Rigorous, automated method for artifact-based functional comparison of software processes

Rodion M. Podorozhny
University of Texas
Austin TX 78712
512-232-7931
podorozh@mail.utexas.edu

Dewayne E. Perry
University of Texas
Austin TX 78712
512-471-2050
perry@ece.utexas.edu

Leon J. Osterweil
University of Massachusetts
Amherst MA 01003
413-545-2186
ljo@cs.umass.edu

ABSTRACT
Sound methods of analysis and comparison of software processes [1] are crucial for such tasks as process understanding, process correctness verification, evolution management, process classification, process improvement, choosing the appropriate process for a certain project.

The aim of this work is to lay the foundations for a systematic and rigorous comparison of processes by establishing fixed methods and conceptual frameworks that are able to assure that comparison efforts will yield predictable, reproducible results.

The comparison process presented here assumes that the comparison will be done relative to a fixed standard feature classification schemas for the processes used, and with the use of a fixed formalism for modeling the processes. The aspect of the process described in this paper is focused on functional comparison of processes according to the predefined comparison topics. The process assumes that the compared processes are functionally similar at the top level of abstraction. The suggested comparison process is based on comparison of artifacts [2] produced by the processes when given the same input. This fixed feature classification schema for artifact elements is used as a point of reference for the comparison and it enables one to avoid making conclusions based on interpretation of process activities names.

The paper describes our comparison process and its application comparing two versions of a logistics software process from the telecommunication domain.

[1] A software process is thought of as a way of producing products or delivering services specified in software and requiring human involvement

[2] This paper uses the term "Artifact" to denote a data structure of a software process.

Keywords
Software Process, Comparison

1 Introduction

This work presents a novel approach for comparing software processes that enables one to significantly increase the objectivity and repeatability of comparisons. To our knowledge, this is the first attempt at a partially automated comparison of software processes based on the artifacts they produce. While our work focuses on the application of the comparison process to software process comparison, it is more general. It is also applicable in domains other than software process, such as data-based comparison of software applications, for instance, for evaluation of continuous program optimization techniques ([18]).

It is our belief that certain tasks (e.g. software development) are very unlikely to be completely automated in the foreseeable future if ever. Thus there will be a need for software process systems with human involvement in their execution. We believe that the operation of such systems can be properly described and analyzed with the use of the concept of a software process as introduced in ([19]).

One of the hallmarks of a mature scientific or engineering discipline is its ability to support the comparison and evaluation of the artifacts with which it deals. Systematic comparisons rest upon classification. Thus we believe that the establishment of a discipline of process engineering requires the development of techniques and structures for supporting the classification, comparison, verification, evaluation, and improvement of processes. Systematic, rigorous and automatable comparison techniques can help achieve the goals of process engineering.

In their earlier work on this topic, Xiping Song and Leon Osterweil proposed techniques and structures for a disciplined and rigorous software process comparison, and demonstrated their use by carrying out classifications and comparisons of processes drawn from the narrow and specialized domain of software design processes [20], [21]. These comparisons were guided by a formal comparison process, Comparison of Design Meth-

ods (CDM), and were performed according to a fixed base framework. The base framework can be thought of as a classification schema and provides guidelines for grouping comparable activities, artifacts, and features.

The accuracy of such a comparison is limited by the accuracy of process models. It is very likely that any process comparison method is limited by the accuracy of the models. While the CDM was one of the first steps toward systematic process comparison, it was performed manually without any automation. The classification of activities and artifacts relied on the subjective interpretation by the comparer of the names and model descriptions.

Our work builds on experience with the CDM by using the concept of the base framework, a systematic comparison process, and formal process models. It must be noted that structure, content, and construction method of our BF, comparison process, and process models are different from the CDM. The improvements, over the previous work in this area, we strive to achieve are increased objectivity and greater repeatability of comparison results, more focused comparisons, more general applicability than the domain of software development methods, addressing the ambiguity introduced by human involvement in process execution, and partial automation of the comparison process.

The suggested comparison approach assumes that the two compared entities (software processes or applications) are in the same problem domain and have a similar purpose, but might have certain differences in how they achieve their goals starting from the input of the same kind and providing output of the same kind. The approach is based on the comparison of artifacts produced by the compared processes along the execution paths prompted by the same input (e.g. the same formal requirements for a software system fed to different software development processes). Thus the approach also makes an assumption that the paths traversed through the compared processes in response to the same input are comparable and produce comparable artifacts.

Metaphorically speaking, the approach suggests "coloring" semantically similar portions of the input artifacts of the compared processes or applications in the same "color". The approach also suggests ensuring the continuity of this "coloring" throughout the elements of the artifacts along the executed path in such a way that elements from artifacts of different stages of the process execution that deal with the same concern retain the same "color". Then the compared processes (applications) are instrumented so that activity decomposition units would get "painted" by the artifact concerns[3] after execution. The analysis for functional comparison entails determining the artifact concerns related to specific comparison topics and then identifying the sets of activity decomposition units that got "painted" by those artifact concerns during execution. Thus the comparison approach points out the differences or similarities in the produced artifacts in the portions identified as semantically identical. The approach also suggests sets of comparable process activities for the compared processes and functional differences between them without basing these results on the activities' naming.

In this paper we describe our comparison process and its validation by way of a case study application to an industrial telecommunications logistics process.

In Section 2 we describe the related work. The approach is introduced in Section 4. Section 5 covers our experience with the comparison process applied to a telecommunications logistics process. We conclude with description of future research directions in Section 6.

2 Related Work

The work by Xiping Song and Leon Osterweil was aimed at comparing software processes rigorously modeled in software process languages according to a base framework (BF) specifically defined for the CDM. The decomposition units of activities and artifacts in their method were classified according to the BF by manual comparison of the description of an activity and/or artifact in the process model or the original source with the description of the BF class. The further comparison of the so classified and hence comparable process decomposition units of the two compared processes was also done manually, guided by a comparison process formalized at a rather high abstraction level. The comparison process described in [21] assumes that the process models are at sufficient level of accuracy and elaboration for the complete comparison by artifacts and activities. Our approach also uses the concept of a base framework, though its structure, content, and construction method are different from the one suggested by the CDM process. We chose a flat framework that, initially, only includes classes for atomic elements of artifacts (as per a process source description) from a specific problem domain. The comparison approach provides for process decomposition unit groupings at a later stage. Our comparison process is generalized for use with processes from different problem domains rather than being focused on software design methods as the CDM.

The need to compare modeled processes according to a fixed base framework was also recognized somewhat earlier by Sjaak Brinkkemper et al. ([13]). However, their comparison had no guidelines as explicit and formal as the CDM. The BF suggested in ([13]) has a flat

[3] A meaningful portion of the product artifact whose evolution can be traced from input to output in a sequence of artifact elements

structure as well. The content and construction method are different from the BF in our approach. The BF classes in ([13]) are constructed on the basis of the elements of the process and artifact decomposition units of the compared processes. Instead we treat our BF as a pre-defined ontology for a specific problem domain onto which we initially map only those artifact units that need to be examined for a focused comparison. Thus the BF construction in our approach is incremental and it is guided by the comparison topics instead of an across-the board comprehensive comparison.

Both of these approaches used process models based on the original descriptions of the process authors. Thus their comparison is as accurate as the process models are sufficient in accuracy and elaboration. The comparison is also as complete as the base framework and the process models allow them to be.

Analysis of in-place software processes and measurement of the correspondence of a particular process execution to its model have similar goals with process comparison in that they attempt to evaluate processes. Some fairly recent work in these directions has been done by Jonathan Cook and Alexander Wolf ([7], [8], [9]).

3 Motivating example

As our motivating example we used a telecommunications ordering process employed by Telcordia. The ordering process elaborates the activity of adding a service to a customer. This company uses a proprietary process specification language for rigorous specification of such logistics processes. Their logistics processes also use a predefined set of artifact formats. In addition to the format specification, there is a set of well-formedness conditions defined for the artifacts. One of the challenges the developers of these processes face is the task of change management. After a change is made to the process the developers have to make sure that the process still produces artifacts complying with the well-formedness conditions. If the new version of the process produces an undesirable result then the developers have to find out the cause. This is not always as trivial as it would seem even for relatively small processes as not a single developer understands the process in its entirety. There are also different possible interpretations of the process by developers. The suggested comparison approach alleviates some of these problems by providing a rigorous analysis of process artifacts and suggesting possible causes for the differences based on such an analysis.

The study assumed that two seemingly identical versions of the same process need to be compared to find out if the artifacts produced comply to a certain well-formedness condition, and to point out the reason for the differences if there are any. Such a set-up is likely to highlight the benefits of the artifact-based trace analysis technique that can be used to complement the static analysis of the process template specification such as by Jamieson Cobleigh et al. ([6]).

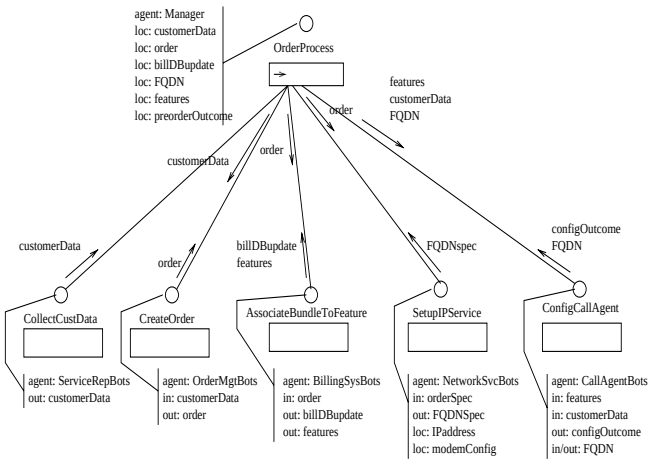

Figure 1: Order process

The representation of the motivating example process template is depicted in Fig. 1. This figure shows a functional decomposition of process steps. The steps' interfaces include specification of an agent class (*agent:* prefix)[4], local parameters (*loc:* prefix), input parameters (*in:* prefix, and output parameters (*out:* prefix). The data flow is depicted along the decomposition links: the inscriptions near the arrow into a step contain input parameters and that near the arrow out of a step contain output parameters. A complete process specification also includes the resource model that specifies the agents available in the environment, the artifacts specification, and the agents' problem solver components specification that define the transformations from input artifact formats to output ones. The process program declares the agent classes for steps. The actual agents are bound to steps during process execution, therefore it is possible to run the same process template in different environments. A more detailed description of the motivating process is given in Section 5.

An example of a well-formedness condition for this telecommunications ordering process is the need to base voice communication service on a data communication service. If the ordering process does not establish that a customer ordering the voice communication also needs the data communication then the process creates malformed artifacts that result in billing the customer for the voice service that will not function. To avoid this scenario the executing software process (including the template and functionality of the agents responsible for

[4] An agent is an entity responsible for execution of a step.

performance of the steps) has to be shown to comply with the well-formedness condition. Any differences and their possible causes must be found, be they in the process template or agent functionality, and must be reported. Our comparison approach suggests a rigorous and automated way to provide these results.

4 Overview of the comparison approach

In this section we will give a brief overview of the approach to artifact-based comparison and definitions of the main concepts and terms.

Definitions

First we introduce the notions and terms that describe the compared entities, aspects of the comparison approach, and some qualitative measures of comparison results.

Software process, artifact, base framework, comparison topic, agent

By the term **software process** we understand a way, specified in software, of producing a product or delivering a service. A software process describes the set of activities, constraints on the order of execution of those activities, and data structures (artifacts) that the activities manipulate. One of the major differences between a software process and a software system is the assumption that some of the software process activities cannot be completely automated and hence have to be performed by human agents. The need for human involvement in process execution is also one of the main reasons why programming languages are not suitable for the specification of software processes. Instead, a number of software process specification languages that take into account the peculiarities of software processes have been developed over the years (Little-JIL [3], [24], APPL/A [22], Oz [2], Weaver [4], APEL [17], SPADE [1], MERLIN [15], ALF [12], DYNAMITE [16]). We assume that the software process to be compared has already been described either in a structural form of English or in a process language. It is also possible to capture a software process specification from its enactment in a certain organization even if the process has not been specified explicitly.

The term **software process artifact** (or **artifact**) is used in this paper to denote some data structure manipulated by the activities of a software process (such as documents, code, test cases etc.).

The **base framework** denotes a problem domain specific framework for software process decomposition units, and features or metrics that can be derived on their basis. The base framework can be thought of as a classification schema that provides guidelines for grouping comparable activities, artifacts, features, or metrics of software processes from the same problem domain. Software processes are likely to be in the same problem domain if their purpose and functionality largely overlap and if their input and output artifacts are largely the same by nature and content.

Our approach focuses on the comparison of explicitly defined aspects of software processes. The software process aspects to be compared are defined by **comparison topics**. These topics relate the vaguely defined high-level topics to formally defined conditions on the content and/or structure of process artifacts. They describe the comparer's assumption that if a certain set of first order logic formulas defined on content and/or structure of artifacts evaluates to be true then it is the case that a certain high-level feature of a software process is present. The definition of comparison topics is one of the decision-making aspects in the comparison approach that allows variance of opinion.

The term **agent** is used in this paper to denote an entity that is

- intelligent in that it can optimize its actions to achieve a goal,
- independent in that it can choose the set of actions to achieve a goal by itself,
- interactive in that it interacts and cooperates with other agents to achieve its goals.

This term is used in conjunction with the kind of software process specification and design of software process execution systems we assume.

Comparison approach

Briefly, the approach is based on:

- the use of the artifact section of base framework common for the compared processes (this results in common naming of semantically similar low level artifact elements)
- the specification of comparison topics as first order logic formulas on the semantically similar low level artifact elements to make sure that the same condition is specified for the compared processes
- the instrumentation of software processes (results in annotated traces of artifacts)
- the execution or simulation of instrumented software processes given the same input
- the derivation of sets of activities/agents from annotations in artifact traces, such that those activities/agents are responsible for compliance or violation of the first order logic formulas that describe comparison topics

The comparison approach is outlined in Fig. 2. The sections that follow refer to Fig. 2 while detailing the comparison approach. We now use a functional notation to express the comparison process of Fig. 2 rigorously. The process consists of the following principal functional transformations, spaces and sets:

1. $\boldsymbol{Process\_Modeling_{MF,AMF}} : \boldsymbol{SP} \rightarrow \boldsymbol{SP\_Model_{MF,AMF}}$,
where $\boldsymbol{SP}$ is a space of all software processes with a similar purpose and $\boldsymbol{SP\_Model_{MF,AMF}}$ is a space of models of SPs in the modeling formalism MF such that the artifact formats of MF are expressed in the artifact meta-format AMF.

2. $\boldsymbol{Artifact\_Modeling_{MF,AMF}} : \boldsymbol{AF^{sp}} \rightarrow \boldsymbol{AF\_Model^{sp}_{MF,AMF}}$,
where AF^{sp} is a space of formats of all artifacts of process $\boldsymbol{sp} \in \boldsymbol{SP}$, and $AF\_Model^{sp}_{AMF}$ is a space of artifact formats $\boldsymbol{AF^{sp}}$ of the process modeling formalism MF such that the formats themselves are expressed in the artifact meta-format AMF. Functional transformation $\boldsymbol{Artifact\_Modeling_{MF,AMF}}$ is a subset of transformation $\boldsymbol{Process\_Modeling_{MF,AMF}}$.

3. $\boldsymbol{Operations} = (\boldsymbol{Create}, \boldsymbol{Derive}, \boldsymbol{Retain}, \boldsymbol{Modify})$

4. $\boldsymbol{Activity\_Instrumentation_{MF,AMF}} : \boldsymbol{Act\_Atoms^{sp}} \rightarrow (\boldsymbol{Act\_Atoms^{sp}} \times \boldsymbol{Operations})$,
where $\boldsymbol{Act\_Atoms^{sp}}$ is a set of atomic activities of process $\boldsymbol{sp}$. $\boldsymbol{Operations}$ is a finite, fixed set of the predefined kinds of operations on artifact elements expressed in artifact meta format $\boldsymbol{AMF}$. Reasoning involved in assigning operation kinds to atomic activities is explained in Section ??. Functional transformation $\boldsymbol{Activity\_Instrumentation_{MF,AMF}}$ is a subset of transformation $\boldsymbol{Process\_Modeling_{MF,AMF}}$.

5. $\boldsymbol{Classify\_Artifacts_{MF,BF}} : \boldsymbol{AF\_Atoms^{sp}} \rightarrow \boldsymbol{AF\_Groupings_{BF,MF}}$,
where $\boldsymbol{AF\_Atoms^{sp}}$ is a space of all artifact format elements at the lowest level of decomposition (atomic elements) from process $\boldsymbol{sp}$ where $\boldsymbol{AF\_Groupings_{BF,MF}}$ is a space of all artifact format atomic elements, structured by BF.

6. $\boldsymbol{Formalize\_Topics_{CT}} : \boldsymbol{CT} \rightarrow \boldsymbol{Features_{AF\_ClAtoms^{sp}}}$,
where $\boldsymbol{CT}$ is a space of comparison topics and $\boldsymbol{Features_{AF\_ClAtoms^{sp}}} = \{\boldsymbol{f_i}\}$ is a set of artifact-based features formalized as first order logic expressions $\boldsymbol{f_i}$ operating on a set of atomic artifact elements $\boldsymbol{AF\_Atoms^{sp}}$, such that $\boldsymbol{AF\_ClAtoms^{sp}} = \{\boldsymbol{el} \in \boldsymbol{AF\_Atoms^{sp}} \mid \boldsymbol{el} \in \boldsymbol{grouping}\ \&$
$\boldsymbol{grouping} \in \boldsymbol{AF\_Groupings_{BF,MF}}\}$. Thus, $\boldsymbol{AF\_ClAtoms^{sp}}$ is a set of atomic artifact elements of process $\boldsymbol{sp}$ classified (grouped) according to the base framework $\boldsymbol{BF}$ and represented in the modeling formalism $\boldsymbol{MF}$. It is also required that the artifact formats of $\boldsymbol{MF}$ are specified in the artifact meta-format AMF common for the models of the compared processes.

7. $\boldsymbol{Execution^{sp\_Model,spInput}} : \boldsymbol{SP\_Model_{MF,AMF}} \rightarrow \{(\boldsymbol{A^{sp\_Model_{MF,AMF},spInput}}, \boldsymbol{Annots_{AF\_Atoms^{sp}}})\}$
,
where $\boldsymbol{A^{sp\_Model_{MF,AMF},SPinput}}$ is a set of artifacts that represents a trace of the execution path through software process model $sp\_Model_{MF,AMF}$ when given a certain input SPinput and $\boldsymbol{Annots_{AF\_Atoms^{sp}}}$ is a set of annotations for atomic artifact elements.

$\boldsymbol{Annots_{AF\_Atoms^{sp}}} = \{\boldsymbol{Annot_i}\}, \boldsymbol{0 \leq i \leq |AF\_Atoms^{sp}|}$, where $\boldsymbol{Annot_i} = \{\boldsymbol{annot_{i,j}}\}$; $\boldsymbol{0 \leq i \leq |AF\_Atoms^{sp}|, 0 \leq j \leq |Act\_Atoms_{sp,atom_i^{AF}}|}$.

$\boldsymbol{Act\_Atoms_{sp,atom_i^{AF}}}$ is a set of atomic activities of process $\boldsymbol{sp}$ that created or modified the atomic artifact element $\boldsymbol{atom_i^{AF}}$ and the atomic artifact elements that address the same low-level concern throughout the process execution (sets of the elements addressing single low-level concerns are implicitly determined by process modeling, subsection on instrumentation expands on this issue below).

$\boldsymbol{annot_{i,j}}$ is a tuple $(\boldsymbol{AF\_atom_i^{sp}}, \boldsymbol{operation_{i,j}}, \boldsymbol{activityID_{i,j}}, \boldsymbol{agentID_{i,j}}, \boldsymbol{timestamp_{i,j}}, \boldsymbol{runID_{i,j}})$,
$\boldsymbol{0 \leq i \leq |AF\_Atoms^{sp}|, 0 \leq j \leq |Act\_Atoms_{sp,atom_i^{AF}}|}$,
where $\boldsymbol{AF\_atom_i^{sp}}$ is an atomic element from artifact formats of process $\boldsymbol{sp}$. $\boldsymbol{operation_{i,j}}$ is the operation performed by agent $\boldsymbol{agentID_{i,j}}$ while executing activity $\boldsymbol{activityID_{i,j}}$ on its $\boldsymbol{runID_{i,j}}$ run on atomic artifact element $\boldsymbol{AF\_atom_i^{sp}}$ at time $\boldsymbol{timestamp_{i,j}}$. Set of annotations $\boldsymbol{Annots_{AF\_atom_i^{sp}}}$ for artifact element $\boldsymbol{AF\_atom_i^{sp}}$ is a set partially ordered by time: $\boldsymbol{timestamp_{i,j} \leq timestamp_{i,j+1}}$.

8. $\boldsymbol{Feature\_Checker_{Features_{AF\_Atoms^{sp}}}} : \boldsymbol{SP\_Model_{MF,AMF}} \rightarrow \boldsymbol{Z^*}$,
where $\boldsymbol{SP\_Model_{MF,AMF}}$ is a space of models of SPs in the modeling formalism MF and $\boldsymbol{Z^*}$ is the set of non-negative integers.

9. $SP\_Comparison_{sp1,sp2,BF}$ is a set of tuples $(feature_i, A_{sp1\_Model_{MF,AMF,feature_i}, spInput}, A_{sp2\_Model_{MF,AMF,feature_i}, spInput})$, $0 \leq i < |Features_{AF\_ClAtoms_{sp1} \cap AF\_ClAtoms_{sp2}}|$, where $A_{sp1\_Model_{MF,AMF,feature_i}}$ is a set of artifacts produced by software process $sp1$ on input $spInput$ such that they have to be examined to establish if $sp1$ complies with conditions for having $feature_i$. $A_{sp2\_Model_{MF,AMF,feature_i}}$ is defined in the same way, but for process $sp2$.

10. $Feature\_Comparator_{Features_{AF\_ClAtoms_{sp1} \cap AF\_ClAtoms_{sp2}}, BF} : A_{sp1\_Model_{MF,AMF}, spInput} \times A_{sp2\_Model_{MF,AMF}, spInput} \rightarrow SP\_Comparisons_{sp1,sp2,BF}$, where $SP\_Comparisons_{sp1,sp2,BF}$ is a space of comparisons of features identified by BF between processes $sp1$ and $sp2$.

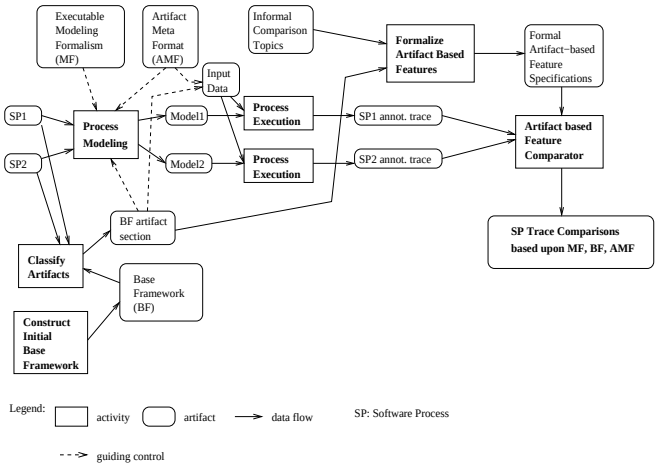

Figure 2: Comparison Approach Overview

The first activity of the comparison approach is the definition of the artifact section of the BF and the classification of the lowest-level artifact elements [5] of the compared process by it. This comparison activity is difficult to automate: it requires substantial human involvement. It is one of the major decision-making points of the approach that allows for variance of opinion. A reasonably good understanding of the domain is likely to make this variance manageable. The rest of the comparison approach bootstraps from the definition of the artifact section. The decisions made in this activity influence the comparison results as much as the comparison topics formalization. While this activity is not automated, the comparison approach specifies a systematic way to classify the artifacts. This activity is depicted in Fig. 2 as **Construct Initial Base Framework**.

The artifacts of the compared processes have to be defined using a meta-format common for the artifacts of the compared processes. This activity corresponds to the functional transformation 2, $Artifact\_Modeling_{MF,AMF}$. It is part of the functional transformation 1, $Process\_Modeling_{MF,AMF}$, and it is depicted in Fig. 2 as **Process modeling**. The choice of the meta-format depends on the problem domain of the compared processes.

The comparison approach uses process model instrumentation for gathering data about the manipulations of artifacts. Most of the comparison results are based on the way the artifact elements are manipulated. Hence it is important to properly model and elaborate the activities that perform these manipulations. We assume that an activity is atomic if it performs a single operation to produce an atomic artifact element. A more specific definition of an atomic activity depends on the chosen Artifact Meta Format. The specification of atomic activities corresponds to the functional transformation1, $Process\_Modeling_{MF,AMF}$, and it is reflected in Fig. 2 as **Process Modeling**. The instrumentation part of $Process\_Modeling_{MF,AMF}$ corresponds to the functional transformation 4, $Activity\_Instrumentation_{MF,AMF}$. The intention of the comparison approach is to base instrumentation decisions on the basic manipulations possible on artifact elements expressed in the chosen Artifact Meta Format.

Having defined the artifact section of the base framework common for the compared processes as a point of reference, the comparer should express comparison topics as first order logic formulas on the artifact elements that map to the common artifact section of BF. This activity corresponds to the functional transformation 6, $Formalize\_Topics_{CT}$, and it is shown in Fig. 2 as **Formalize Artifact Based Features**. The term **feature** is used in the figure to denote a formalized comparison topic which is a set of first order logic formulas defined on the artifact elements that evaluates to true if the process possesses a specified property.

The comparison activity of execution of compared processes corresponds to the functional transformation 7, $Execution^{sp\_Model,spInput}$, and it is shown in Fig. 2 as **Process Execution**.

Once the traces of artifacts have been produced, they are analyzed to find out the sets of leaf activities that are most likely responsible for complying to or violating conditions for compared features. The basic activity of such an analysis is checking a set of artifacts

[5] Meaning the artifact elements at the lowest level of decomposition as specified in a given process model.

against formalized comparison topics. This comparison activity corresponds to the functional transformation 10, ***Feature_Comparator***, and it is shown in Fig. 2 as **Artifact Based Feature Comparator**.

5 Case study with a telecommunications process

Base framework artifact section

Since this study used the change management task the compared processes use the same artifact format for artifact representation. Therefore the common artifact portion of the base framework consists of the elements of the artifact formats defined by the process. Thus the functional transformation 5, $\boldsymbol{Classify\_Artifacts_{MF,BF}}$, is foregone in this case.

Process modeling

This section describes modeling of the telecommunications process from the original description that was obtained from Telcordia Research representatives (formerly at MCC) via interviews and inspection of the original process specification in the company's proprietary process language, the Collaboration Management Infrastructure (CMI). The telecommunications process used in the study is proprietary and its detailed description has not appeared in open publications at the time of this writing. Some information about the CMI and experience of its application can be obtained from [14], [10], [11].

The process modeling activity described below corresponds to the functional transformation 1, $\boldsymbol{Process\_Modeling_{MF,AMF}}$, introduced in Section 4 where the MF modeling formalism used is the Little-JIL ([24]), AMF (Artifact Meta-Format) is a graph-based meta-format.

The versions differ in an execution of one step so the description is largely the same for them both. The process top step *OrderProcess* specifies sequential execution of its substeps. It also requires an agent of the Manager resource class and declares a number of local artifacts. The process is started when a customer connects a new modem to the cable which triggers collection of information about the customer (*CollectCustData* step). The Little-JIL implementation uses an agent of resource class ServiceRepBots for execution of this step that creates a new customer description artifact (*customerData*) without the actual information collection. The output artifact, *customerData*, contains information about the customer's name, address, and request for service configuration.

Next, the *CreateOrder* step is executed by an agent of the OrderMgtBots resource class. The input to this step is the *customerData* artifact; the *order* artifact is the output. The step's agent transforms the format of *customerData* artifact into that of the *order* artifact by adding the CustomerOrderClass artifact element and its instances. The CustomerOrderClass artifact element contains more detailed technical information about the service request.

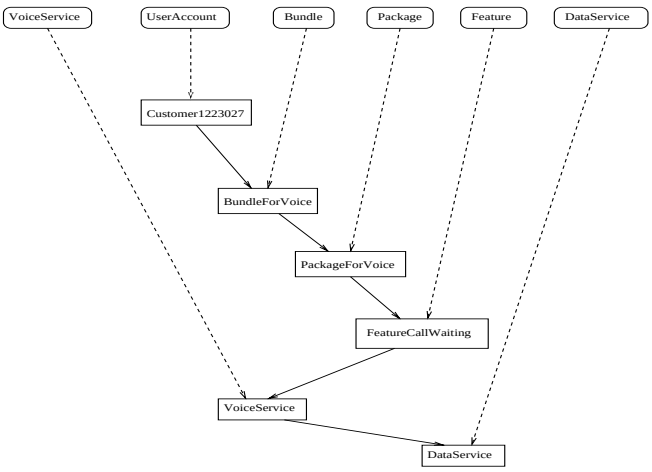

Figure 3: billDBupdate artifact instance

The next step to be performed, *AssociateBundleToFeature*, suggests the new service configuration for the customer based on the service request. It is the execution of this step that makes the difference between the two process versions. The *order* artifact containing the service request forms input to the step. The output consists of the *billDBupdate* (an example is in Fig. 3) and *features* artifacts. The *features* artifact is auxiliary and it is a subset of the *billDBupdate* artifact. This step is performed by an agent of the BillingSysBots resource class. The configuration contained in *billDBupdate* specifies aggregations of low-level services and dependencies between them at different levels. Such services as a data connection and voice connection form the lowest level. The next level up includes entities representing aspects of the services, these aspects are called *features* in the telecommunication logistics process terminology. For instance, call waiting is a feature of a voice connection. While the features might be thought of as attributes of services, they are represented as individual entities that build up on the services. Features are aggregated into packages. A package is aggregated by technical considerations of the service implementation. The packages are further aggregated into bundles that take into account business considerations for aggregation. A bundle is then attached to a user account. The telecommunication logistics process terminology does not differentiate between the kinds of relationships such as aggregation or dependence in the service configuration. The original artifact format specifications refer to all of them as relationships. Thus they are modeled with the same relationship kind.

The *SetIPService* step is executed next. Even though there is no data-dependence between the *SetIPService* and *AssociateBundleToFeature* steps the original process specification requires sequential execution. This step is performed by an agent of the NetworkSvcBots resource class. The input consists of the *order* artifact, while the output is the *FQDNSpec* artifact (Fully Qualified Domain Name specification).

The *ConfigCallAgent* step configures a "call agent" and confirms that the requested service has been properly set up. The "call agent" in the step's name is a telecommunication logistics process term that has nothing to do with the Little-JIL agents. This step is executed by an agent of the CallAgentBots resource class. Its input consists of the *features*, *customerData*, and *FQDN* artifacts. Its output consists of the *configOutcome* (configuration outcome) artifact that signals either success or failure of the service set-up.

The final process model has been presented to the authors of the original process specifications and it was deemed accurate for the purpose of the comparison.

Process simulation
The ordering processes specified in the Little-JIL have been run in the Juliette environment using the same resource model and input. The process execution produced two traces of artifacts. An entry corresponds to an artifact instance name which consists of a timestamp, the name of the agent that processed or produced the artifact, the name of the step the agent performed, the name of the artifact, and a run ID. This activity corresponds to the functional transformation 7, $\boldsymbol{Execution}^{\boldsymbol{sp\_Model,spInput}}$.

Comparison topic and its formalization
The well-formedness condition to be checked is the requirement for a voice service to rely on an existing data service in the customer's service configuration. This requirement is reflected in a relationship from the voice service to the data service in the *billDPupdate* artifact. One version of the process checks for the data service and establishes the necessary relation. The other version omits this action and produced a malformed artifact which would lead to a failure of the service request set-up in a deployed telecommunications process. The comparison process determines this difference and points to the step/agent combinations that are responsible for it. This activity corresponds to the functional transformation 6, $\boldsymbol{Formalize\_Topics}_{\boldsymbol{CT}}$.

This comparison topic is formalized as a first order logic rule in the Xlinkit rule specification language ([5]). The formalized comparison topic is phrased as $\forall \boldsymbol{vs} \in \boldsymbol{voiceservices}\ \exists \boldsymbol{link} \in \boldsymbol{associations}$ s.t. $\boldsymbol{link.source} = \boldsymbol{vs} \vee \boldsymbol{link.destination} = \boldsymbol{ds}, \boldsymbol{ds} \in \boldsymbol{dataservices}$. The Xlinkit rule specification of the comparison topic is shown in Fig. 4.

```
...
<consistencyrule id="wellform1">
  <header>
    <description>
      Voice service should be associated to data service
    </description>
  </header>
  <forall var="vs" in="$voiceservices">
      <exists var="l" in="$associations">
             <and>
                <equal op1="$vs/@name"
 op2="$l/@source"/>
 <exists var="ds" in="$dataservices">
     <equal op1="$ds/@name"
      op2="$l/@dest"/>
                 </exists>
     </and>
      </exists>
  </forall>
</consistencyrule>
</consistencyruleset>
```

Figure 4: Comparison topic example

Process comparison
The comparison process next ran the Xlinkit consistency checker on the artifacts from the traces of the two processes using the rule specification in Fig. 4. This activity corresponds to the functional transformation 8, $\boldsymbol{Feature\_Checker}_{\boldsymbol{Features}_{\boldsymbol{AF\_Atoms}^{\boldsymbol{sp}}}}$. The output of this comparison process step is a set of consistency links between the rule and the artifact elements from the artifact traces. The checker produced a consistent link from the rule to the *VoiceService* artifact element in the artifact *billDBupdate* of the first process version (Fig. 3). The checker produced an inconsistent link from the rule to the *VoiceService* artifact element in the artifact *billDBupdate* of the second process version.

Finally, the comparison process performed the functional transformation 10, $\boldsymbol{Feature\_Comparator}$. The annotation lists for the *VoiceService* elements from the artifacts *billDBupdate* showed that both artifact elements were produced by the same step (*AssociateBundleToFeature*), but by different agents. The comparison process thus showed that the second version of the process does not comply with the well-formedness condition and the reason for that is a different agent for the execution of the *AssociateBundleToFeature* step.

6 Conclusions and future work
The experiment with the ordering process showed that the comparison approach is viable for comparison of process executions differing in interpretation of the same steps. This result would be impossible to obtain using static analysis of process template specification. The approach is useful for comparison of processes dealing with transformation of artifacts of predefined formats. In case of a change management task we did not have a great variance of opinion about correspondence of ar-

tifact elements between the two processes. Therefore identification of common artifact portion of the base framework was trivial. We expect it to be the case in the area of logistics processes because in such processes the set of predefined artifact formats does not vary nearly as often as the processes that manipulate them. The decision-making that allows for variance of opinion is confined by the comparison process largely to the choice of the common artifact portion of the base framework. Such a focussed localization of decision-making enables higher repeatability of the comparison process results done by different comparers once they decide on a reasonably good understanding of the artifacts from the problem domain of the compared processes.

The comparison results seem to be more useful if the compared processes are elaborated to the level of explicit specification of manipulation of the lowest level artifact elements. For instance, for useful comparison, in case of artifacts represented as graphs, the compared processes would have to explicitly specify the transformations of nodes, links, and their attributes between artifacts from different stages of the process.

In summary, 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.

There are several directions of further work worth pursuing, particularly that of experimenting with an approach in non-software process domains. One of such experimentation directions is the application of the comparison process to the evaluation of path specific optimization techniques for user-guided applications (such as GUIs). The authors of continuous optimization techniques ([18]) assume that a user is likely to make the application traverse the same execution path when the same user accomplishes the same kind of task. Hence the optimizer takes the traversed path into account when scheduling the instructions. Our approach can help in data-based evaluation of the optimizers by keeping track of instructions that were applied to elements of application's data structures. Thus it would be possible to show different emphasis the optimized execution paths put on application's data elements and explain possible variance in performance of the optimizers. These comparison results might be used by AI techniques for choosing the appropriate path-specific optimization technique during an application's execution.

Another experimentation direction is application of the comparison to more extensive processes.

The comparison process itself can be evolved. One evolution is a more generalized set of possible kinds of artifact elements transformations based on the nature of artifacts. Once such a set is identified for a certain problem domain, the agents for process steps of this domain could be assembled from such a basic set of manipulations. Identification of such a set would also help an automated elaboration of the compared processes to the same level of specificity which will further aid the comparison by parceling out the comparable and functionally analogous portions of the steps from the compared processes.

REFERENCES

[1] Sergio Bandinelli, Alfonso Fuggetta, Carlo Ghezzi, and Luigi Lavazza. SPADE: An Environment for Software Process Analysis, Design, and Enactment. In Anthony Finkelstein, Jeff Kramer, and Bashar Nuseibeh, editors, *Software Process Modelling and Technology*, chapter 9, pages 223–248. Research Studies Press, Ltd., Taunton, Somerset, England, 1994.

[2] I. Ben-Shaul. Oz: A decentralized process centered environment, 1994.

[3] Aaron G. Cass, Barbara Staudt Lerner, Eric K. McCall, Leon J. Osterweil, Stanley M. Sutton, Jr., and Alexander Wise. Little-JIL/Juliette: A Process Definition Language and Interpreter. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000), Limerick, Ireland*, pages 754–757, June 2000.

[4] C.Fernström. PROCESS WEAVER: Adding process support to UNIX. In *The Second International Conference on the Software Process*, pages 12–26, 1993.

[5] C.Nentwich, L.Capra, W.Emmerich, and A.Finkelstein. xlinkit: a consistency checking and smart link generation service. In *ACM Transactions on Internet Technology, 2(2)*, pages 151–185, May 2002.

[6] Jamieson M. Cobleigh, Lori A. Clarke, and Leon J. Osterweil. Verifying Properties of Process Definitions. In *Proceedings of the ACM Sigsoft 2000 International Symposium on Software Testing and Analysis (ISSTA 2000)*, pages 96–101. Portland, OR, August 2000.

[7] Jonathan E. Cook, Lawrence G. Votta, and Alexander L. Wolf. Cost-Effective Analysis of In-Place Software Processes. *IEEE Transactions on Software Engineering*, SE-24(8):650–663, August 1998.

[8] Jonathan E. Cook and Alexander L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, July 1998.

[9] Jonathan E. Cook and Alexander L. Wolf. Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. *ACM Transactions on Software Engineering and Methodology*, 8(2):147–176, April 1999.

[10] D.Baker, A.R.Cassandra, H.Schuster, D.Georgakopoulos, and A.Cichocki. Providing customized process and situation awareness in the collaboration management infrastructure. In *Proceedings of the 4th IFCIS Conference on Cooperative Information Systems (CoopIS'99)*, 1999.

[11] D.Georgakopoulos, H.Schuster, A.Cichocki, and D.Baker. Collaboration management infrastructure in crisis response situations. Technical report cmi-009-99, Microelectronics and Computer Technology Corporation, 1999.

[12] G.Canals, N.Boudjlida, J.-C.Derniame, C.Godart, and J.Lonchamp. ALF: A framework for building process-centered software engineering environments. In Anthony Finkelstein, Jeff Kramer, and Bashar Nuseibeh, editors, *Software Process Modelling and Technology*, pages 153–185. Research Studies Press, Ltd., Taunton, Somerset, England, 1994.

[13] 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.

[14] Dimitrios Georgakopoulos, Hans Schuster, Donald Baker, and Andrzej Cichocki. Managing escalation of collaboration processes in crisis mitigation situations. In *Proceedings of the 16th International Conference on Data Engineering (ICDE'2000)*, 2000.

[15] G.Junkermann, B.Peuschel, W.Schäfer, and S.Wolf. MERLIN: Supporting cooperation in software development through a knowledge-based environment. In Anthony Finkelstein, Jeff Kramer, and Bashar Nuseibeh, editors, *Software Process Modelling and Technology*, pages 103–129. Research Studies Press, Ltd., Taunton, Somerset, England, 1994.

[16] Peter Heimann, Gregor Joeris, Carl-Arndt Krapp, and Bernhard Westfechtel. DYNAMITE: Dynamic task nets for software process management. In *Proceedings of the 18th International Conference on Software Engineering*, pages 331–341. IEEE Computer Society Press, 1996.

[17] J.Estublier, S.Dami, and A.Amiour. APEL: A graphical yet executable formalism for process modelling. In *Automated Software Engineering*, March 1997.

[18] Thomas Kistler and Michael Franz. Continuous Program Optimization: Design and Evaluation. *IEEE Transactions on Computers*, 50(6):549–566, June 2001.

[19] Leon J. Osterweil. Software Processes are Software Too. In *Proceedings of the Ninth International Conference of Software Engineering*, pages 2–13, Monterey CA, March 1987.

[20] 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.

[21] 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.

[22] Stanley M. Sutton, Jr., Dennis Heimbigner, and Leon J. Osterweil. APPL/A: A Language for Software-Process Programming. *ACM Transactions on Software Engineering and Methodology*, 4(3):221–286, July 1995.

[23] Frank Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3:121–189, March 1995.

[24] A. Wise. Little-JIL 1.0 Language Report. Technical report 98-24, Department of Computer Science, University of Massachusetts at Amherst, 1998.