Comparison process specification for a repeatable comparison of architecting processes *

(Research-In-Progress, approx. word count 2700)

Rodion M. Podorozhny
University of Texas
ECE Department
Austin, TX 78712
USA
phone: +1 512 232 7931
fax: +1 512 471 3621
podorozh@mail.utexas.edu

Leon J. Osterweil
University of Massachusetts
Dept. of Computer Science
Amherst, MA 01003
USA
phone: +1 413 545 2186
fax: +1 413 545 1249
ljo@cs.umass.edu

Dewayne E. Perry
University of Texas
ECE Department
Austin, TX 78712
USA
phone: +1 512 471 2050
fax: +1 512 471 3621
perry@mail.utexas.edu

February 19, 2001

*This work was supported in part by the Air Force Materiel Command, Rome Laboratory, and the Advanced Research Projects Agency under Contract F30602-94-C-0137.

Comparison process specification for a repeatable comparison of architecting processes

Abstract

We describe experimentation aimed at making the comparison of processes and process specification formalisms more of an exact science. Our aim is to lay the foundations for this more exact science by establishing fixed methods and conceptual frameworks that are able to assure that comparisons will yield predictable, reproducible results.

The focus is on the comparison of architecture description languages (ADLs) by comparing architecting processes. Earlier work on comparing ADLs ([6], [2]) did not use a systematic comparison process. Nor did they take into account the context in which ADLs are used.

This work-in-progress paper concentrates on the comparison process and the design of the experiments. In the first experiment two architecting processes are modeled and executed with the same ADL. The other related experiment uses two ADLs for the same process. Then we intend to compare the product artifacts and particular paths traversed.

Keywords Architecture Description Languages, Architecting Process, Software Process, Process Formalism, Comparison, Base Framework

1 Introduction

1.1 Background

In earlier work ([7], [11]) we have argued that processes should be viewed as products of a process-development activity.

Processes, just like software, must be thought of as being products of thought. They are intellectual products. They can be sensed, understood, and evaluated only by indirect means through their effects and manifestations. They have important dynamic properties, yet obey no laws of Physics and defy physical measurement.

We have also suggested that many of the ideas, approaches, techniques, and formalisms of software engineering should be applied to the engineering of processes as well. In particular, we suggested in [12], [9] that software design notations and software process programming formalisms could be useful in establishing baselines that could be effective bases for the objective classification and comparison of processes. This paper builds upon this work and provides further evidence to support this suggestion. We focus on the effects that the choice of a process has on the applicability and usefulness of ADLs used in software processes.

1.2 Discussion of Problem

We believe that one of the hallmarks of a mature scientific or engineering discipline is its ability to compare and evaluate its artifacts. Comparison, in turn, rests, in large part, upon classification. Thus we believe that the establishment of a discipline of process engineering requires the development of techniques and structures for classifying, comparing, and evaluating processes.

In [12] we have proposed CDM (Comparison of Design Methodologies) and presented a model of this process for the comparison of design processes. There we have also proposed the use of a BF (base framework), a classification schema for organizing the key components of design processes. In [11], [12] we demonstrated the use of CDM and BF in comparing some software design processes.

We felt that it was important to show that the results of a comparison conducted according to CDM are repeatable. In [12] CDM was specified formally in a graphical notation and informally described in text. The formal specification though was at a rather high level of abstraction. We believe that there is a dependency between the level of detail of a process specification and the likelihood of the repeatability of results obtained. In other words, the greater the detail level, the more likely the results will be repeatable. Therefore, to further increase the potential for repeatability of the results, we specified two main substeps of CDM (namely, Classify_Components and Make_Comparisons) in greater detail. For the specification of these substeps we used Little-JIL, a software process language ([13]).

In earlier work we evaluated the sensitivity of a comparison of software design processes to the modeling formalisms used ([8]). Here we evaluate the sensitivity of a comparison of architecting processes (APs) to the ADL used.

1.3 Proposed Experiments

The experiments were designed to determine how repeatability is influenced by variation in the architecting process and in the ADL. In the first of the two experiments we evaluated how the comparison results produced by the CDM process are influenced by the architecting process. The second experiment measures the influence of the ADL on the comparison results produced by the CDM process.

The explanation of the first experiment follows. The reader is advised to refer to the diagram in Fig. 1. The goal of this experiment was to compare the results obtained by the use of two different architecting processes with a similar purpose, in this case discovery of a software system's architecture, by executing CDM, enhanced by precise JIL specification and using a single fixed classification schema (depicted in Fig. 1 as CS). The modeling formalism (MF) to be used is Little-JIL ([13]). The two architecting processes are "Focus" by Lei Ding and Nenad Medvidovic ([3]) and "Renaissance" (*www.comp.lancs.ac.uk/projects/renaissance*), these choices are explained in sections 2.3– 2.6. We intend to compare the results obtained by the two processes and draw conclusions about the influence of the process differences. We will refer to this experiment as the

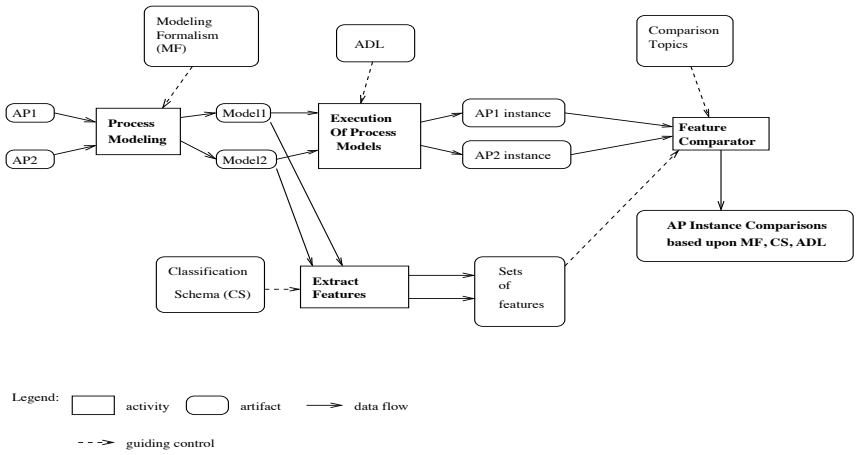

Figure 1: Model of the Two Processes experiment

Two Processes experiment throughout the rest of the paper.

The second experiment is aimed at evaluation of the sensitivity of the comparison results to the choice of ADL selected. Please refer to Fig. 2 for the diagram of this experiment. In this experiment, a single architecting process is first executed on some input with ADL1 as a language to represent its architecture related artifacts. Next the same architecting process with the same input is executed with ADL2 instead of ADL1. We then intend to compare the results obtained by the two executions of the same architecting process and draw conclusions on the influence of differences in ADLs. We will refer to this experiment as the Two ADLs experiment. Both experiments will use a combination of classification schemas for software processes (extension of BF suggested in [9]) and a classification schema for ADLs (extension of classification in [6]). The latter is used for comparison of artifacts represented in an ADL. "Wright" ([1]) and "C2ADL" ([5]) have been chosen as ADLs for the

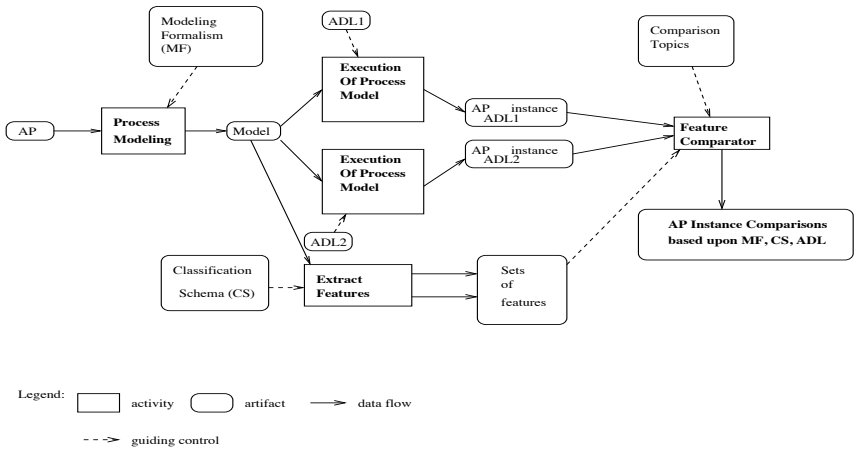

Figure 2: Model of the Two ADLs Experiment

experiments. The rationale for these choices is briefly described in sections 2.3– 2.6.

2 Approach

2.1 Comparison of the results of two architecting processes as a way to determine the influence of the ADL used

Fig. 1 models a comparison process that is similar to CDM (described in [12]). This model emphasizes the major functional components in the process and focuses on comparing architecting processes. The version of CDM in Fig. 1 pays attention to comparing particular process instances as opposed to process "templates". The functional decomposition provides a conceptual framework that seems convenient as the basis for establishing a discipline of process comparison. The figure

also shows dependencies between these functional components. Note that although the diagram is strictly sequential, the process of comparison may be iterated using different formalisms and classification schemas.

The figure highlights the critical role played by the choice of MF, the process modeling formalism and CS, the feature classification schema, and the choice of an ADL. Any formalism allows us to see only those components of an architecting process that can be expressed in it. A classification schema allows us to compare only the components in those classes and categories that it includes. Thus, if an architecting process has an aspect not captured by the formalism and/or schema, that aspect will not be considered and hence comparison results may be skewed and/or inaccurate. Similar reasoning applies to an ADL used to capture artifacts of an architecting process.

We now use functional notation to express the comparison process of Fig. 1 more rigorously. The process consists of four principal functional transformations:

- $\boldsymbol{Process\_Modeling_{MF}} : \boldsymbol{AP} \rightarrow \boldsymbol{AP\_Model_{MF}}$, where AP is the space of all architecting processes and AP_Model_{MF} is the space of models of APs in the modeling formalism MF.
- $\boldsymbol{Extract\_Features_{MF,CS}} : \boldsymbol{AP\_Model_{MF}} \rightarrow \boldsymbol{Feature\_Structure_{CS,MF}}$, where $Feature\_Structure_{CS,MF}$ is the space of all feature sets, structured by CS.
- $\boldsymbol{Execution_{AP\_Model,ADL}} : \boldsymbol{AP\_Model_{MF}} \rightarrow \boldsymbol{AP\_Model\_Instance_{CS,MF,ADL,APinput}}$, where $AP\_Model\_Instance_{CS,MF,ADL,APinput}$ is an instance of a particular execution path through an AP_Model_{MF} when given a certain input.
- $\boldsymbol{Feature\_Comparator_{CS}} : \boldsymbol{Feature\_Structure_{CS,MF}} \times \boldsymbol{Feature\_Structure_{CS,MF}} \rightarrow \boldsymbol{AP\_Comparisons_{CS}}$, where $AP\_Comparisons_{CS}$ is the space of comparisons of features identified by CS.

The repeatability of results is an important characteristic of a high quality process. We believe that specification of the comparison process (or any process for that matter) in greater detail will help to increase the repeatability of the results. Therefore, $\boldsymbol{Extract\_Features_{MF,CS}}$ and $\boldsymbol{Feature\_Comparator_{CS}}$ functions have been elaborated in the Little-JIL process specification language ([13]). To ease the understanding of the process principal steps the following sections summarize the steps in an Ada-like language, a language which is more common than Little-JIL and more likely to be known by the reader. The corresponding specifications for the $\boldsymbol{Feature\_Comparator_{CS}}$ step are presented in Figs. 3 & 4. The space limitations do not permit to go into greater detail about the rest of the comparison process. Here we will briefly describe the details of this step in informal natural language.

```
1  PROCEDURE Make_Comparisons(BF_model1: IN classif_table;
2   BF_model2: IN classif_table; artifact_topic_list: IN topic_list;
3   BF: IN class_list; Comp_Results: IN OUT comparison_results)
4  IS
5          loc_class: BF_structures.class;
6          topic_comp: pair_comparison;
7  BEGIN
8          FOR domain_model IN BF.Domain_Models LOOP
9           FOR loc_class IN BF.domain_model LOOP
10           CASE MCTH_Level (loc_class) IS
11                WHEN artifact =>
12                        Compare_Classes( loc_class, BF_model1.loc_class,
13                          BF_model2.loc_class, artifact_topic_list,
14                          Comp_Results );
15                WHEN activity =>
16                        Compare_Classes(loc_class, BF_model1.loc_class,
17                          BF_model2.loc_class, activity_topic_list,
18                          Comp_Results);
19                WHEN concept =>
20                        Compare_Classes(loc_class, BF_model1.loc_class,
21                          BF_model2.loc_class, concept_topic_list,
22                          Comp_Results);
23           END CASE;
24                Final_Summary(Comp_Results, summary_topics, summary);
25          END LOOP;
26         END LOOP;
27 END Make_Comparisons;
```

Figure 3: The code of *Make_Comparisons* step

```
1  PROCEDURE Compare_Classes( Class: IN string, component_set1:
2   IN component_list; component_set2: IN component_list; Topics: IN topic_list;
3   Comp_Results: IN OUT comparison_results)
4  IS
5                   component1, component2: Formalism_Structures.component;
6                   pair_comp: pair_comparison;
7                   pairs_list: array (POSITIVE range <>) of pair;
8                   summary: string;
9  BEGIN
10                 Identify_pairs(pairs_list, component_set1,
11                                component_set2);
12                 FOR component_pair IN pairs_list LOOP
13                         Compare( component_pair.component1,
14                         component_pair.component2, topic_list, pair_comp );
15                         Add(Comp_Results, pair_comp);
16                 END LOOP;
17                 Summarize_comparison( Comp_Results, summary);
18                 Add(Comp_Results, Class, summary);
19                 RETURN;
20 END Compare_Classes;
```

Figure 4: The code of *Compare_Classes* step

2.2 The $Feature\_Comparator_{CS}$ step

The reader is advised to refer to Figs. 3, 4 for the Ada-like specification of this step. The purpose of this step is to conduct a comparison of the features that belong to the same CS class according to previously defined comparison topics. Let us give a brief overview of the process.

This process uses the sets of features that were extracted from the two models and deemed relevant to the specified comparison topics. To arrive at the comparison of two software processes, first, a comparison is done between pairs of sets of comparable components in every CS leaf class according to the pair comparison topics, then a summary is done according to class comparison topics for every class of the CS. Consequently, a composite comparison is constructed. Carefully chosen topic lists will help the comparer to see the differences between the processes in the areas of interest.

We can now define the entire process of using CDM, MF, and CS to compare two instances of APs requiring an ADL and some input as (Fig. 1):

$$CDM_{MF,CS,ADL,AP1,AP2,AP\_input} : AP \times AP \rightarrow AP\_Comparisons_{CS} ,$$

Similarly, the process of comparing the influence of an ADL on an architecting process can be represented as (Fig. 2):

$$CDM_{MF,CS,ADL1,ADL2,AP,AP\_input} : AP \times AP \rightarrow AP\_Comparisons_{CS} ,$$

The ***Process_Modeling*** function of CDM entails the construction of process models. In the ***Extract_Features*** phase the features of the processes are extracted from the process models, based on the classification schema used. The last phase, ***Feature_Comparator***, takes two structures of features, one per process model, compares them, and outputs a comparison of the two feature structures. In this research we attempt to study the sensitivity of an ADL modeling capability to the context it is used in (i.e. an architecting process). We also intend to study the sensitivity of an architecting process to the ADL used.

This work is a continuation of assessing the influence of the choices of formalisms on the results of software process comparisons. In our previous work we evaluated the influence of a modeling formalism on the comparison results ([8]). Now we are suggesting the use of a similar approach to evaluate the influence of a modeling formalism used by processes being compared (i.e. ADLs). As another objective, we are suggesting a comparison of formalisms in a particular context. We hope that the results of these experiments will enable us to better understand the rigorous methodology of software process comparison.

2.3 The Classification Schema used

Ideally, a comprehensive classification schema should include all the details of all the features of a software process with a particular purpose. In that case, we would be sure that no comparison could overlook any features of the software process models being compared. While it is doubtful that such an ideal classification schema can be created, we believe it is feasible to develop a sequence of approximations to this ideal through iterative creation and modification. Each time a new or different process feature is encountered, we envisage classifying it and putting it in the proper place in the classification schema. Thus, analysts performing comparisons should be prepared to check whether all the features of compared process models are captured by the classification schema used and suggest modifications needed. Ideally, this would be done as a community activity leading to an increasingly broadly accepted classification schema. The classification schema used for this experiment is based on schemas suggested for comparison of software processes and ADLs in [10], [4], and [6].

2.4 The architecting processes used

The following criteria were used to choose architecting processes for our experiments:

- processes have to be of similar purpose so that their comparison would make sense
- processes have to explicitly mention the use of an architecture (supposedly captured in an ADL)

- processes have to be defined in a formal way with the use of some notation that can be regarded as a modeling formalism
- processes have to be developed by people other than the comparers

Architecture discovery processes suggested by Lei Ding and Nenad Medvidovic ([3]) and the RENAISSANCE consortium (*www.comp.lancs.ac.uk/projects/renaissance*) have been chosen for the experiments because they satisfy these criteria. They are similar in purpose as both strive to recreate a software system's architecture based on its code as input. Both processes also support evolution of the software system. The experiments imply executing these processes in a process execution environment with the same input. Small (5000-7000 lines) software systems are intended to be used as input to the processes in the experiments. The processes are not completely automatic thus another requirement is that the software system used is not developed by the person aiding in the processes execution.

2.5 The ADLs used

The ADLs to be used for experiments should not have widely different, explicitly stated problem domains (i.e. it is more likely that an ADL intended to capture a real-time system will be somewhat different in the emphasis of its expressiveness from an ADL intended to capture a GUI system; whether it is indeed so might be a topic of further experimentation). Wright ([1]) and C2ADL ([5]) have been chosen for the experiment. Wright seems to be a "general purpose" ADL while C2 is leaning towards GUI software systems. The emphasis of the experiment is more on validating the comparison process than obtaining comparison results for a comprehensive set of ADLs.

2.6 The modeling formalism used

Little-JIL ([13]) has been chosen as the formalism because it is a rather comprehensive process language and the processes specified can be simulated with some degree of automation in the corresponding execution environment.

3 Future Work.

We expect that the experiments described above will show the features of architecting processes and ADLs that contribute to the differences in both quantitative and qualitative ways. Intuitively, it is the expressiveness of ADLs and possible variations of the execution of an architecting process that will make the processes execute differently with the same input. The comparison will enable us to highlight the relationship between the kinds of activities and artifacts of compared processes, i.e. it will allow to determine the intersection of processes. More experiments will be needed to validate the comparison process, so further experimentation is a direction worth pursuing.

The comparison process itself can be evolved. One evolution direction is mathematical formalization of the classification and functional comparison steps. The problems to be solved include the rigorous

mathematical modeling approach of certain aspects of processes and the discovery of useful process properties that can have rigorous definitions amenable to verification. Higher degrees of automation of the classification step can also be achieved by the analysis of functional correspondences between process steps and artifacts expressed in mathematical notation.

It also seems important to set this line of research on a more solid foundation of rigor by using formalisms to define the CS comparison framework and the features within. A formalism capturing the structure of comparison topics also will be very helpful. The comparison topics of our experiment were not rigorously defined (i.e., activities, concepts, techniques, additional properties of methods). Such definitions would certainly make the comparison results more objective, since there would be less room for ambiguity. In addition, the comparer would have clear guidelines concerning the actual comparison of process components and the comparison job would become easier and more repeatable. Furthermore, it would be beneficial to introduce a more formal measure between processes based on the CS used and comparison topics used to compare features identified by the CS.

In closing, it is important to emphasize that this experiment is strongly encouraging in that it indicates that rigorous, reproducible comparison of processes is quite feasible. This line of research was undertaken in reaction to a long string of process comparison work that was completely informal, offering no basis for scientific validation through reproducible experimentation. It was our goal that process comparison be made rigorous, semantically well-founded, and reproducible through the use of formally defined comparison processes (such as CDM), comparison schemas (such as BF), and semantically well-based modeling formalisms. This work continues to provide evidence that this sort of rigor and reproducibility is possible.

References

[1] R.J. Allen. A Formal Approach to Software Architecture. Technical Report CMU–CS–97–144, Carnegie Mellon University, 1997.

[2] P. C. Clements. A Survey of Architecture Description Languages. *8th Int'l Workshop Software Specification and Design*, March 1996.

[3] Lei Ding and Nenad Medvidovic. Focus: A Light-Weight, Incremental Approach to Software Architecture Recovery and Evolution. Technical report, Computer Science Department, University of Southern California, Los Angeles, CA.

[4] Sjaak Brinkkemper Geert van den Goor, Shuguang Hong. A Comparison of Six Object-Oriented Analysis and Design Methods. Technical report, University of Twente, Enschede, the Netherlands, 1992.

[5] N. Medvidovic, R.N. Taylor, and E.J. Whitehead Jr. Formal Modeling of Software Architectures at Multiple Levels of Abstraction. In *Proceedings of the California Software Symposium*, pages 28–40, April 1996. Los Angeles, California.

[6] Nenad Medvidovic and Richard Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, January 2000.

[7] Leon J. Osterweil. Software Process Interpretation and Software Environments. Technical Report CU–CS–324–86, Department of Computer Science, University of Colorado, Boulder, CO, April 1986.

[8] Rodion M. Podorozhny and Leon J. Osterweil. The Criticality of Modeling Formalisms in Software Design Method Comparison,. In *Proceedings of the 19th International Conference of Software Engineering*, pages 303–313, 1997.

[9] Xiping Song and Leon Osterweil. Toward Objective, Systematic Design-Method Comparisons. *IEEE Software*, pages 43–53, May 1992.

[10] Xiping Song and Leon J. Osterweil. The Models of the Design Methodologies. Technical Report UCI–91–19, University of California, Irvine, Irvine, CA 92717, 1991.

[11] Xiping Song and Leon J. Osterweil. Engineering Software Design Processes to Guide Process Execution,. Technical Report TR–94–23, University of Massachusetts, Computer Science Department, Amherst, MA, February 1994. Appendix accepted and published in Preprints of the Eighth International Software Proces Workshop.

[12] Xiping Song and Leon J. Osterweil. Experience with an approach to comparing software design methodologies. *IEEE Transactions on Software Engineering*, 20(5):364–384, May 1994.

[13] Stanley M. Sutton, Jr. and Leon J. Osterweil. The design of a next-generation process language. In *Proceedings of the Joint 6th European Software Engineering Conference and the 5th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 142–158. Springer-Verlag, 1997.