A Global Repository for Software System Models

Mircea Lungu
Software Composition Group
University of Bern
Switzerland
Email: mircea@lungu.org

Richard Wettel
Faculty of Informatics
University of Lugano
Switzerland
Email: richard.wettel@usi.ch

***Abstract*—A large part of reverse engineering research is based on building models of target software systems and then analyzing those models. Due to the shortage of collaboration between researchers and to the high cost of coordination, the same model often ends up being built multiple time by different researchers, which leads to wasted time and resources. Moreover, since many analysis techniques require a distinct model of every snapshot of every version of a software system, this duplication happens for every individual version of the system that needs to be analyzed. However, building the model for a version of a system with a version of a parser should only be done once. In this paper we argue for the construction of a unique repository for models of software systems.**

One added advantage is that for a unique version of a software system, we can collect multiple models extracted by fact extractors with multiple techniques and allow a cross-pollination of analysis techniques by providing a unique place where the results of such an analysis are available.

I. Introduction

Parsing a software system to build a model from it is usually resource intensive and time consuming. In our experience as researchers, we often tried to sidestep it by asking a colleague whether he already had a model for a given system or for the given version of the system that we were interested in. Many times we were successful inside our own group, but even there we had to pay the price of time and communication overhead.

Another problem that reverse engineering researchers face is finding case studies for new techniques and tools. It would be preferable if there was a central repository of models that would serve as the library of models where one could easily go and pick the models that he wants to analyze.

To address these two problems, we propose a global repository of software system models, which eases both the sharing and the discovery of existing models. The advantages of such a repository will be twofold:

- It will provide a common place where researchers can find case-studies
- It will provide a place where researchers can store and combine the results of multiple analyses for an individual system

The contributions of this paper are the following: (1) We identify the need of a central repository of models; (2) We propose a structure of the database that supports such a global model repository; (3) We discuss the problems that could be associated with such a model repository.

In Section 2 we present the architecture of the repository. In Section 3 we discuss various issues that are associated with such a global repository. In Section 4 we conclude and look at future work.

II. The Repository Meta-model

Many types of analysis involve analyzing multiple versions of a software system. In his thesis, Gîrba introduced the Hismo meta-model for representing a software system [**?**]. In his meta-model, each structural entity in a system is modeled distinctly for each version of the system. At the highest abstraction level, the system is modeled as a sequence of system versions.

In our repository, the stored models will be associated with each individual version of a system. As a result, each version of the system will contain multiple associated models corresponding to various parsers, versions of extractors, and types of models. The repository will keep track of the information that is summarized in Figure 1.

The bits of information captured in the repository and illustrate in Figure 1 are:

- **Versioning Control System.** Many software systems have changed their versioning control system during their history. Therefore, we store the versioning control system that is used for each of the versions of the system that are modeled.
- **The Previous Version.** Normally, the version identifiers are numerical and the order can be inferred by sorting them. However, in the cases where a system changes from one versioning control system to another one, this ordering might not work anymore. There are also versioning control systems where the version identifiers are not necessarily numbers that can be ordered so in these cases having the reference of the previous version is needed.
- **Meta-model.** This is a URI that describes the meta-model used to represent this model. For a given system version there can be multiple models built with different meta-models.
- **Fact Extractor.** There are usually multiple versions of a fact extractor. It is important to know which fact extractor has been used for the generation of a model since there might be peculiarities about each version of the extractor.
- **Contributor.** By filling in this field somebody takes responsibility for the correctness of the data.

Version Identification	Versioning Control System	One of the known VCSs: Git, CVS, Subversion.
	System Version	A unique identification of a system. Can be one or more of the following: date/time of checkout, VCS identification number, or release number.
	Previous Version	In some cases the order of the versions needs to be specified
	VCS URL	The URL of the VCS repository.
Model	Meta-Model	A link to the description of the meta-model that describes this model.
	Fact Extractor	Details about the fact extractor that was used to extract this model, such as name and version.
	Contributor	Details about the person that extracted this model. Email or a unique username.
	Construction Date	Date of the creation of the model
Data	Properties	Metrics that summarize the properties of this model.
	Information	The actual information about this model.

Fig. 1. Informations about each of the model versions stored in the Global Repository

- **Construction Date.** Construction date is extra information that might be useful for the presentation of the models.
- **Properties.** The properties are metrics that summarize the model. They are useful for building a UI which would allow searching the model space for case studies that meet certain criteria (*e.g.* systems of a certain size).
- **Information.** The information is the actual payload of the model. The information in the model needs to be conforming to the meta-model description.

III. Discussion

A. Choosing the Versions to Import

The repository allows for flexibility regarding the number of versions that need to be modeled for each system. In fact, for different systems, different researchers or practitioners might create different numbers of models based on their needs. Once a model for a given version is in the repository it can be reused by the others. If a model is missing, it can be added and then it becomes available to others.

B. Migrating to different meta-models

Together with the repository we envision a growing body of tools that will allow model transformations. They will be useful for the times when a new meta-model appears and the old models would have to be migrated to the new representation.

C. Comparing Fact Extractors

It is known that fact extractors do not always agree with each other, or even with their own previous versions on metrics for the models that they extract. Having multiple models associated with a system and a version allows discovering problems with the fact extractors themselves.

D. Building applications on top of the Repository

Having such a repository will enable various applications to be built on top of it. One class of applications will enable the browsing and comparing the many models stored in the repository. The comparison could be done using various techniques of visual summarization of the systems:

- Using CodeCity to present the structure and the size of the systems [**?**]
- Using architectural views with Softwarenaut to present the dependency between the modules in the system [**?**]
- Visualizing the System Complexity to understand the types of class hierarchies [**?**]

E. Providing an API for accessing the repository.

The data for each model will be probably stored as files on disk or as entries in a database. To allow abstracting away from these details, we will provide a web API for accessing the models.

F. Relevance for the Moose Framework

We plan to start populating the repository with FAMIX models, the way they are modeled in the Moose analysis framework [**?**]. Having the repository public will allow showcasing the large number of systems that have been modeled and analyzed with Moose.

G. State of the Work

Currently we are in the process of amassing a large number of systems and building models for them. Although we started by building FAMIX models, we intend to keep the Repository open to any other type of meta-model.

IV. Conclusions and Future Work

In this paper we started from the observation that it makes no sense to build twice the model of a version of a system with a version of a particular fact extractor. In order to address this problem we are constructing a global model repository that will allow for a better collaboration between developers.

Having a global model repository would allow for a better collaboration between developers and for easier access to case studies for researchers.