Department of Computer Science
San Marcos, TX 78666

Report Number TXSTATE-CS-TR-2006-14

Software architecture for flexible integration of process model synthesis methods

Rodion Podorozhny
Anne Ngu
Dimitrios Georgakopoulos
and Dewayne Perry

2006-05-08

Software architecture for flexible integration of process model synthesis methods

Rodion Podorozhny[1], Anne Ngu[1], Dimitrios Georgakopoulos[2], and Dewayne Perry[3]

[1] Texas State University, Computer Science Dept., 601 University Dr.,
78666 San Marcos, TX, USA
{rp31, hn12}@txstate.edu
[2] Telcordia Research Austin Center, 106 6th St., Littlefield Bldg.,
Austin, TX, USA
dimitris@research.telcordia.com
[3] The University of Texas, ECE Dept.,
78712 Austin, TX, USA
perry@ece.utexas.edu

Abstract. In this paper we suggest an architecture that would integrate various methods for synthesis of a software process model based on domain knowledge about artifacts, process fragments, tools, and limited process execution observations. Our approach suggests using a meta-process specification for integration of various process synthesis methods to provide a generalized process model. We also propose using a process execution observation for confirmation of a synthesized process model.

1 Introduction

The area of software engineering is mainly concerned with methods for synthesis and analysis of software. In this paper we are focusing on an automated method for synthesis of software processes based on very few observations and various domain knowledge. The automation of synthesis brings many benefits: repeatability, efficiency, cost reduction. In addition, a process synthesized by our approach is itself automated due to the use of learning methods for synthesis of control flow decisions. Some of the early work on process discovery is based on analysis of retrospective data about the process executions [1,2,3,4]. The process models discovered with these early methods focus on the set of activities and the process control flow. The process synthesis method integration we suggest expands the earlier process discovery work in several directions. Our approach is novel because it allows to automatically synthesize a process based on very few observations of process execution (process instance) thanks to the use of learning methods and a greater variety of domain knowledge. It provides a process model enhanced with activity attributes and control flow decision-making.

In this work we are focusing on the architecture for integration of methods for synthesis of various process aspects that comprise a process model. The architecture

assumes availability of the domain knowledge about the process artifacts, tools to be used by a process, the environment in which a process is to be enacted, and a limited number of observations of process execution. In a way, we propose an approach that involves synthesis of a process model via limited observations of examples of its execution.

2 Problem statement and approach intuition

The problem calls for a synthesis of a *rich* process model based on a *single observation* of process execution. The assumptions include availability of domain knowledge including artifact well-formedness constraints, bill of materials, generalized description of a finished product or service, tools used for process execution, process fragments, environment conditions for process execution and others. Each of these kinds of information can be either *incomplete* or have some degree of *uncertainty*

Next let us give an explanation of our intuition about a possible solution. Abstractly, this problem seems to require us to synthesize an accepting computation abstraction specification (e.g. Turing machine) for a language based on one sentence of that language. The retrospective data about previous executions of this process is assumed to be very limited. How could we produce a generalized process specification if we are given only one path which even might be providing an unreliable product ? It seems that doing so without the knowledge about artifacts is impossible due to the lack of information. Thus, our approach prescribes construction of a generalized process model based on whatever domain knowledge is available and then checking whether the observed execution path is a feasible path through our model. It is quite possible that the observed path itself provides a low quality product. Nevertheless, if that path is feasible in the "guessed" process model then it validates the model. Otherwise, the path is considered to be a counterexample, the process model is considered incorrect and thus it is modified to satisfy the observed path.

At the implementation level, we suggest specifying the synthesis process itself in a process language, this allows for flexibility of the integration of methods for synthesis of particular process aspects based on particular domain knowledge available.

2 Process model aspects

In this section we will discuss in greater detail the kinds of information needed to specify a *rich* process model. The final integrated process synthesis system must be able to synthesize the specification of these aspects from available domain knowledge and the observation.

By a process we understand a systematic, disciplined way of either producing a final artifact or delivering a service. Since it is possible to model a service as an elec-

tronic artifact we will use the term “final artifact” or product to denote a process outcome.

By a generalized process model we understand a process “program” that, if specified in a rigorous process specification language, can be instantiated by a process enactment engine provided an input and environment specifications. The following aspects characterize a *rich* generalized process model:

- set of activities
- each activity characterized by an identifier, interface (sets of input and output artifact types), pre-condition and post-condition of activities
- constraints on control flow of activities, relaxed to the extent possible
- hierarchical decomposition of activities according to various sets of criteria
- specification of artifact type system manipulated by activities
- specification of resource needs of activities including people roles and tools
- predictors that provide distributions for cost, duration of individual activities and assessment of quality of output artifacts of an activity.

The execution of a process is greatly influenced by guidelines or reasoning mechanisms for control flow choices which are not part of process specification per se, rather they are part of the resources specification. These guidelines define reasoning of resources assigned to execution of process activities. It is our intent that the process synthesis system will discover such guidelines to accompany the generalized process model. The generalized process model must not over-constrain the control flow. That is why a straightforward mapping of the observation onto the generalized process model is unacceptable. For instance, it might turn out that a certain sequence of artifact transformations was enforced by scarce resource availability while in the generalized model those artifact transformations can happen in parallel given abundant resources.

4 Architecture of process synthesis integrator

The previous section outlined the various aspects of a process model that must be ultimately synthesized. Intuitively it is clear that various synthesis methods would be needed to synthesize those aspects. The choice of a particular method depends on the aspect, kind, amount, and certainty of domain knowledge available, nature of domain knowledge, whether the chosen methods are an effective match. Thus the architecture must allow for great flexibility in the choice of the set of activities involved in the synthesis, the tools used, the sequence of their application. Therefore we suggest using a process specification and an associated process execution system for the synthesis process itself.

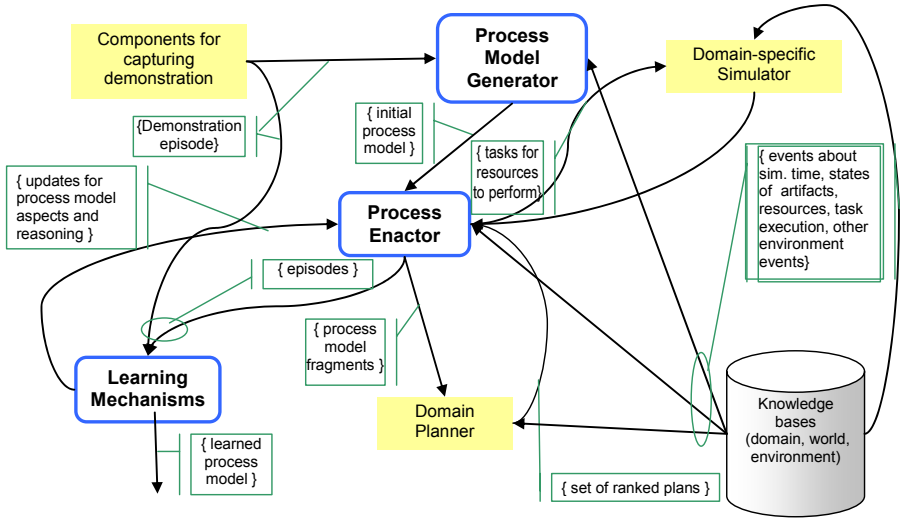

Figure 1. Architecture of a process synthesis integrator

According to Perry&Wolf [11] an architectural description is comprised of elements, form, and rationale. The architecture of the process synthesis integrator depicted in Fig. 1 contains the elements (sets of components and connectors) and the form (constraints on their interconnections). The architectural description in Fig. 1 uses rounded corner rectangles to denote the internal components, yellow rectangles denote external components, directed arcs to denote data flow, rectangles associated with the arcs to denote the data, the cylinder to denote persistent storage.

Below we will describe the architecture rationale. The Process Model Generator forms an initial process model in a chosen process specification language based on the domain knowledge (product, process, known execution traces, resource utilization) and refines some aspects of the initial process model based on the observation. Next Process Enactor, Domain-specific Simulator, Learning Mechanisms, and Domain Planner synergistically subject the initial process model to dynamic analysis and refinement. The interaction of these architecture elements is as follows. The Process Enactor receives an initial process model and, provided the cost, duration, quality of activities can be estimated, submits a fragment of the process to the Domain Planner. If the estimates of cost, duration, quality are not available the Process Enactor chooses the first set of alternative activities based on the control flow constraints specified in the initial process model.

Once the set of the activities is identified and resources are assigned, the Simulator starts modeling the artifact transformations by the modeled resources. It notifies the Process Enactor of various events such as completion of activity instances by resources, time ticks, contingencies due to the modeled environment, contingencies due to artifact states. The Process Enactor reacts to the events from the Simulator or events generated by the Process enactor itself (e.g. time-out of activity completion). The reactions themselves are specified as process fragments, they can be either domain-specific or default reaction processes (e.g. reaction to time-out of activity com-

pletions, resource unavailability contingencies, resolution of contradictions between needs of concurrently running process fragments, pre-emption control decisions).

The Simulator models the transformations of artifacts as these artifacts are processed by resources according to manipulations prescribed by process activities. The state of modeled artifacts is used by the Learning Mechanisms to update the knowledge they accumulated about various aspects of the process such as the control flow decisions, decisions on the sets of activities to be executed, activity attributes and resource assignments. Thus the simulation and learning of a single process instance continues until either all the activities are finished and/or final artifacts are produced or predefined time runs out or it is determined that there are insufficient resources to produce final artifacts. On completion of a process instance the Learning Mechanisms update their knowledge based on the outcome.

The process synthesis integrator will attempt to recreate an observation by simulation based only on the process model, resource knowledge and knowledge of environment changes. Considering the space restrictions we cannot provide much more detail about the components of this architecture. Some additional information about process synthesis is mentioned below.

4.1 Process Model Generator: As our process specification and execution system we chose the industry-level Atlas system developed at Telcordia Research Center in Austin. The Atlas system has already been successfully used in a number of projects [12][13][14]. Its design is tightly connected to a database used for persistent storage of processes and process fragments which results in the linear dependency of the scalability of the process execution system on the scalability of the database.

We have some initial experience with the process model generator implemented in the Atlas process language.

Next we give a high-level description of the principle of operation of the process model generator component of the integrator architecture.

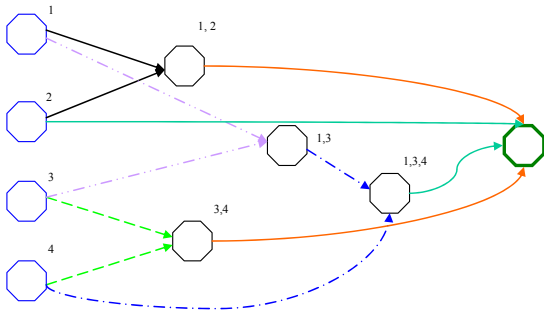

Figure 2. Partial order state space.

In Fig. 2 each state corresponds to subsets of process artifacts that comply with artifact well-formedness constraints. It is possible to generate these states based on the bill of materials and the well-formedness constraints of artifact combinations. The transitions indicate a partial order of artifact composition transformations. In Fig. 2 we assume there are four product parts that comprise the final artifact(s). Octagons represent states that are marked with the sets of parts. For instance, the blue octagons

denote the 4 individual atomic artifacts from the bill of materials and are marked with artifact “1”, artifact “2” and so on. The well-formedness constraints allow for 5 physically possible combinations of these artifacts, one of which is the final product (all artifacts combined). Fig. 2 shows two possible ways of deriving the final product.

Based on the partial order state space, the process model generator must construct a hypothetical general process model.

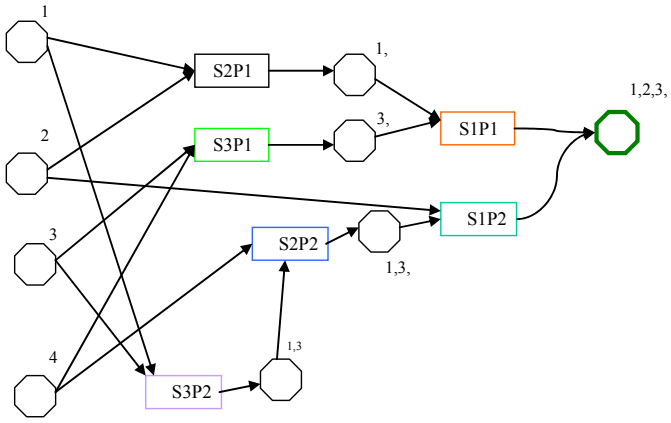

Figure 3. Partial order activity state.

The result of this mapping is shown in Fig. 3 in which the rectangles correspond to activities and the color of their border corresponds to the color of transitions in Figure on which they were based. By using data flow analysis we can construct a partial process model that represents functional decomposition of activities and the constraints on their execution. The partial process model in an Atlas-like process language for this assembly example is shown in Fig. 4.

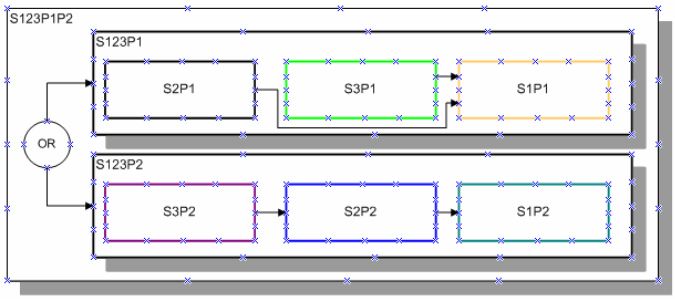

Figure 4. Partially-specified process

Rigorous well-formedness constraints are not available in all process domains. Then it is difficult to generate a partial order state space based on artifact combinations. Other domain knowledge has to be used to form an initial process model in such cases. For instance, the set of tools with well-defined functionality or domain-specific intents can be used to suggest either sequences of tool applications or partial order of sub-intents that can lead to a product.

4.2 Learning mechanisms: The learning mechanisms we considered are supervised neural network learning, reinforcement learning, and evolutionary computation. The hybrid method will leverage the strengths of each of the individual methods. Such process aspects as duration, cost, quality, process activity clustering (to generate or confirm activity decomposition captured by the synthesized process) and guidelines/automation for making control flow choices in the synthesized process are expected to be learned by these mechanisms.

In situations where the information consists of discrete state variables and discrete actions, and the state is fully known, reinforcement learning [9] is an effective approach. In other situations, the state is not fully known, and the state and the actions are described with continuous values. Such situations are difficult for reinforcement learning because it is hard to discretize the space and to identify which utility values need to be changed. However, recurrent neural networks can be constructed through evolutionary learning, and can perform robustly in such situations [5,7].

These learning methods are brought together to learn an effective decision policy for the process. The decision policy is initially represented statistically in terms of neural network weights and Q-tables. Using standard techniques for knowledge extraction, this knowledge is then translated into a rule-based description of the process [8,9,10].

6 Conclusions and Future work

The experience we have with synthesizing assembly processes based on the bill of materials and the final product description is encouraging, yet the vast majority of work still lies ahead. We need to make the process model generator be able to use a more generalized description of the final product that does not directly refer to the artifacts in the bill of materials. The other approaches to generation of the initial process model that do not rely on availability of the rigorous artifact well-formedness specifications must be implemented. The learning mechanisms must be evaluated based on their applicability to capturing the various process aspects described. Finally the whole integrator must be evaluated on real-life process domain knowledge from various process domains, to name a few: mechanical assembly, web-services integration, mechanical object control procedures, software development, crisis responses.

We would like to thank Dr. Misty Nodine of Telcordia Research for discussions, useful comments, and editing, Dr. Donald Baker of Telcordia Research for his help with the Atlas process execution system, and Prof. Risto Miikkulainen of the University of Texas, Austin for discussions and help with the application of reinforcement learning and neuroevolution methods to the process synthesis.

References

1. 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), July, 1998, pp 215-249.
2. 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), August 1998, pp 650-663.
3. 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), April, 1999.
4. Alexander L. Wolf and David S. Rosenblum, "A Study in Software Process Data Capture and Analysis", *ICSP 2 - 2nd International Conference on Software Process*, February, 1993, pp. 115—124.
5. Moriarty, D. E., Schultz, A. C., and Grefenstette, J. J. (1999). Evolutionary Algorithms for Reinforcement Learning. *Journal of Artificial Intelligence Research*, 11:199-229.
6. Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986), Learning Internal Representations by Error Propagation. In Rumelhart, D. E. and McClelland J. M., *Parallel Distributed Processing*. Cambridge, MA: MIT Press.
7. Stanley, K. and Miikkulainen, R. (2002). Evolution of Neural Networks through Augmenting Topologies. *Evolutionary Computation*, 10:99-127.
8. Stanley, K. and Miikkulainen, R. (2004). Competitive Coevolution through Evolutionary Complexification. *Journal of Artificial Intelligence Research*, 21:63-100.
9. Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.
10. Towell, G. G. and Shavlik, J. W. (1993). The Extraction of Refined Rules from Knowledge-Based Neural Networks. *Machine Learning*, 13:71-101.
11. Dewayne E. Perry and Alexander L. Wolf. “Foundations for the study of Software Architecture”, *ACM SIGSOFT Software Engineering Notes*, 17:4 (October 1992).
12. D.Georgakopoulos, H.Schuster, A.Cichocki, and D.Baker. (1999) “Collaboration Management Infrastructure in Crisis Response Situations”, Technical Report CMI-009-99, Microelectronics and Computer Technology Corporation
13. D.Baker, A.R.Cassandra, H.Schuster, D.Georgakopoulos, and A.Cichocki. (1999) “Providing Customized Process and Situation Awareness in the Collaboration Management Infrastructure”, Proceedings of the 4th IFCIS Conference on Cooperative Information Systems (CoopIS'99)
14. Dimitrios Georgakopoulos, Hans Schuster, Donald Baker, and Andrzej Cichocki. (2000) “Managing Escalation of Collaboration Processes in Crisis Mitigation Situations”, Proceedings of the 16th International Conference on Data Engineering (ICDE'2000)
15. Wagner, Thomas A., Garvey, Alan J. and Lesser, Victor R., “Satisficing Evaluation Functions: The Heart of the New Design-to-Criteria Paradigm”, UMass Computer Science Technical Report 1996