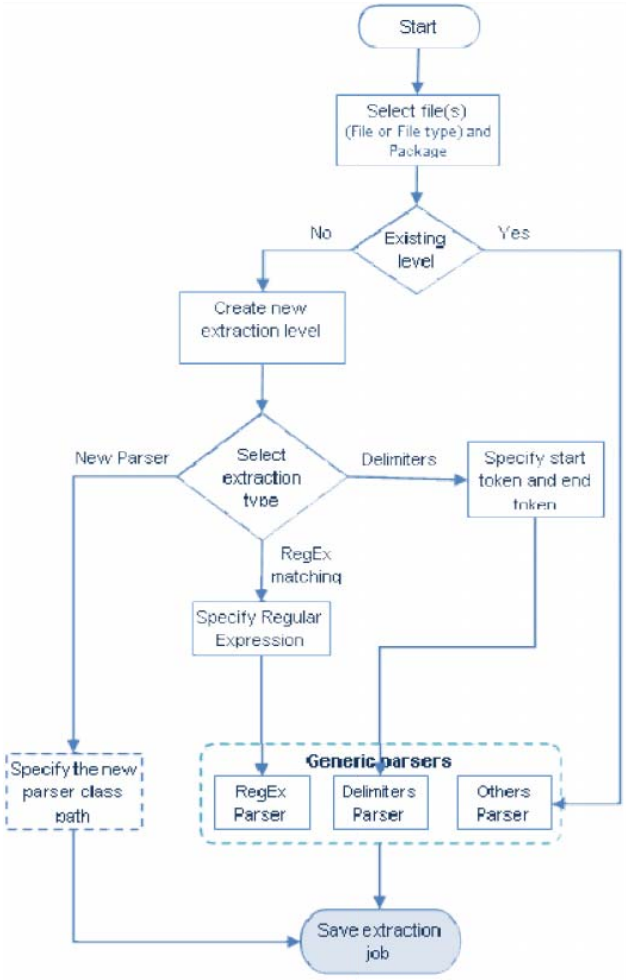

Fig. 3 Jobs creation process

Our web platform provides to developers an intuitive interface for manage extraction jobs of artifacts (add, modify, delete and launch). The user must select the file (s) designated by the extraction (or several specific file by selecting file type), then it can choose an existing abstro-granular level (using parsers supplied with the application) or create a new one. To create a new level the user must choose among three types of extraction:

- **Delimiters:** allows artifacts extraction framed by the tokens (start and end) user-specified, this type of extraction uses a generic parser DelimitersParser.
- **RegEx:** for extracting artifacts respecting the user-specified regular expression, this type of extraction uses a generic parser RegExParser.
- **New parser:** Give users the ability to mention a new parser specific to their needs.

The meta-model EJ2M (Fig. 4) presents the data of different extraction jobs created by users through the web application.

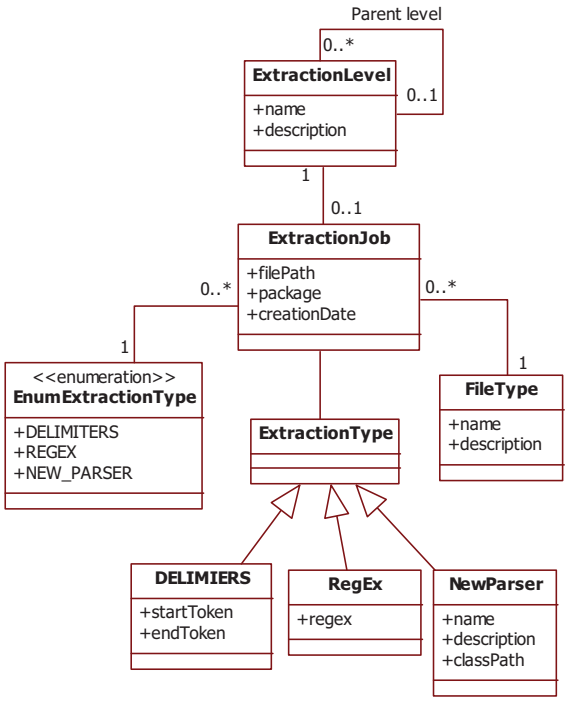

Fig. 4 EJ2M: Extraction Jobs Meta-Model

The launch of created jobs is translated by unified representation of all software artifacts required in a homogeneous form (Fig. 2). This is all done as a web application within the Eclipse platform. In other words, the developer has a uniform representation of all heterogeneous artifacts without leaving his usual work environment and therefore it is possible to simultaneously conduct changes on the platform and an activity analysis in order to improve the quality and consistency of data throughout the development process. The prototype allows users to add new parsers and customize the extraction by adding new jobs to treat other files types, in addition to modify those already defined for a specific extraction management.

IV. Validation Prototype

For operational validation of the proposed approach for software evolution, we have implemented an integrated platform and hosting various modules and components that implement each one of the aspects of evolution discussed in this article.

Our platform and all the modules that constitute were implemented with Java language by Eclipse IDE (Integrated Development Environment) as web application shared and scalable. The peculiarity and the central objective of Java is that written in this language software should be easily portable to multiple operating systems such as UNIX, Windows, Mac OS or GNU / Linux, with little or no modification. The choice of Eclipse was dictated by the fact that it is an open source IDE interesting a number of increasingly growing developers worldwide [14]. This is, in our opinion is the best way to disseminate our implementations within the developer community for which they were designed in the first place.

This, we think he should promote the use and test our implementations and therefore their continuous improvement. It is also a good opportunity to find partners testing our tools on real projects sizes.

A web project consists of a set of resources such as source code files, xml documents, properties files, texts, etc. The platform we have developed allows retrieving and viewing heterogeneous artifacts by different kinds of parsers (Parsers) and allows a representation of these artifacts homogeneously in the meta-model U2MHA previously proposed.

We implemented parsers by file type (java, properties, xml, text ...) and libraries in java to retrieve artifacts semi-automatically. It also allows to listen to the different construction operations (add, modify, delete) to update the repository artifacts. We also implemented a web application accessed by everyone, to customize the extraction file type and level of artifact through a simplified web form instead of extracting all the software artifacts. Through the web interface of this application, you can modify the technical parameters of the application, search and view the extraction results (artifacts).

For more complex artifacts such as compounds artifacts, text files and documents specifying business rules, we adopt an approach that is to define it manually by the expert of evolution. That means that the repository of software artifacts is extracted by different parsers and different static analysis tools and also by information provided by the experts or development actors (architects, quality specialists, etc...).

In this section, we describe the structure and design of the validation prototype of the approach of semi-automated construction of artifacts that we offer. The prototype still allows analyzing different artifacts constituting a software application and was primarily experienced in the case of distributed applications Java type.

The prototype contains:

- The Parsers (parser by file type) to analyze the different file types to feed the referential database of software artifacts (see meta-model U2MHA – Fig. 2).
- A uniform mechanism of extraction based parsers and allows you to customize and develop new extractors by file type and abstro-granular levels.
- A web application allowing developers to specify the types of files and abstro-granular levels involved in the extraction, customization and development of other types of parsers to support other file types by applying and search and view the referential of artifacts.

A. Overall Architecture of the Validation Prototype

Our platform, called Archi Artifacts is a modular tool of extraction and management of heterogeneous artifacts of distributed applications. It is built using Eclipse. Eclipse is an open source platform widely used to build an open and extensible development platform consisting of tools and runtimes for building, deploying and managing software including all phases of the life cycle. Eclipse provides a set of tools assisting the developer in every stage of development

and testing to deployment. We developed parsers, an extraction strategy and a web project designed to manage and customize the extraction of heterogeneous artifacts affecting distributed applications. Fig. 5 shows the overall architecture of Archi Artifacts. Archi Artifacts comprises three major modules, are: Artifact Parser, Artifact Extractor and Artifact Data Visualizer.

The internal architecture showing the interaction of these modules is shown schematically in Fig. 5. We discuss the implementation details of these three modules in the following sections.

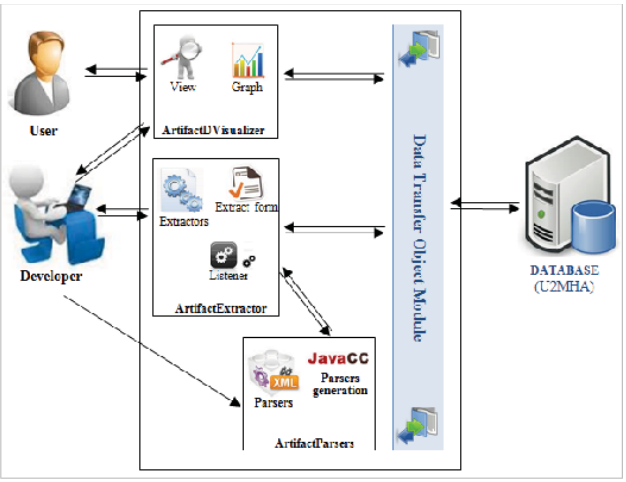

Fig. 5 Validation Overall architecture of the validation prototype

B. Artifact Parser Module

Artifact Parser is a module of our platform including parsers by file type and artifact level, allowing analysis of source documents of various types of files and programming language (+ + Java, C, C, Perl, PHP, COBOLE, text, xml, doc, etc.). Parsers have been developed using Java Compiler Compiler (JavaCC) and other existing open source parsers. JavaCC is a compiler compilers and a lexical analyzer generator entirely written in Java and integrated the Eclipse platform. It allows to interpret the grammar of a given language and to generate a parser. This simplified us the opportunity to add new file types as the progress of the development of our prototype. We have developed many parsers also includes a parser of source code and Java byte code. For the latter parser, a step of decompilation was necessary. So we integrated a Java decompiler called JreversePro. It is an open source Java decompiler used under the GNU GPL license. A source from the input files, ArtifactParser module is used by the Artifact Extractor module for generating various software artifacts and their relationships as specified by our U2MHA model.

C. Artifact Extractor Module

A modular and scalable architecture, including an extraction mechanism that allows management of extraction jobs through a graphical interface and allows developers to add other file types dealing other extraction modules. The module contains listeners of modification actions on selected file types, using extractors to update the U2MHA repository.

The *Artifact Extractor* module is a strategy that includes specific implementations using different parsers (depending on the type of files to scan) to generate different software artifacts and their relationships as specified by our U2MHA model. This mechanism was implemented using the design pattern Strategy. The Strategy pattern is a software design pattern that enables an algorithm's behavior to be selected at runtime under certain conditions. The intended use of this Strategy is to enable evolution experts to customize or create a new specific extraction code for other file types. The result of the execution of *Artifact Extractor* module in other words the generated artifacts (stored in the repository of artifacts U2MHA) is used either for research and consulting software artifacts by Visualizer or by other tools of analysis and management of data consistency.

D. Artifact Data Visualizer Module

It allows viewing the extraction results, history and the relationship between the different artifacts, and can be evolved perhaps by the developers to meet the expectation of users.

Artifact Data Visualizer is a visualization module multiple of our referential of artifacts (extraction result), we have implemented a web interface based on HTML5 and CSS3for the research and consultation of the various software artifacts. We are currently working on the visualization as graphs and manipulation of these graphs to define the various operations. We will use the framework Java Universal Network / Graph (JUNG). This is an open source library that can be reused for modeling, analysis and visualization of data as graphs or networks. The library allows the definition of the graph data structure and use some primitive user interfaces associated with handling tools graph construction.

V. Conclusion and Prospects

In this article, we have presented an approach for extracting heterogeneous artifacts on a simplified and unified meta-model. This approach can be used by several computer tools for better software development evolution. It includes a unified meta-model of representation of different heterogeneous artifacts and a uniform and modular mechanism of extraction to support information engineering in software development projects. And we validated our approach by developing a web platform to better use by the developer community.

Through its modular architecture, our web prototype serves as a good basis to implement a whole chain of processing tools above it. Many tasks in the software development and reverse engineering require not only to retrieve information that interesting to us, but also to deal with a certain special rules. Tasks such as pretreatment, refactoring, consistency validation, etc., all require a custom extraction tool. For example, block modification of any entity before updating the class in the UML model. This example shows that our approach is a good basis for the implementation of tools and languages for the development of software engineering.

Our implementation works continue and we are currently using this approach to manage the consistency of heterogeneous software artifacts.

REFERENCES

[1] M. Ambros, M.Lanza, "Reverse Engineering with Logical Coupling", WCRE '06 : In Proceedings of the 13th Working Conference on Reverse Engineering, IEEE Computer Society, pp. 189-198, 2006.
[2] M. Lungu, M.Lanza, T. Girba, R.Heeck, "Reverse Engineering Super-Repositories", In 14th Working Conference on Reverse Engineering, pp. 120-129, 2007.
[3] H. Kagdi, S. Yusuf, J. I. Ma- letic, "Mining sequences of changed-files from version histories, MSR '06 : In Proceedings of the 2006 international workshop on Mining software repositories, ACM, pp. 47-53, 2006.
[4] F. V. Rysselberghe, S.Demeyer, Studying Software Evolution Information by Visualizing the Change History, ICSM '04 : In Proceedings of the 20th IEEE International Conference on Software Maintenance, IEEE Computer Society, pp.328-337, 2004.
[5] X. Blanc, I. Mounier, A. Mougeno , T.Mens, "Detecting model inconsistency through operation-based model construction", In Proceedings of the 30th international conference on Software engineering, pp.511-520, 2008.
[6] M. Eichberg, M. Mezini, K. Ostermann, and T. Schäfer, "XIRC: A Kernel for Cross-Artifact Information Engineering in Software Development Environments", In Proceedings of the 11th Working Conference on Reverse Engineering, pp.182-191, 2004.
[7] A.Egyed, "Fixing Inconsistencies in UML Design Models", In Proceedings of the 29th international conference on Software Engineering, pp. 292-301, 2007.
[8] X. Blanc, I. Mounier, A. Mougeno , T.Mens, "Incremental Detection of Model Inconsistencies based on Model Operations", In Proceedings of the 21st International Conference on Advanced Information Systems Engineering, pp. 32-46, 2009.
[9] S. Caffiau, P. Girard, L. Guittet, X. Blanc, "Vérification de cohérence entre modèles de tâches et de dialogue en conception centrée-utilisateur", Revue des sciences et technologies de l'information, ISI, vol. 16, no. 5, pp.9-41, 2011.
[10] C. Nentwich, L. Capra, W. Emmerich, A.Finkelsteiin, "xlinkit: A Consistency Checking and Smart Link Generation Service", ACM Transactions on Internet Technology, TOIT, vol. 2, no. 2, pp. 151-185, 2002.
[11] H. Ehrig, U. Prange, G.Taentzer, "Fundamental Theory for Typed Attributed Graph Transformation, Graph Transformations", In Second International Conference, ICGT 2004, Springer 2004.
[12] A. Ahmad, H. Basson, L. Deruelle, M.Bouneffa, "A knowledge-based framework for software evolution control", in 27th INFormatique des ORganisationsetSystèmesd'Information et de Décision (INFORSID), pp. 26-29, 2009.
[13] M. Zekkaoui, A. Fennan, "Consistency Management of Heterogeneous Software Artifacts", International Journal of Computer Applications, vol. 78, no. 14, pp. 35-41, September 2013.
[14] G. Goth, "Beware the March of this IDE: Eclipse is overshadowing other tool technologies", IEEE Software, vol. 22, no. 4, pp. 108-111, August 2005.