2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications

An Approach to Software Process Design and Implementation Using Transition Rules

Andre L. Ferreira and Ricardo J. Machado
Dept. de Sistemas de Informação
Universidade do Minho, Guimarães, Portugal
{andre.ferreira,rmac}@dsi.uminho.pt

Mark C. Paulk
Institute for Software Research
Carnegie Mellon University, PA 15213 USA
mcp@cs.cmu.edu

***Abstract*— When software organizations adopt several best practices models into one single environment the number of activities and processes tends to increase. Managing effectively a large and complex system of processes requires process modeling capabilities at higher levels of abstraction. This paper presents a modeling approach for software process design and implementation to deal with the increasing size and complexity of large systems of processes. A model based on an extension of UML 2.0 Component Diagrams is used to develop a process architecture. A transition mechanism using Little-JIL process programming language is also proposed to support the refinement of a process architecture to a process implementation.**

Software Process Modeling; Software Process Architecture; Multi-model Environemtns; Little-JIL

I. INTRODUCTION

To improve competitiveness software organizations are adopting several best practices models simultaneous with the objective of obtaining the cumulative added value each model [1]. Each model adopted imposes new requirements leading to an increase in the number of activities performed. e.g., if one considers a full implementation of CMMI or ISO12207 combined with Information Technology Infrastructure Library the number of expected activities is overwhelming.

With the high number of activities, the number of dependencies between activities tends to increase, e.g., outputs from certain activities become inputs to a set of related activities. These dependencies, if not managed properly, can lead to process inefficiencies when new practices are adopted and can greatly limit analysis and improvement efforts by process improvement groups. Being able to model precisely these activities and related dependencies becomes a necessity to deal with aforementioned limitations.

A process model provides a representation of activities to be performed by the organization. It delivers process guidance and supports enforcing and partial automation of the software development process. Developing a process model should consider development phases of process design and process implementation. Process implementation involves the specification of low-level process details to allow a process to be enacted. Process design involves developing a process architecture with the necessary process abstractions to allow, consistently relating and incorporating process elements, supporting reuse enhancement and tailoring of processes [2]. Each of these phases deals with process modeling at different levels of abstraction.

Developing a process model requires the use of a Process Modeling Language. In the past, complexity and inflexibility of these languages limited their successful adoption [3]. More recent proposals have gained interest not only by the research community but also by industry. SPEM (Software and Systems Process Engineering Meta-model), a UML based modeling language defines process basic constructs, like *activities*, *tasks* and *roles* among others to support process implementation. Little-JIL [4] is a non-UML process modeling language subject of increasing research that focuses on agent coordination that also targets process implementation. Concerning the process design phase, research has focused on defining process abstractions to support the definitions of process architectures. Recently, the concept of software process lines has been introduced. The goal is to define relevant process structures to support process design where reuse and process variability are considered at an abstraction level where low-level process detail is absent.

In this scenario, requirements for modeling processes at higher levels of abstraction (design phase) differ from modeling at a lower level (implementation phase) thus, is likely that each requires different modeling languages. This paper proposes a model to support the design phase and a set of transition rules to make a transition from design to implementation phase. For the design phase we defined a model based on an extension of UML 2.0 Component Diagrams and considered Little-JIL process programing language for the implementation phase. The transition rules guide a systematic mapping of UML components to Little-JIL process coordination constructs. This approach is aimed primarily to process engineers responsible for development and maintenance of process systems that tend to have a large number of activities and dependencies.

This paper is organized as follows: the next section presents relevant research in the area of process modeling, focusing on approaches that deal with modeling of processes at higher levels of abstraction. Section 3 introduces a model to support process design based on UML components diagrams to define processes at higher levels of abstraction. It is described how it can be linked with Little-JIL process constructs to support the transition between design and implementation phases. Section 4 describes a small example of applying the transition rules

978-0-7695-4488-5/11 $26.00 © 2011 IEEE
DOI 10.1109/SEAA.2011.57

introduced in section 3. Section 5 concludes and outlines future work.

II. SOFTWARE PROCESS MODELING

The first perspectives on the subject of software process modeling have been set by Osterweil [5] and Humphrey [6]. They relate development of software processes to Software and Systems Engineering. Osterweil argues that process development should include phases of requirements specification, architecture and design, and then process implementations providing ready to execute work instructions to software developers. Albeit a theoretical analogy to systems and software engineering was set, the bulk of research in the field of software process engineering has focused on process implementation phase and a clear insight of what process design should consider is a recent research subject.

Several authors have developed approaches to support software process modeling at higher levels of abstraction. Basically, structuring concepts that organize process related abstractions are proposed to support process implementation. Zhao et al. [7] argue the application of agent-based technology to support software process modeling. They consider a software processes as a collaboration of groups of process agents, which are responsible for managing software development activities. The concept of *Process Landscape* is described by Gruhn and Woolen and focuses on modeling complex processes at high levels of abstraction [8]. The underlying context is organizations carrying out parts of a process with different levels of autonomy and distributed among different locations. A process landscape considers process clusters as an abstraction to group processes at the same level of detail. The landscape predominant view is of logical dependencies between clusters of processes and interfacing is defined as a first class modeling entity.

A recent trend in software process modeling are Software Process Lines which are motivated by the need to have processes more aligned with distinct development needs. It targets highly heterogeneous project environments where development contexts may demand different development practices. In software process lines modeling process commonalities and variability assumes center relevance to support process reuse and adaptation. The concept is comparable to the well-established concept of software product lines. Work related to process lines is described by Barreto et al. [9] and Munch et al. [10]. The emphasis of the work is on describing how to create a set of processes by comparing alternative (variability) workflow elements to core process elements (commonalities). Alternative workflows are mapped to features diagrams that help development of project-specific processes.

The common line of thought in high level process modeling approaches is that process abstractions are used to manage coherent sets of concerns, ignoring low level modeling details. Higher order process elements like process architectures and process components are defined to be used along with basic or primary process modeling elements. Recently, expressing process commonality and variability is considered relevant to align process implementation with development needs. Several approaches document approaches to express variability at the implementation level [11, 12] and few deal with it at higher levels of abstraction.

This paper presents an approach to make a transition from a process architecture specification to a process implementation. A conceptual model is proposed to support the creation of a process architecture. Little-JIL process programming language is used to specify process implementation and also support the transition from design to implementation phases. Interfaces declared between architectural elements are translated to Little-JIL language constructs using a set of transition rules.

III. SOFTWARE PROCESS DESIGN AND IMPLEMENTATION

According to Humphrey, a process architecture should provide a supporting infrastructure relating process abstractions to support the activity of process engineering. It should facilitate consistently relating and incorporating process elements, supports reuse, enhancement and tailoring of processes. In this paper the perspective in defining a process architecture is to focus on the narrow aspect of high level components interfacing and abstract away the details of lower level process definitions. Component types and connector types are defined to support the creation of a process architecture. A process architecture should also allow an explicit transition to low level process implementations. With these goals in mind we:

- Define a process design model as an extension of UML 2.0 meta-model Component Diagrams to guide the definition of process architectures based on reusable process architectural components.
- Describe how Little-JIL is used to specify low-level process definitions and detail how Little-JIL semantic constructs can be used in the refinement of process architectural components by applying a set of transition rules.

A. Process Design Model

Fig. 1 depicts the class diagram for the process design model as an extension (classes in grey) of the UML 2.0 Component diagrams.

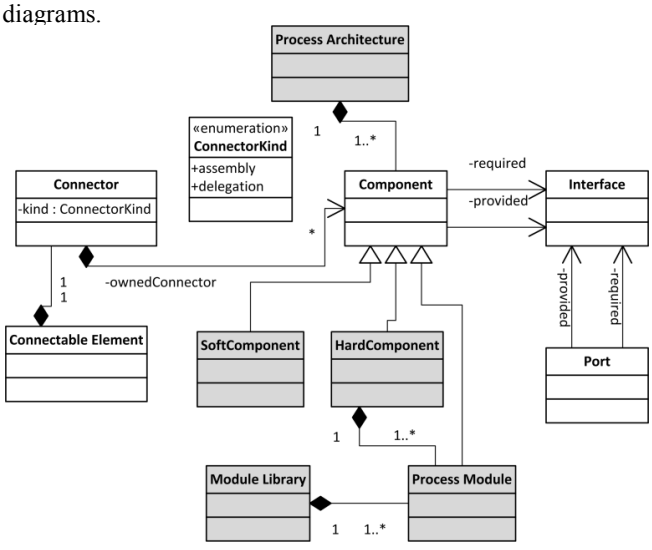

Fig. 1. Process Design Model

A *Module Library* provides a repository of *Process Modules*. The goal is to support management of textual descriptions of methods applied in software development by defining process modules. We adopt the perspective of SPEM specification that software processes should be defined, mostly, but not only, by combining existing methods from a method repository.

A *Process Architecture* is composed of process Components. A distinction is made between *SoftComponent* and *HardComponent*. Soft components are used to declare architectural components with no semantics defined e.g., required functionality is yet to be defined to satisfy identified process needs (e.g., satisfy an expected quality practice or goal) or its implementation is assured by external parties, e.g., outsourced functionally. A hard component has its semantics defined by combining existing process modules from a module library. Next, we describe how these process architectural building blocks have their semantics implemented by using Little-JIL language constructs.

B. *Transition rules from Design to Implementation phases using Little-JIL*

Little-JIL [4] is defined as an agent coordination, visual based, with a formal graphical syntax process programming language. It implements sharp separation of concerns by separating internal specification of how agents carry out their work and their coordination. The basic semantic construct of Little-JIL is the *Step* (see Fig. 2). A step represents a unit of work and contains the specification of the type of an agent needed to perform the task associated with that step. Steps are connected to each other with badges that represent both control and artifact flow. A Little-JIL process program resembles a work breakdown structure where the coordination specification of the agents is external and is limited to the observable behavior of each agent. It does not specify a data model for parameters and resources or provides information on how to carry out basic units of work called *Leaf Steps*.

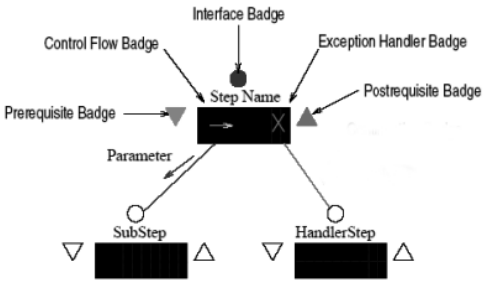

Fig. 2. Little-JIL Step

Three Little-JIL semantic constructs are particularly useful in supporting the transition from process architecture to actual process implementations, namely: messages, channels and reactions. A Little-JIL *Message* is sent during execution of a Little-JIL program to signal occurrence of events. An event can be associated to the need of performing a set of practices outside of the scope of a specific practice. Messages can be used to declare events and are associated to Little-JIL *Reactions*. Reactions provide a mechanism to respond to the arrival of messages to Little-JIL steps. Often, performing a specific set of practices outside the scope of what is being executed is required. *Reactions* provide a mechanism to combine pairs of process modules by attaching the root step of a second process module to any step in the hierarchy of steps of a first module. Reactions along with messages allow a very precise identification of what and when is occurring between two separate process modules. A *Channel* provides a communication mechanism not tied to the hierarchical structure of Little-JIL process programs. It allows data-centric synchronization and support communication of potentially parallel or independent threads of execution. Based on these process constructs a set of transition rules are defined to help the transition from an architectural specification to a process implementation.

Transition rule 1 - a process module is implemented de defining a Little-JIL process tree. When a process module is connected through an interface to another process module the origin and target steps of both process trees are identified.

Transition rule 2 – A *reaction* arc is declared between the parent step of the requiring step of the first process tree and the root step of the connecting process tree.

Transition rule 3 - A *message* identifies the event associated with the expected collaboration between process modules. The message is declared on the process step (actually on the arc that links the step to its parent) that requests the collaboration and in the reaction arc linking both process trees.

Transition rule 4 - A *channel* is declared in the parent step of the originating step to allow the communication of parameters between process trees. Parameters to be communicated are identified and the flow is declared in the arcs of communicating steps by binding operations available in Little-JIL. Based on these transition rules a small example is described next, where a process component from process architecture is implemented using Little-JIL.

IV. PROCESS DESIGN

This section describes an example of applying the transition rules introduced. A process component built by combining process modules from a module library is used to demonstrate how a portion of a process architecture can be translated to a process implementation using Little-JIL constructs and transition rules. Fig. 3. depicts two *HardComponents*, composed of 4 process modules. The Static Verification component includes 3 process modules. The *Code Review* and *Configuration Management of General Documents* process modules are connected by an interface indicating that information is flowing between both modules. In this case the Code Review process module has a requiring interface named *report_channel* indicating that some information must flow between modules. The connecting module is to provide a service to fulfill that requirement.

Applying rule 1, each process module is to be implemented by a Little-Jill process tree and the connecting steps from both process trees are identified. Fig. 4 depicts partially a process tree for *Code Review* that includes the *Preparation* step and associated sub-steps and the *Configuration Managements of General Documents* with the *CM* root step and associated sub-

steps. Only a portion of the trees is depicted, the emphasis is on the steps where the interface will be implemented.

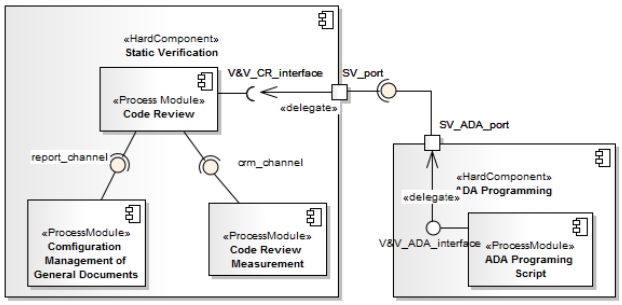

Fig. 3. Process Component Diagram

Applying transition rule 2 a *reaction* arc is declared between the parent step (*Preparation*) of the originating step *(Fill Review Report*) and the target step (*CM*). The reaction is signaled by the lightning badge in the parent step.

Applying rule 3, a message is declared in the arc of the originating step (*report_msg: StepFinishedEvent* □), indicating which event should trigger the execution of the tree associated to the reaction arc. The message (*report_msg)* is also declared in the reaction arc indicating it should be triggered by the reception of the message.

Applying transition rule 4, the *report_channel* in Fig. **3** is implemented by declaring a channel (report_channel) in the parent step (*Preparation*) of the origin step (*Fill Review Report*). Intervenient parameters are written (*report_done* ►*report_channel*) and read (*report_done* ◄ *report_channel*) from the channel. Bindings are declared in the reaction arc. The parameter *document* of the connecting process tree is bound with *report_done* from the originating tree.

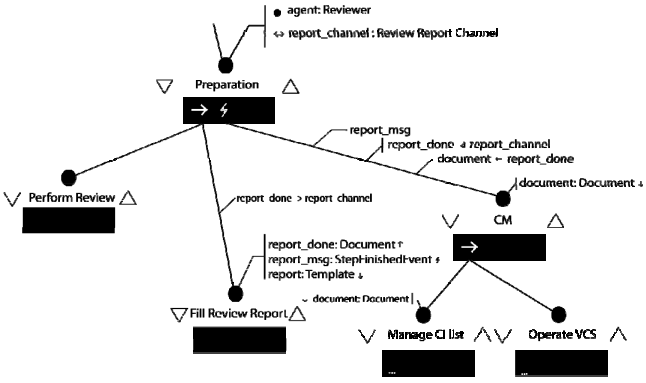

Fig. 4. Interface ***report_channel*** for Static Verification HardComponent implemented using Little-JIL

V. CONCLUSION AND FUTURE WORK

When software organizations adopt multiple best practice models into one single environment, a high number of quality requirements and goals are expected to be satisfied. This often leads to an increase in the number of activities and processes performed by the organizations, leading to large and complex systems of processes that become hard to analyze and manage effectively. This paper presents a conceptual model to support the definition of process architectures as an approach to deal with increasing size and complexity of these systems. Further, it describes how a process architecture can be refined to a process implementation using Little-JIL process programming language along with a set of transition rules. An example is given based on a small software development scenario. In future research we aim to evolve the process design model to deal with process variability at the design phase, exploring the concept of software process lines. Process adaptation or tailoring at higher level of abstraction is helpful in facilitating process implementation in highly heterogeneous project environments. A process architecture should provide a reusable structure and form the basis for developing different types of process development life cycles.

REFERENCES

[1] J. Siviy, P. Kirwan, L. Marino, and J. Morley, "The Value of Harmonizing Multiple Improvement Technologies," *SEI,* 2008.

[2] R. Conradi and M. Jaccheri, "Process Modelling Languages," in *Software Process: Principles, Methodology, and Technology*. vol. 1500, J.-C. Derniame, B. Kaba, and D. Wastell, Eds., ed: Springer US, 1999.

[3] R. Bendraou, Je, x, ze, J. M. quel, M. P. Gervais, and X. Blanc, "A Comparison of Six UML-Based Languages for Software Process Modeling," *Software Engineering, IEEE Transactions on,* vol. 36, 2010.

[4] A. Wise, "Liittle-JIL 1.5 Language Report," Department of Computer Science, University of Massachusetts, Amherst, MA 2006.

[5] L. Osterweil, "Software processes are software too," presented at the Proceedings of the 9th international conference on Software Engineering, Monterey, California, USA, 1987.

[6] W. S. Humphrey and M. I. Kellner, "Software process modeling: principles of entity process models," presented at the Proceedings of the 11th international conference on Software engineering, Pittsburgh, Pennsylvania, United States, 1989.

[7] X. Zhao, K. Chan, and M. Li, "Applying agent technology to software process modeling and process-centered software engineering environment," presented at the Proceedings of the 2005 ACM symposium on Applied computing, Santa Fe, New Mexico, 2005.

[8] V. Gruhn and U. Wellen, "Structuring complex software processes byProcess landscaping", in *Software Process Technology*. vol. 1780, ed: Springer Berlin / Heidelberg, 2000.

[9] A. Barreto, E. Duarte, A. R. Rocha, and L. Murta, "Supporting the Definition of Software Processes at Consulting Organizations via Software Process Lines," Seventh International Conference on the Quality of Information and Communications Technology (QUATIC), 2010.

[10] J. Münch, M. Vierimaa, and H. Washizaki, "Building Software Process Line Architectures from Bottom Up," in *Product-Focused Software Process Improvement*. vol. 4034, ed: Springer Berlin / Heidelberg, 2006.

[11] R. Lee, T. Martínez-Ruiz, F. García, and M. Piattini, "Towards a SPEM v2.0 Extension to Define Process Lines Variability Mechanisms," in *Software Engineering Research, Management and Applications*. vol. 150, ed: Springer Berlin / Heidelberg, 2008.

[12] R. Martinho, J. Varajao, and D. Domingos, "A Two-Step Approach for Modelling Flexibility in Software Processes," in *23rd IEEE/ACM International Conference on*, *Automated Software Engineering* , 2008.