An Architectural Evolution Dataset

Michel Wermelinger and Yijun Yu
Computing and Communications Department, The Open University, UK

***Abstract*—A good evolution process and a good architecture can greatly support the maintainability of long-lived, large software systems. We present AREVOL, a dataset for the empirical study of architectural evolution. The dataset comprises two popular systems from the same domain and using the same component model, to make comparative studies possible. Besides the original component metadata, AREVOL includes scripts to obtain simplified models that nevertheless support rich studies of architectural evolution, as the authors' previous work has shown.**

I. INTRODUCTION

Systematic approaches to software evolution and software architecture are especially beneficial for systems that are large, complex, difficult to maintain, and have many users to continuously satisfy. Architectural design should facilitate system evolution and the evolution process should be aware of the system's architecture and support its conceptual role. The AREVOL dataset aims to support the empirical study of the architectural evolution of successful long-lived systems, in order to help contribute towards the improved understanding and practice of architectural evolution.

This paper describes the March 2015 release of AREVOL[1]. It comprises the architectural data of many releases of two development environments, Eclipse and NetBeans, that use and extend the OSGi component model. Including only two systems may seem too restrictive, but there are advantages to this choice.

First, Eclipse and NetBeans are long-lived, complex, and popular platforms on which many other systems are built. They are what Flyvbjerg [1] calls 'most likely' cases, which for our purposes means case studies that are likely to follow good architectural evolution principles to help them become and remain widely adopted platforms. As Flyvbjerg points out, 'most likely' cases are well suited for the falsification of hypotheses. For example, if the case studies follow some design principles and evolution laws but not others, one could argue those not followed may not be as determinant for achieving good architectural evolution, at least for the class of systems the case studies represent.

Case studies are also useful as rich narratives of concrete and context-dependent knowledge, which Flyvbjerg argues is more valuable than the vain search for non-existent predictive theories and universals. Our previous work [2] is an example of the kind of rich narrative that can be extracted from a case study. We used a subset of AREVOL's Eclipse dataset to conduct a detailed assessment of its architectural evolution against well-known guidelines and principles, including some of Lehman's laws [3] and Martin's guidelines [4], [5]. This not only enabled us to evaluate the relevance of those guidelines and principles for architectural evolution, but also to extract lessons from Eclipse's architectural evolution, e.g. the observation of a stable architectural core and of a systematic architectural change process throughout the system's life.

Second, presenting two datasets from the same domain allows for future comparative studies which may even go beyond architectural evolution, e.g. to analyse the 'feature race' between Eclipse and NetBeans. Moreover, their adoption of the same component model facilitates the reuse of data extraction and processing scripts. It also allows, for example, a study on how OSGi is used in practice, because each system introduced its own extensions to OSGi.

Third, the chosen systems have explicit architectural metadata, which supports more accurate studies compared to those using reverse engineered architectures. In his keynote at the 2009 Working International Conference on Software Architecture, Alex Wolf argued that configuration files are an under-explored source of architectural information. AREVOL, being a collection of OSGi configuration files that describe the system's component architecture, provides such a source in a format that facilitates its exploration.

II. THE DATA

Eclipse and NetBeans each extend OSGi in a non-standard way, and thus the terminology varies across the three. We will use generic terms to achieve a consistent presentation.

In essence, the configuration files of each system state what are the *components*, their *dependencies*, and their provided and required *ports*. For example, a component may not depend on the help component, but may use its ports to add help text. A component may require its own provided ports. For example, the Eclipse UI component provides ports for other components to add menus, buttons, etc. to the GUI, and uses those ports to create the default interface.

A system goes over time through many *builds*, which range from major releases to nightly builds. The system's architectural evolution is the observable change of components, dependencies and ports over snapshots, i.e. builds. We don't include nightly and other small builds in AREVOL, as we deem them of little relevance for architectural evolution.

AREVOL has four folders: `bin` contains the scripts; `Eclipse` and `NetBeans` contain the configuration files for various builds of each system, and the corresponding architectural facts (components, dependencies, provided and required ports) extracted from them; `view` contains files defining the build sequences to be analysed. The rest of the section details

[1]Available from bitbucket.org/mwermelinger/arevol.

the primary data in the `Eclipse` and `NetBeans` folders. Section III explains the format of the extracted facts, and the relevant files in `bin` and `view`.

A. Eclipse

The data is from several builds of the Eclipse Software Development Kit (SDK), the main ones being the *major* or *minor releases* (e.g. 2.0 or 2.1) and the *service releases* that follow them (e.g. 2.0.1). In parallel to the maintenance of the current release, the preparation of the next one starts, going through some *milestones* followed by some *release candidates*.

We downloaded, for each available build, the entire SDK from archive.eclipse.org or its mirrors. AREVOL contains all 4 major, 13 minor, and 30 service releases since the start of the project, and 34 milestones and 20 release candidates. In total, the Eclipse dataset spans a period of 14.5 years from November 2001 to February 2015.

The Eclipse archives only keep the latest milestones and release candidates. Therefore AREVOL only includes those available now and 4 years ago, at the time of our Eclipse study mentioned in the introduction, for which we fortunately kept a local copy. To obtain the missing builds, one would have to use directly the version control repository, which poses its own challenges to accurately retrieve the public releases.

Each build of Eclipse (except for the initial release 1.0) provides one or more high-level *features*, which are used by Eclipse's update manager to allow users to selectively and incrementally upgrade their installation of Eclipse. A feature may be composed of other more specialised features, i.e. features are organised hierarchically. Each feature is implemented by a set of components. Features and components may have the same name.

For each feature there is a metadata file `feature.xml` that lists the feature's sub-features and the components that implement it. For each component there are one or two metadata files (`plugin.xml` and, since release 3.0, `MANIFEST.MF`) that list the component's provided and required ports and the components it depends on. Initially, the dependencies were described in both files, but after a short transition period, they are now in `MANIFEST.MF`, whereas port information is in `plugin.xml`.

We wrote a Bash script that goes through each build and extracts the required configuration files, keeping Eclipse's folder structure. The result is put in one folder per build. The extract below from AREVOL's folder structure shows that features were introduced in 2.0, manifest files in 3.0, and that a component may be described by one or two files.

```
Eclipse/
  1.0/
    plugins/
      org.eclipse.webdav/plugin.xml
  2.0/
    features/
      org.eclipse.jdt/feature.xml
    plugins/
      org.apache.ant/plugin.xml
  3.0/
    features/
      org.eclipse.jdt/feature.xml
    plugins/
      org.apache.ant/plugin.xml
      org.eclipse.core.runtime/plugin.xml
        META-INF/MANIFEST.MF
      org.eclipse.osgi.util/
        META-INF/MANIFEST.MF
```

B. NetBeans

The data was extracted from multiple NetBeans stable source releases, obtained from download.netbeans.org. AREVOL includes all 28 major, minor and service releases from 3.5.1 (July 2003) to 8.0.2 (November 2014), spanning roughly 11.5 years.

Unlike Eclipse, NetBeans doesn't group components into features, but like Eclipse it describes components with a text (`manifest.mf`) and an XML file (`project.xml`). Contrary to Eclipse, the port information is in `manifest.mf` and the dependencies are in `project.xml`. NetBeans also organises the files differently: `manifest.mf` is in the component's root folder, while `project.xml` is in a `nbproject` subfolder.

Like for Eclipse, we wrote a Bash script that creates one folder per build and copies into it the configuration files and their enclosing folders. The extract below illustrates the resulting AREVOL folders and files.

```
NetBeans/
  3.5.1/
    netbeans-src/
      jarpackager/manifest.mf
      beans/manifest.mf
      web/manifest.mf
        taglibed/manifest.mf
        servletapi23/manifest.mf
      j2eeserver/manifest.mf
  4.0/
    nbbuild/manifest.mf
      misc/manifest.mf
        nbproject/project.xml
    beans/manifest.mf
      nbproject/project.xml
    web/manifest.mf
      servletapi24/manifest.mf
      project/manifest.mf
        nbproject/project.xml
      jspdebug/manifest.mf
        nbproject/project.xml
```

III. THE ARCHITECTURAL MODELS

Besides providing the original configuration files, so that other researchers are not restricted in their studies, we include in AREVOL basic architectural facts for each build, extracted from those files, and a script that users can adapt to construct a simple architectural evolution model over a sequence of builds of their choice.

A. The snapshot model

The architectural facts were extracted by AWK and XSLT scripts we wrote and ran on the text and XML configuration files. The result is one text file per build, in the relational Rigi Standard Format [6]. The files are named *system*`/`*build*`.txt` with *system* being `Eclipse` or

NetBeans, and *build* being the build number. Each line of those files is one of the following tuples:

- FEATURE *featureName*
- COMPONENT *componentName*
- CONTAINS_FEATURE *featureName featureName*
- CONTAINS_COMPONENT *featureName componentName*
- PROVIDES *componentName portName*
- REQUIRES *componentName portName*
- DEPENDS *componentName componentName*

The above capture which features and components exist in each build, their hierarchical relations, their dependencies and their ports. We need two containment relations because components and features may have the same name: a tuple CONTAINS *feature name* wouldn't tell whether the second name refers to a feature or a component. Note that only Eclipse has features, and hence the FEATURE and CONTAINS_... relations don't exist for NetBeans.

We also provide, for each system, an RSF file view/*system*-all.txt that lists the builds included in AREVOL. The files have in each line a tuple

BUILD *buildNumber buildDatetime*

with *buildDatetime* of the form YYYYMMDDHHMM.

B. The evolution model

The first stage towards an architectural evolution model is to define the sequence of builds to be considered. For example, one may wish to study how the architecture was transformed from one particular release to the next through the corresponding sequence of milestones and release candidates. To specify such a sequence, the user simply makes a copy of the *system*-all.txt file and removes the tuples about the builds to be ignored. The sequence is determined by the builds' dates, not their numbers. As an example we provide, for each system, a file view/*system*-major.txt that only includes its major releases.

The second decision is how to map the snapshot model to Briand et al's simple structural model [7] that forms the basis for an axiomatic software metric framework. In their model, a system consists of *modules*, *elements* and *relationships*. A system is represented as a directed graph, with elements as nodes and relationships as directed edges, and a module is a subset of the elements, and all the existing relationships among that subset.

The model is very flexible: by choosing what are the modules, elements and relationships, the same system can be analysed at different levels of granularity, provided the data is available, and those analyses can be compared. For example, if source code is available, modules may be Java packages, elements may be classes and relationships may be calls between classes. In our architectural context, the elements are always components, but relationships can represent component dependencies or the usage of ports, i.e. an edge from A to B states that A provides a port that B requires. Modules can be features, name spaces, or any other grouping of components, and different modularisations can be compared, e.g. with clustering analysis or by measuring the modules' average coupling and cohesion.

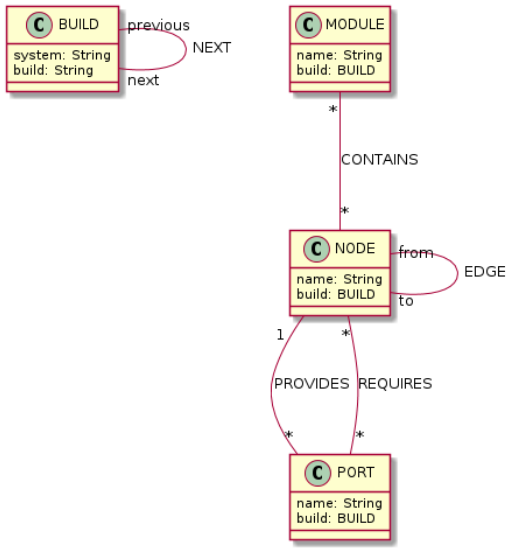

Fig. 1. An architectural evolution model

The graph of elements and relationships and the containment of elements in modules can both be represented in a relational way, similar to the snapshot model. We therefore use the freely available relational calculator Crocopat [8] to transform the snapshot model to Briand et al's model. The transformation is defined by scripts bin/*system*.rml, one for each system. As provided, both scripts map a component to an element and a dependency to a relationship. The script for Eclipse maps a feature to a module, whilst the script for NetBeans creates two modules, one with components named org.netbeans..., and the other with third-party components like Ant. To generate a different relationship or modularisation, the user has to edit the scripts.

Once the build sequence and the model transformation have been defined, the user runs the evolution Bash script in the bin folder. This script takes the RSF file with the build sequence as argument, calls the user-edited Crocopat script, and generates to standard output a list of RSF tuples:

- BUILD *system build*
- NEXT *previousBuild nextBuild*
- MODULE *build name*
- NODE *build name*
- CONTAINS *build moduleName nodeName*
- EDGE *build fromNodeName toNodeName*
- PROVIDES *build nodeName portName*
- REQUIRES *build nodeName portName*

The first relation captures which builds are collected in this file, with *system* being either Eclipse or NetBeans and *build* being a build number. The NEXT relation captures the build sequence. The first build in the sequence is the one never appearing as the second argument of NEXT. All other tuples then refer to a build from the sequence, so that a single

file captures the whole evolution in a self-contained way. The following four relations capture Briand et al's model. Note that `ELEMENT` and `RELATION` are reserved words in Crocopat, so we use `NODE` and `EDGE` instead. Lastly, the `PROVIDES` and `REQUIRES` tuples are copied from the snapshot model for each build, so that the evolution of provided and required ports can be analysed.

To illustrate the architectural evolution model, AREVOL includes in the `view` folder the result of running `bin/evolution view/`*`system`*`-major.txt` for each system.

IV. USAGE SCENARIOS

Besides the comparative analysis suggested in Section I, the architectural evolution tuples extracted for a given build sequence can be used for a variety of analyses, for example:

- What is the stable core, i.e. which elements and relations exist throughout all builds in the sequence?
- Are there any cycles in the relations (strong coupling) and are they removed over time?
- Is the system increasingly open for extension, i.e. is the number of provided ports growing?
- Is the architectural change process restricted to only certain types of builds?
- Do modules become more cohesive and less coupled over time?
- Are the defined modules a good modularisation of the architecture, e.g. are architectural co-changes mostly encapsulated within modules?

We have addressed these and many other questions in our assessment of Eclipse's architectural evolution [2], using 2 different build sequences and 3 different modularisations of Eclipse. We refer the interested reader to that work to see examples of what can be inferred from the evolution model presented in the previous section.

V. ADVANTAGES

We adopted the RSF format for various reasons: it is plain text and thus universally readable; it can be easily converted into CSV, by replacing spaces by commas, for processing by a spreadsheet application; it is a popular format in software engineering tools; it can be processed by powerful and freely available tools, like awk, sed, and Crocopat [8]. By relying on such a simple text format and writing scripts for those tools, we make it easier to integrate AREVOL with other researchers' own software repository mining tool chains.

The snapshot model is very general. It can be generated from the meta-data of the installation tar balls (as in our case), from the result of applying an architectural reverse engineering tool to the source code, or even from specifications written in some architecture description language. It is therefore possible to obtain evolution models with the same module-element-relation structure for systems that have radically different primary data available. The 'front-end' scripts that compute metrics and visualisations from those models can then be reused without change.

Working at architectural level has the benefit of reducing the memory and computational power needed, because there are far fewer nodes and edges than at implementation level. For example, Beyer et al applied Crocopat to the Eclipse 2.1.2 graph of 7,081 classes and their 59,344 call dependencies [8], whereas each build's architectural graph of Eclipse has only hundreds of nodes and arcs. Scripts can thus run quickly even though data is in RSF text files, and not in a relational database. Moreover, Beyer et al report that Crocopat is much more efficient than MySQL for computing transitive closures of relations, which is important for structural analysis of design in general.

VI. CONCLUDING REMARKS

AREVOL is a data set for empirical studies of architectural evolution. We chose Netbeans and Eclipse as examplars of systems for which a sustained evolution of a good architecture is key to their success as a platform for many third-party projects. Moreover, their use of the OSGi component model allows us to efficiently and accurately extract their architecture. We suggested various studies that AREVOL can support.

Previously we had only published secondary data (the metrics used for architectural assessment) resulting from analysing some of the Eclipse builds. Now we have substantially expanded the primary data, by adding Netbeans and 4 more years of Eclipse builds, and published it, together with new scripts that allow for user-defined sequences and modularisations to ease further studies. Those scripts generate a simple architectural evolution model that can be measured, analysed and visualised. Our previous work is evidence that rich studies can be made from such a model.

We have published AREVOL as a Git repository (see the footnote on the first page), so that scripts can be versioned, future incremental additions to the dataset can be downloaded more efficiently, and other researchers can contribute additional case studies and scripts through pull requests.

REFERENCES

[1] B. Flyvbjerg, "Five misunderstandings about case-study research," *Qualitative Inquiry*, vol. 12, no. 2, pp. 219–245, Apr. 2006.

[2] M. Wermelinger, Y. Yu, A. L. Rodriguez, and A. Capiluppi, "Assessing architectural evolution: a case study," *Empirical Software Engineering*, vol. 16, no. 5, pp. 623–666, 2011.

[3] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski, "Metrics and laws of software evolution - the nineties view," in *Proc. Symp. on Software Metrics*. IEEE, 1997, pp. 20–32.

[4] R. C. Martin, "Granularity," *C++ Report*, vol. 8, no. 10, pp. 57–62, Nov.-Dec. 1996.

[5] ——, "Large-scale stability," *C++ Report*, vol. 9, no. 2, pp. 54–60, Feb. 1997.

[6] K. Wong, *The Rigi User's Manual, Version 5.4.4*, June 1998.

[7] L. C. Briand, S. Morasca, and V. R. Basili, "Property-based software engineering measurement," *IEEE Trans. Software Eng.*, vol. 22, no. 1, pp. 68–86, 1996.

[8] D. Beyer, A. Noack, and C. Lewerentz, "Efficient relational calculation for software analysis," *IEEE Trans. Software Eng.*, vol. 31, no. 2, pp. 137–149, 2005.