Automatically analyzing software processes: Experience report

Rodion M. Podorozhny
UT Advanced Research In Software Engineering (UT ARISE)
The University of Texas, Austin
podorozh@mail.utexas.edu

Dewayne E. Perry
UT Advanced Research In Software Engineering (UT ARISE)
The University of Texas, Austin
perry@ece.utexas.edu

Leon J. Osterweil
Laboratory for Advanced Software Engineering (LASER)
The University of Massachusetts, Amherst
ljo@cs.umass.edu

Abstract

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

The purpose of our research 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 analysis framework presented here assumes that the comparison will be done relative to a fixed standard feature classification schema for the processes used, and with the use of a fixed formalism for modeling the processes. The aspect of the system described in this paper is focused on functional analysis of processes according to the predefined comparison topics, well formedness constraints, and instrumented agents.

The paper describes our experience using our analysis system and its application to a logistics software process from the telecommunication domain.

1 Introduction

This work presents a novel approach for analyzing and 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 analysis and comparison of software processes based on the artifacts they produce. While our work focuses on the application of our analysis system to software process analyses and comparisons, it is more general. It is also applicable in domains other than software process, such as data-based comparison of software applications for evaluation of continuous program optimization techniques ([10]).

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 [12].

One of the hallmarks of a mature scientific or engineering discipline is its ability to support the analysis, comparison and evaluation of the artifacts with which it deals. Systematic analyses and 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 analysis techniques can help achieve the goals of process engineering.

The analysis system discussed here assumes that the analyzed and compared processes 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. Our approach is based on the analysis and comparison of artifacts produced by the processes along the execution paths prompted by similar input (e.g. similar formal requirements for a software system fed to different software development processes). Thus our approach also makes an assumption that the intent of the analyzed processes is in response to the similar input is comparable and produces comparable artifacts.

In this paper we describe our experience analyzing and comparing two versions of a telecommunications logistics process, what results we get from the process analysis, how our system compares to other approaches, and lessons learned from the experience.

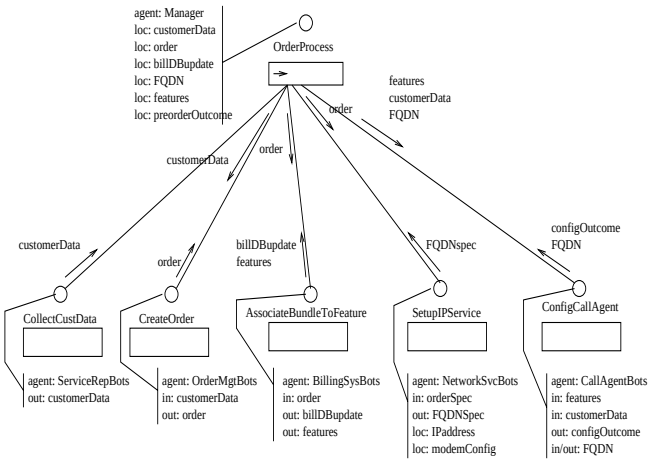

Figure 1. Order process

2 Logistics process example

As our 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. ([5]).

The representation of the motivating example process template is depicted in Fig. 1. We use the Little-JIL process language ([3]) to show software processes in this paper. The visual representation of the Little-JIL is based on a functional decomposition. The steps are depicted as rectangles with a step's name above the rectangle. The steps' interfaces include specification of an agent class (*agent:* prefix)[1], 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.

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 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.

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

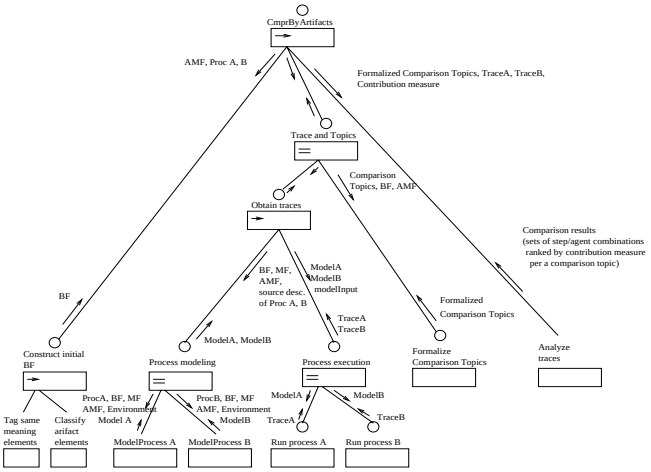

Figure 2. Steps in analysis and comparison

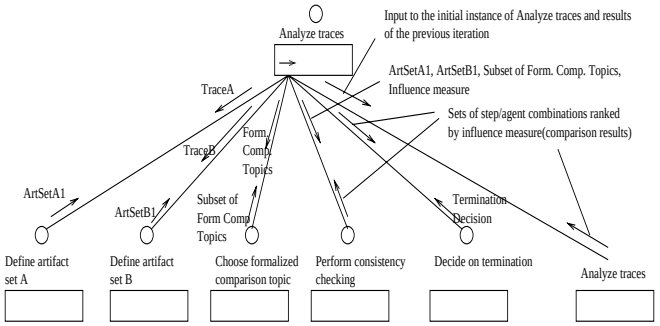

Figure 3. Analyze traces

3 Steps in analyzing and comparing processes

In this section we discuss the use of an analysis system to analyze and compare the example processes. Figures 2 and 3 illustrate the steps required by the analysis system illustrated in the Little-JIL.

The current implementation of the process focuses on an artifact-based analysis and comparison of two software processes. The software processes are assumed to have structured artifacts with predefined formats such that the processes specify transformations between the artifact formats.

The process of analysis and comparison shown in Fig. 2 is automated by a toolset. The steps for process execution and trace analysis are completely automated. The rest of the steps such as creation of the base framework, process modeling, comparison topics specification have to be executed by a user in a systematic way by following guidelines. The toolset assists the user in executing the non-automated steps. For instance, it provides the Artifact Meta-Format for artifact base framework specification and an agent framework for process specification. The non-automated steps are provided with guidelines for a systematic manual execution.

3.1 Artifact ontology specification

The analysis process starts with constructing an initial base framework (BF) for the artifact section (step **Construct initial BF**). The **base framework** denotes a problem domain specific framework for artifacts, software process decomposition units, and process features that can be derived on their basis. The base framework can be thought of as a classification schema or ontology that provides guide-

```
<Node>
 <MetaComponentClass>
   <Attribute attrClass="java.lang.String"
name="name"
value="customerData"/>
   <Attribute attrClass="java.util.Hashtable"
name="children"
value="Customer1223027"/>
   <Attribute attrClass="java.lang.String"
                name="customerPhoneNumber"
value="000-000-00-00"/>
   <Attribute attrClass="java.lang.String"
name="defaultName"
value="defaultValue"/>
   <Attribute attrClass="java.lang.String"
                name="customerStreetAddress"
value=""/>
   <Attribute attrClass="java.lang.String"
                name="customerZipCode"
value="11111"/>
 </MetaComponentClass>
</Node>

<Node>
 <MetaLinkClass>
    <Attribute attrClass="java.lang.String"
name="name"
value="association"/>
    <Attribute attrClass="java.util.Hashtable"
name="children"
value="Customer1223027RequestsServiceReq8745"/>
    <Attribute attrClass="java.lang.String"
name="type"
value="association"/>
  </MetaLinkClass>
</Node>
```

Figure 4. Example of BF specification

lines for grouping comparable activities, artifacts, or features of software processes from the same problem domain. Software processes are likely to be in the same problem domain if their purpose and functionality overlap.

The step is decomposed into the **Tag same meaning elements** and **Classify artifact elements** substeps to be executed sequentially. This step has a substantial subjective involvement of a human user. The BF can be constructed either from an existing ontology or it is generalized from the artifact formats of the analyzed processes. The goal is to identify the semantically overlapping portions of the artifact formats and tag the semantically similar elements of those formats. This is done in the **Tag same meaning elements** substep based on the source descriptions of the processes **Proc A** and **Proc B**. The output of this substep is a table of correspondence of artifact elements from the original process descriptions and their common naming. The correspondence is needed only between artifact elements in the overlapping portion of the semantics of artifacts. Such an overlapping is likely to exist in processes from the same problem domain and with the same purpose.

In the case of our analysis system we used a common artifact meta-format (**AMF**) and the artifact element naming conventions for tagging. Thus artifact elements are classified according to the AMF (step **Classify artifact elements**) and artifact elements with the same meaning are named the same in the process models and artifacts of the same class. The **Tag same meaning elements** substep precedes the **Classify artifact elements** since it is beneficial to reduce the number of elements to be classified. This reduction is the result of giving the same names to the elements with the same meaning, so the classification decision is made only once for both same named elements from different processes. In our example the BF corresponds to the formats of artifacts used by the telecommunications process. The process's authors at Telcordia have already specified the artifact formats rigorously. Since the two analyzed processes use the same artifact formats the task of identifying common ontology (BF) is simplified. The categories of the artifact elements map directly to the categories of the ontology. To obtain the BF specification in our example we wrote every artifact template from Telcordia's source process specification in the AMF. Thus we obtained BF specification for all categories in artifacts used by both analyzed versions of the process: *customerData, order, billDBupdate, FQDN, FQDNSpec, features, modemConfig, IPaddress, preorderOutcome, configOutcome*. An example of BF specification is shown in Fig. 4. This figure shows specification of BF artifact categories *customerData* and *association*. The category specifications also indicate their properties. The actual artifacts used by pre-ordering processes would contain elements that map to these categories and that might be considered their instances. A user would specify the BF categories manually using the Artifact Meta Format to describe the artifact BF categories found in the original description of the analyzed or compared process.

3.2 Process modeling

Once the artifact section of the BF is defined, the modeling of the processes in the same executable process modeling formalism can proceed. The input to this step includes the base framework (**BF**), process modeling formalism (**MF**), artifact meta-format (**AMF**), and the source description of the analyzed processes **Proc A** and **Proc B**. It is preferable to feed rigorous specifications of processes elaborated to the level of manipulation of the lowest level decomposition units of artifacts.

This step is further decomposed into modeling of the individual processes that can proceed in parallel. This step may require substantial human involvement but can be automated in the case if the source descriptions are rigorously defined by building a translator from the formalism used in the source descriptions to the common formalism used for analysis.

The expressiveness of the process formalisms can influence the analysis results if they do not allow modeling of the artifact elements or steps that manipulate them related to the comparison topics. The output of the modeling step consists of the process models in the common modeling formalism (**ModelA, ModelB**). In our implementation we use the Little-JIL as the common modeling formalism for process analysis and comparison. Thus the modeling involves representation of the functional decomposition of the process, specification of the process step interfaces, specification of the artifact formats in the AMF, specification and development of the agents to execute the steps, instrumentation of the agents per a step kind, specification and development of the step-specific GUIs, and the definition of the environment to be the same for both processes (the developed agents are included into the environment).

The original Little-JIL has been extended to generalize the agent and instrumentation specification for individual problem domains. The user must take care not to overspecify the agents beyond the elaboration of the lowest level activities from the source processes. If the source processes assume certain common low level activities then it is advisable to use the same implementation for the agents from both processes. The extended Little-JIL agent architecture allows for reuse of agents' problem solving components. The Little-JIL artifact specification and the agents must use the artifact formats specified in AMF and complying with the naming conventions for the artifact elements with overlapping semantics.

In our implementation of the analysis system the user would specify the process template in the Little-JIL using

the visual editor. An example of a process template we created is shown in Fig. 1. We created two process templates for the analyzed processes.

The user would also specify the agents for the process template using Java and the domain specific agent framework. The framework allows specification of low level agent actions (operations) and then specifying the sets of actions that agents must execute in response to incoming events. In our case the vast majority of events processed by agents are generated by the Little-JIL interpreter. These events carry information about assignment of certain tasks to agents. A task corresponds to an instantiation of process steps. Any task assigned to an agent goes onto that agent's agenda list. The agent framework simplifies the specification of agents by providing a uniform way to specify actions and by providing a generalized way to instrument the process. Every time a certain agent executes an action the information about the action's result is written to the artifact trace. The user only has to specify an action without explicit specification of the instrumentation code.

The analysis system is limited by the level of elaboration of the source processes. If the source process does not describe the activities at the level of manipulation of artifact elements then this method is unlikely to be applicable. The generalized instrumentation components simplify the user's task in the process modeling stage. Nevertheless, the user must make subjective decisions regarding continuity of the artifact concerns. The user must decide on the kind of operation a given agent performs on a given artifact element when performing a certain step ($Operations = (Create, Derive, Retain, Modify)$). Thus every agent, when executing, would add an entry to the annotation lists of the output artifact elements explaining the operation it performed on that element and noting agent and step IDs and the timestamp. Also, the user must decide which output artifact elements are going to inherit the annotation lists from the input artifact elements. It is this decision that ensures the continuity of artifact concern traces. It is likely that specifics of a given problem domain might simplify this task. For instance, in logistics processes there is often a limited, predefined set of artifact formats with predefined and explicit relationships between elements from artifacts of different stages of a process.

Actions comprise the problem solving component of an agent. Part of the problem solver for the NetworkSvcBots agent is shown in Fig. 5 as an example. In this figure the *started* method is invoked in response to an event signifying the start of a certain task assigned to an agent. If the task's name is *ResetModemStepName* then the agent will perform the *GetFQDN* action among others. The example shows the generalization of action specification. An action is instantiated and then the action is executed when it is passed the input artifacts in a graph-based Artifact Meta-Format (implemented as ArchGraph). Having a set of domain specific actions it is fairly easy to create agents using this framework. The user would create or reuse a set of actions specific to the problem domain of analyzed processes so that to specify agents. Thus agents for the two versions of the pre-ordering process reused a number of actions.

```
...
public synchronized void
started(AgendaItemEvent evt) {
AgendaItem item = evt.getAgendaItem();
...
  if (itemName.equals(ResetModemStepName)) {
...
GetFQDN getFQDN = new GetFQDN();
ArchGraph[] args = {modemConfig};
ArchGraph fqdn = getFQDN.execute(args, agentStepID);
item.complete();
...
  }
}
```

Figure 5. Example of specification of NetworkSvcBots agent's problem solver

First we wrote a set of actions in Java for the agents of the analyzed processes. The actions used the artifact categories specified in the AMF to represent manipulation of artifacts. For instance, the *getFQDN* action manipulates the artifact BF category *FQDN*. Then we wrote the automated agents that used the actions. Our analysis system also allows for specification of human-assisting and human-modeling agents by providing a framework for step-specific GUI specification. For instance, the agents we specified for the analyzed process in Fig. 1 are *ServiceRepBots, OrderMgtBots, BillingSysBots, NetworkSvcBots, CallAgentBots*.

3.3 Process execution

The next step of the analysis system, **Process execution**, requires execution of the so modeled and instrumented processes (**ModelA, ModelB**) on the same input (**modelInput**). The result of such an execution is a set of two traces of artifacts whose elements are annotated with a list of operations, agents, and steps that were performed on them. The annotation lists in an artifact would cover the trace until this artifact is produced. Thus product artifacts would contain the most comprehensive annotation lists. The annotations of artifact elements are partially ordered by timestamps by construction via the instrumentation code that is run during the process execution. Thus every artifact element relevant to the comparison topic[2] must have a history of all manipulations done to it in the annotation list. This step outputs the traces of artifacts with annotation lists (**TraceA, TraceB**). The traces follow the execution paths through the process

[2]the one that needs to be checked in order to determine if an artifact complies with a certain comparison topic

```
...
<Node>
  <MetaComponentInstance>
    <Attribute attrClass="graph.model.ComponentClass"
       name="class" value="customerData"/>
    <Attribute attrClass="java.lang.String"
       name="name" value="Customer1223027"/>
    <Attribute attrClass="java.lang.String"
       name="customerPhoneNumber"
       value="617-234-92-32"/>
    <Attribute attrClass="java.lang.String"
       name="customerName" value="Edward Jackson"/>
    <Attribute attrClass="java.lang.String"
       name="customerStreetAddress"
       value="962 Hill Dr."/>
    <Attribute attrClass="java.lang.String"
       name="customerZipCode" value="01403"/>
  </MetaComponentInstance>
</Node>

<Node>
  <MetaLinkInstance>
    <Attribute attrClass="java.lang.String"
      name="class" value="association"/>
    <Attribute attrClass=
       "graph.model.ComponentInstance"
      name="source" value="Customer1223027"/>
    <Attribute attrClass=
        "graph.model.ComponentInstance"
      name="dest" value="ServiceReq8745"/>
    </MetaLinkInstance>
</Node>
...
```

Figure 6. customerData artifact in graph-based AMF

models **ModelA, ModelB** that correspond to the same input **modelInput** and hence are considered comparable.

The user obtains the artifact traces automatically by starting the Little-JIL environment and running a process specification with the environment containing the domain specific agents. Since the artifact ontology and consequently the artifact formats used by the agents of the processes are the same then it is possible to conduct a meaningful analysis and comparison of the artifacts. In our example we ran the analyzed processes and obtained two traces of annotated artifacts specified in the AMF. Unlike the BF specification in Fig. 4 the artifacts contain the actual elements corresponding to the BF categories. An example of *customerData* artifact specification is shown in Fig. 6. It was produced automatically by running the process.

3.4 Comparison topic specification

The step for definition and formalization of comparison topics (**Formalize Comparison Topics** step) can be executed after the initial BF is constructed and in parallel with the **Process modeling** and **Process execution**. This is reflected by auxiliary decomposition steps **Trace and Topics** and **Obtain traces**. This step implies specification of comparison topics in terms of first order logic formulas operating on the artifact elements with common naming conventions. This step outputs **Formalized Comparison Topics** as a set of first order logic formulas. In the case of our example the comparison topic is whether both processes fulfill the requirement that a voice service must 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 produces a malformed artifact which would lead to a failure of the service request set-up in a deployed telecommunications process. This comparison topic is formalized as a first order logic rule in the Xlinkit rule specification language ([4]). The formalized comparison topic is phrased as $\forall vs \in voiceservices$ $\exists link \in associations$ s.t. $link.source = vs \wedge link.destination = ds, ds \in dataservices$. The Xlinkit rule specification we wrote for the comparison topic in our example is shown in Fig. 7.

```
...
<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 7. Comparison topic example

3.5 Artifact trace analysis

Next, the analysis process calls for analysis of artifact traces **TraceA, TraceB** by way of consistency checking to the formalized comparison topics. This analysis is done in the **Analyze traces** step. The step's input consists of **Formalized Comparison Topics** and annotated artifact traces **TraceA, TraceB**.

The step's output forms the results of the artifact-based comparison - consistency links between formalized comparison topics and process artifact elements and sets of step/agent combinations ranked by the contribution measure per a comparison topic. One of the main outcomes of such an analysis is comparison of consistency links from the same comparison topic to artifact elements in different

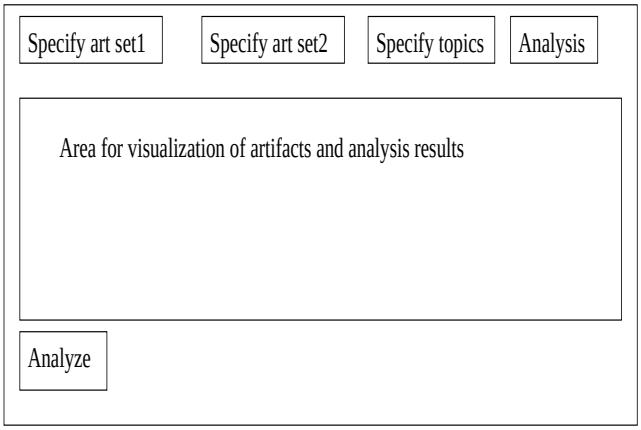

Figure 8. Process analysis toolset GUI

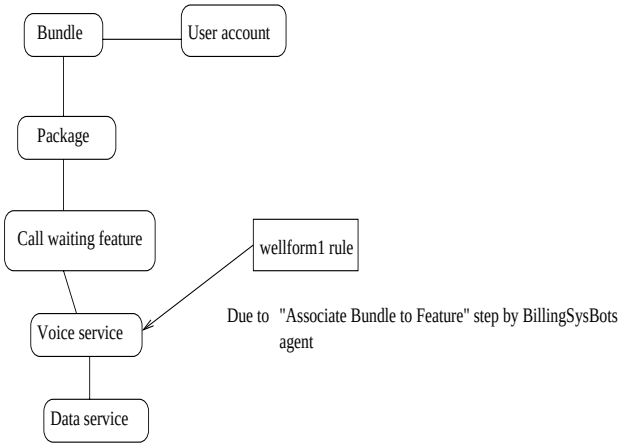

Figure 9. Example of analysis results

processes. The consistency links help highlight whether:

- both comparison processes comply with a certain comparison topic,
- both comparison processes violate a certain comparison topic,
- one process complies with a certain comparison topic while the other violates it.

The annotation lists help point out the steps and agents that are responsible for the analysis outcome. In addition, it is very likely that the sets of step/agent combinations from different traces (processes) corresponding to the same comparison topic are comparable. Such pairs of sets can give insights about the functional similarities or differences between processes and aid the user in making a more objective process comparison.

3.5.1 Choice of initial artifact sets

The set of product artifacts quite often is most useful to be used as the initial artifact set because they contain the most comprehensive annotation lists and they are likely to contain all the artifact elements that need to be checked for comparison topics. Another advantage to this choice of the initial set is high likelihood that the product artifacts are explicitly defined in the process source description. To reduce the amount of computation it is advisable to choose artifacts that are known to be relevant for the chosen comparison topics. Thus, the initial artifact sets **ArtSetA1** and **ArtSetB1** are defined by the steps **Define artifact set A** and **Define artifact set B**. At this point the user can use the developed toolset to specify the artifact sets to be analyzed from the lists of artifacts in the traces. The toolset visualizes the artifacts as graphs based on their Artifact Meta-Format specification. A general layout of the toolset's interface is shown in Fig. 8. In our example, using the toolset's interface, we chose the artifact trace produce by one analyzed processes as **ArtSetA1** and the artifact trace produced by the other analyzed process as **ArtSetB1**.

3.5.2 Choice of comparison topics

Once the artifact sets are chosen, the user should choose the comparison topic to be used for consistency checking. This is done in the **Choose formalized comparison topic** step. The process analysis toolset also lets the user to choose the topics from a list of files with specification such as in Fig. 7.

3.5.3 Checking of artifacts' consistency to formalized comparison topics

Once the initial artifact sets for both processes and the formalized comparison topic are chosen, the comparison process runs a consistency checker that produces consistency links between the formalized comparison topics and artifact elements of the analyzed processes. The current implementation of the comparison process uses the Xlinkit consistency checker by Christian Nentwich et al. ([4], [11]). Next, the user analyzes the two sets of artifact elements that have consistency links to the same formalized comparison topics, but belong to different processes. By using the toolset, the user clicks the "Analyze" button and receives results as sets of consistency links between the specified comparison topics and artifact elements. The toolset also shows the steps and agents responsible for the consistencies or inconsistencies based on the information collected in the annotation lists of the traces. An example of analysis results is shown

in Fig. 9. In the figure the *billDBupdate* artifact is shown as a graph. The rule *wellform1* corresponds to the comparison topic specified in Fig. 7. It stipulates that any voice service must be based on a data service. In this case the consistency link from the rule's representation points to the artifact element responsible for the complience (*Voice service*). The toolset also points out that the step **Associate Bundle to Feature** and agent **BillingSysBots** are responsible for the complience.

In our example, after we chose the artifact sets and chose the file with the comparison topic, we pressed the "Analyze" button and received results representation that indicated the consistency link between the *wellformed1* topic and *voice service* artifact element of the *billDBupdate* artifact for the first process. The toolset also showed it was due to the way agent *BillingSysBots* performed step *Associate Bundle to Feature*. There was no consistency link from the *wellformed1* topic to elements in the artifacts of the traces of the other process. Thus the two processes were functionally different due to actions the *BillingSysBots* agent performed in step *Associate Bundle to Feature*.

4 A closer look at artifacts and agents

In this section we describe one possible way of implementing the analysis process testbed. The ability to execute a rigorously modeled software process is required by the process. The analysis process also assumes that a software process specification is viewed as a template that is instantiated and run in a specific environment that includes agents, among other resources needed by processes. Agents are assumed to be capable of performing different portions of the process. Thus, one of the most important components of our implementation is a software process execution environment that uses the notion of agents.

We chose the Juliette process environment and the associated Little-JIL process specification language ([3], [15]). This choice is due to a sufficient expressiveness of the Little-JIL language for our purpose and the use of the notion of an agent in the Little-JIL and Juliette.

Nevertheless, the implementation of the comparison approach required two main extensions to be made to the Juliette environment: a problem-domain specific agent framework and a generalized artifact specification.

The addition of the agent framework allows for greater software reuse, greater focus of agents' definitions and hence their greater comprehensibility, and it also reduced the time it takes to define new agents in the same problem domain. Essentially, by restricting the structure of an agent and providing agent templates we gain a great deal of leverage in defining agent-based processes.

The use of Artifact Meta-Format (AMF) enables us to define artifacts according to the chosen common artifact section of BF which is the cornerstone of the comparison analysis. In addition, the use of AMF allows for unified handling of artifacts by agents and facilitates generalized instrumentation.

4.1 Agent framework overview

The Juliette software process environment requires only that agents be able to listen to agenda item events. No guidelines are given as to the structure of the agent and its problem solving component. Neither are there any guidelines about the representation of artifacts. The software process designer using the Little-JIL is left to make his/her own decisions regarding the structure of the agents and regarding the format of the communication between them. After specifying several software processes and building agent structures from scratch a software process designer is likely to recognize that it might be beneficial to introduce more constrained frameworks that provide a predefined structure for the agents that spells out how to go about constructing an agent and enables the reuse of the agents originally written for other processes in the same problem domain. It is understandable that the freedom the stock Little-JIL provides is needed because it is impossible to envision and generalize all areas of application. On the other hand, once such areas of application are recognized, it is beneficial to introduce extensions to the software process environment specific to that application area.

This reasoning led to development of the agent framework for problem domain specific processes shown in Fig. 10. The framework has been developed in Java with the use of XML for artifact meta-format specifications.

4.1.1 Agent framework structure

The framework consists of a hierarchy of classes for the definition of agent components at different levels of generalization: the AutoAgent class, process specific agent, StepProduction, and GraphStepProduction classes. The framework also includes the StepGUI class for step specific GUI specification, ArchGraph class as a container for a generalized artifact definition via a graph-based Artifact Meta-Format and artifact reader and writer classes.

The AutoAgent and process specific agent classes are to be inherited by the user agents. The StepProduction and GraphStepProduction classes are to be inherited by the problem solving components that are parts of a user agent. A problem solving component defines the ways in which an agent performs its tasks. In case of software processes that transform artifacts from one format to another a problem solving component corresponds to specification of an artifact format transformation.

The AutoAgent class generalizes functionality for matching the step that the agent is requested to perform with

a set of actions that accomplish the step. This class implements mapping of step names to action sets, artifact storing, and interface with the Little-JIL language interpreter.

The process specific agent class obligates its subclasses to have a certain common functionality set and a certain common structure specifically suited for process agents from a given problem domain. Part of that common functionality provides for communication with the external environment, in this case - with other components of the Juliette software process environment (such as the Little-JIL language interpreter). The framework also provides for action specifications interchangeable between the agents. The set of actions defines the functionality of a particular agent. The process specific agent invokes problem solving components in response to the step related events from the Little-JIL language interpreter.

Depending on the degree of the desired automation the agents can be:

- Human assistants (such an agent would invoke a step-specific GUI and try to assist the human in accomplishing the step)
- Human-modeling (such an agent would attempt to model the duties of a human for process simulation or guidance purposes)
- Automated (such an agent would accomplish steps amenable to complete automation)

The abstract StepProduction class declares a general functionality for an action which is an *Object execute(Object[] args)* method. Thus the framework forces many to one artifact transformations for the actions of an agent. The GraphStepProduction class enforces the use of the graph-based format for the artifacts that actions manipulate. The domain and/or process specific actions are extending the GraphStepProduction class with implementation of specific actions for artifact transformations. An agent instance instantiates the set of process specific actions that define that agent's functionality and calls the *execute* methods on them in response to the agenda events from the process language interpreter.

Any software process is likely to require step specific GUIs because of human involvement in the process execution. This need is recognized in the agent framework by defining a set of step specific GUIs that is checked by the agent every time it receives a new agenda item.

4.2 Artifact Meta-Format overview

The Artifact Meta-Format introduces a graph-based generalization for the artifact formats. It allows definition of a class hierarchy of nodes, links, and their attributes. The meta-format supports inheritance of node and link attributes by type and value. The meta-classes of the meta-format are:

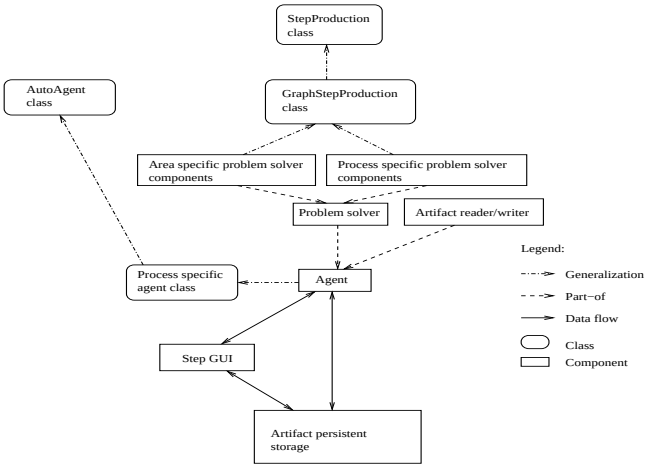

Figure 10. Agent structure

- **MetaComponentClass** - a node class
- **MetaComponentInstance** - a node instance
- **MetaLinkClass** - a link class
- **MetaLinkInstance** - a link instance

The entries of the attribute lists of the meta-class instances are defined by the tuples of their class (attrClass), name, and value.

The artifacts defined in the graph-based AMF are stored persistently as XML files. The corresponding Java framework allows instantiation of a persistent artifact representation via instances of Java classes. The AMF reader class can be thought of as a graph-based AMF class-loader. The framework also provides a writer class that provides functionality for storing artifact instantiations. A visual editor has been developed to support the execution of processes that use this AMF for artifact specification. The artifact graph classes and instances are attributed. The attributes are specified with triples of attribute class, name, and value. The AMF supports inheritance of attributes by types and values.

5 Selected comparison with other process analysis approaches

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 [13], [14]. These comparisons were guided by a formal comparison process, Comparison of Design Methods (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 need to compare modeled processes according to a fixed base framework was also recognized somewhat earlier by Sjaak Brinkkemper et al. ([9]). However, their comparison had no guidelines as explicit and formal as the CDM. The BF suggested in ([9]) has a flat structure as well. The content and construction method are different from the BF in our approach. The BF classes in ([9]) are constructed on the basis of the elements of the process and artifact decomposition units of the compared processes.

An approach to the analysis of processes based on the event data produced by an actual process execution has been suggested in [16]. The kinds of analysis in this work focused on performance characteristics such as times between artifact production and duration of communication events related to a localized aspect of the system produced by the analyzed process.

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 ([6], [7], [8]).

While the above mentioned work by Alexander Wolf, David Rosenblum, Jonathan Cook is a kind of retrospective analysis just as ours is, the kinds of properties investigated by them focused on real-time performance of process activities.

There has also been work on analysis of processes based on a Petri-net-like specification ([2]). Such properties as safety and liveliness (e.g. freedom from deadlock) can be detected through the analysis of this kind. Unlike ([16], [6], [7], [8], [2]) our focus has been on functional analysis of the processes regardless of the dynamic characteristics of process execution.

One of the more recent approaches in process comparison is by Abrahamsson et al. [1]. The authors present comparison of Agile processes. They use an ad-hoc comparison method for comparing processes by high level topics. The focus of their comparison is on organizational and activity sequencing issues rather than on the functional differences. This is primarily due to the fact that Agile software development processes (such as Extreme Programming) omit any description of the guidelines for artifact transformation by their activities. Instead they focus on organizational and activity sequencing issues.

6 Lessons learned and Future directions

The lessons learned from using our process analysis system to compare and analyze the two versions of the telecommunications logistics processes center around the artifact focus, the substantial amount of preparation, and the utility and advantages of the use of the system.

Artifact Focus. The focus on artifacts produced by processes is a very useful one. First, it represents the raison d'etre of processes: the production of processes and services on the basis of various input artifacts. Second, it avoids the tarpit of the widely varying and differing ways that one might accomplish the same tasks. The focus is on the results of the tasks and activities, which parts of the process affect them in which ways, and whether they have certain desired properties or not. And finally, while the ways in which artifacts may be produced vary widely, the artifacts themselves in the same domain are far more likely to be much less variable and far less the subject of disagreement.

Initially, Substantial Preparation. Process analysis and comparisons do not come for free. There is, at least initially, substantial preparation to set the stage for the analysis system. Currently very few processes are sufficiently specified – in fact, this posed a significant problem in our research: there were very few process descriptions in use that we could find that were defined in enough detail to perform our experiments with our analysis system. However, if we are to mature as an engineering discipline and move out the current craft stage, this will have to change.

While there is substantial initial preparation, it should be noted that this preparation serves in a variety of ways for subsequent analyses and comparisons or processes in the same domain. For example, once the processes have been formally specified, they may be used in a variety of comparisons and analyses. Once the base framework has been established for a given domain, it can basically serve for the analyses and comparisons of other related processes. The same is true for the ontology and well-formedness conditions. They are substantially applicable to other work in the same process domain.

Advantages. First, the most obvious advantage is the level of automation provided by the process analysis system. Once the initial preparation has been done, the rest of the analysis is done automatically depending on what input is provided to the system. This is a significant improvement in the state of the art for process comparisons and analyses.

Second, the various analyses and comparisons are repeatable. The points of variability are well defined and have been determined in the preparation. The only remaining point of variability is that where human responses are required in the execution of the processes and that input is controllable as part of the system execution.

And finally, as understanding of the processes grows,

the various automated analyses and comparisons can be extended and evolved in various ways to provide deeper knowledge of the processes under consideration.

Limits. Because of the artifact focus, little has been done at this point to support various useful kinds of process performance analysis. For example, we currently do not support time and cost analyses for process - i.e. comparisons of race and lapse times of processes, nor the amount of effort involved in process execution.

Future Directions. 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 the 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.

One possible direction of future research is performance analysis of software processes. To conduct such analyses the process steps would have to be characterized by their duration, cost, and quality of the artifacts they produce. These step characteristics would have to be obtained by observations of in-place processes. The accuracy of estimates of these characteristics would most influence the reliability of the performance estimates of processes.

Another direction is increase of the level of automation of the analysis system. Greater automation is possible with a more generalized set of possible kinds of artifact element 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 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 analyzed processes.

We would also like to validate the approach further by experiments with a greater variety of more extensive processes from other problem domains.

References

[1] P. Abrahamsson, J. Warsta, M. T. Siponen, and J. Ronkainen. New Directions on Agile Methods: A Comparative Analysis. In *Proceedings of the 25th International Conference on Software Engineering (ICSE 2003), Portland, USA*, pages 244–254, May 2003.

[2] S. Bandinelli, A. Fuggetta, C. Ghezzi, and L. Lavazza. SPADE: An Environment for Software Process Analysis, Design, and Enactment. In A. Finkelstein, J. Kramer, and B. Nuseibeh, editors, *Software Process Modelling and Technology*, chapter 9, pages 223–248. Research Studies Press, Ltd., Taunton, Somerset, England, 1994.

[3] A. G. Cass, B. S. Lerner, E. K. McCall, L. J. Osterweil, S. M. Sutton, Jr., and A. 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.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.

[5] J. M. Cobleigh, L. A. Clarke, and L. 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.

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

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

[8] J. E. Cook and A. 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.

[9] S. B. 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.

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

[11] C. Nentwich, W. Emmerich, and A. Finkelstein. Static Consistency Checking for Distributed Specifications. In *Proceedings of Automated Software Engineering 2001, San Diego, USA*, 2001.

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

[13] X. Song and L. 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.

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

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

[16] A. L. Wolf and D. S. Rosenblum. A Study in Software Process Data Capture and Analysis. In *ICSP 2 - 2nd International Conference on Software Process*, pages 115–124, February 1993.