MITRAS - Inteligent Model for Software Application Transformation

Lucas Silveira Kupssinskü[1]*, **João Carlos Gluz**[1]

[1]PPGCA - Programa de Pós Graduação em Computação Aplicada
Universidade do Vale do Rio dos Sinos (UNISINOS)
93.022-750 – São Leopoldo – RS – Brazil

lucas.kup@gmail.com, jcgluz@unisinos.br

Abstract. *Software maintenance is costly for software developers and customers as well. This work presents a model of software transformation that can be used to allow new features to be implemented without programming knowledge. To achieve this, an abstract graph representation of the software is proposed, this representation can then be modified through graph transformations.*

1. Introduction

Software maintenance may be a burden for software developers and users as well. Often there is demand for small changes while more important projects are due, and users often needs to wait for long schedules and costly services. Some degree of autonomy in software changes, without requiring programming skills, could be beneficial to both sides. To achieve this goal, techniques in program transformation and synthesis are needed.

Although some maintenance activities are specific or requires great refactoring and changes in business rules, we claim that a significative number of activities can be mapped through graph transformations, given that the user choose how he want it to be applied. According to maintenance types as defined in the SWEBOK [Bourque et al. 2014], the proposed model can be applied to adaptive and perfective maintenances.

To allow software to be manipulated by non-developers, the system must have both domain and programming knowledge [Rich and Waters 1993]. The domain knowledge is used to communicate with the user while the programming knowledge is used in synthesizing the software after transformation. Both domain and programming related knowledge are represented in some extent in source code, to our purposes this information is extracted and manipulated as a directed graph

2. Proposed Solution

Mitras objective is to provide a model that enables software modification without explicit programming. In this regard, figure 1 shows the proposed model flow, dashed lines indicate activities that doesnt need to be executed in every transformation.

In the presented approach, the first step is to mine the source code of the desired software, searching for entities such as classes, associations dependencies, configuration files, html pages and to construct a graph embedding this information. One important particularity in the present step is to construct nodes enabling navigation to source code if

*Bolsista do CNPq - Brasil

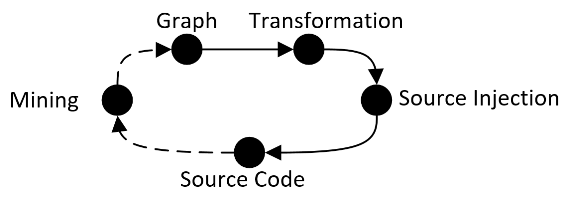

Figure 1. MITRAS Model

necessary. Since each transformation is applied in the graph level, the source code mining only needs to be applied once to the source code, but if necessary it can be applied again an arbitrary number of times.

The extracted graph merges information found in parse trees to elements of the software that are not compiled, such as configuration, properties and html files. Since the graph is directed, all edges have a certain direction that maps some meaning to the relation it represents, for instance, in a inheritance relationship it can show what is the super class. Labels also carry information, they identify nodes as classes, jsp, methods and attributes. In the designed graph, both labels and the direction of edges carries semantics of the model.

Taking as input the graph, the core of Mitras models lies in the transformation capabilities. Using the single pushout technique [Ehrig et al. 2015], specifics subgraphs are pinpointed in the original graph trough isomorphism and are transformed into other graphs with new features embedded. Theoretically, a transformation could be applied in an arbitrarily sized set of subgraphs given that the isomorphism was found, to make the transformation a more manageable tool we restricted the matching step of the transformation to find only one isomorphism at a time.

The changes in the graph model are ultimately transposed to source code in a process we call injection. Source code injection is done identifying elements manipulated in the transformation, such as adding nodes and edges, and doing the equivalent operations in source code and configuration files. After source injection, the code is ready to build and to deploy a new version.

3. Final Considerations

This paper has presented a model for software transformation, showing how graph transformations can be applied to synthesize new features in a software. Through source code mining, it was possible to build an abstract graph with architectural information of the system without exposing all the details regarding implementation. This level of abstraction is sufficiently expressive to provide a base for different graph transformations, the next steps in this research is to find an arbitrary system and execute a controlled experiment.

References

Bourque, P., Fairley, R. E., et al. (2014). *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press.

Ehrig, H., Ermel, C., Golas, U., and Hermann, F. (2015). *Graph and Model Transformation: General Framework and Applications*. Springer.

Rich, C. and Waters, R. C. (1993). Approaches to automatic programming. *Advances in computers*, 37:1–57.