Key UMass Amherst Resources for SERC Collaboration

Leon J. Osterweil (ljo@cs.umass.edu)
Lori A. Clarke (clarke@cs.umass.edu)

Lab. For Adv. SW Engineering Research (LASER  http://laser.cs.umass.edu)

Presentation to SERC Research Review
Malvern, PA
October 16, 2009
Microprocess:
A “Horizonal Technology Cut”
Process is a central issue in system engineering

- **Goal:** systems that are fast, agile, safe, effective,…
- **Approach:** processes for
  - Building, analyzing, using, evolving, training,…
- **Processes specify how systems are**
  - Developed, used, evolved,…
  - As collaborations of people, software, devices
- **(Development) processes are used to build systems**
  - Better systems come from better processes
- **(Usage) processes guide how systems are wielded**
  - Better processes exploit systems better
- **System improvement from Process Improvement**
Example: Agile System Evolution

- How to quickly, surely enhance deployed systems?
- Improve their:
  - Speed, functionality, usability, robustness
  - Quickly, correctly, reliably
- Requires processes for:
  - Coordinating development
    - People, tools, management
  - Assuring product qualities
  - Deploying product
  - Training users
- Requires being sure of these processes
A case in point--*Coming up later in this presentation*

- Agile development (e.g. Scrum) can
  - Speed systems to deployment
  - Close system improvement loops fast
- But can it also
  - Allow defects to creep in unnoticed?
  - Render development vulnerable to poor developer performance?
- Process Analysis:
  - Can be used to identify single points of failure leading to development hazards
  - The basis for removing such defects
  - Process Improvement => System improvement
Key UMass Capability:
Technology-Based Continuous Process Improvement

- Process is a central issue in system engineering
  - Collaboration of people, software, devices, etc.
- Process Improvement is a central goal
- UMass concepts, tools, and technologies support process:
  - Definition
  - Analysis/evaluation
  - Education
  - Performance/execution/simulation
  - Evolution
Our approach is based upon MICROPROCESS research
Process as Object

Resources:
- People
- Money
- Tools
- Time

Input Artifacts

Process

Outputs
- Artifacts
- Effects on the world

Other Behaviors
- Money used
- Time spent
- Errors committed
Macro-Process Focus

Process

Input Artifacts

Resources:
- People
- Money
- Tools
- Time

Outputs
- Artifacts
- Effects on the world

Other Behaviors
- Money used
- Time spent
- Errors committed

Common approaches:
- CMMI, ISO 9000, Six Sigma
Micro-Process Focus

Needed approach: Define, analyze, Automate, precise process definitions

Inputs
- Artifacts
- Resources: People, Money, Tools, Time

Outputs
- Artifacts
- Effects on the world
- Other Behaviors: Money used, Time spent, Errors committed
Bridging Micro- and Macro-

- Use details of process model to predict how system attributes and behaviors are produced
- Suggest changes, predict their effects
- Validate changes before they are made

Each has interests in all of these
Each knows it needs the other’s approach
What we learn from analogies to other disciplines (e.g. medicine)

- Macro- approach comes first
- Limited success in engineering
- Micro- approach/theory follows
- Facilitates more effective engineering
  - Improved predictability
  - Reduced uncertainty
  - Greater cost effectiveness
  - Better understanding of limitations
  - Fewer surprises

We are here (?)
Time for SERC to take the lead in showing how:

Microprocess technology can transform System Engineering
The Microprocess Vision

- Define processes with a precisely defined executable language
- Analyze processes for defects
  - And fix them to improve them
- Execute, simulate the defined processes
  - To provide user Guidance
- Use them as the basis for education and workforce development
The Microprocess Vision

- Define processes with a precisely defined executable language
- Analyze processes for defects
  - And fix them to improve them
- Execute, simulate the defined processes
  - To provide user Guidance
- Use them as the basis for education and workforce development

Apply this to the many processes implied in the SERC Research Strategy
Little-JIL process language features

- Blends proactive and reactive control
- Coordinates human and automated agents
  - Without favoring either
- Emphasizes exception specification, management
- Facilities for abstraction, scoping, hierarchy
- Artifact flow, resource utilization integrated
- Concurrency, synchronization with message-passing
- Articulate specification of resources
- Semantics for aborting activities
- Pre/post condition constructs
- Facilities for human choice

There are many more
Little-JIL: A Real Language with Precise Semantics

- Process definition is a hierarchical decomposition
- Think of steps as procedure invocations
  - They define scopes
  - Copy and restore argument semantics
- Encourages use of abstraction
  - Eg. subprocess reuse
Little-JIL: A Real Language with Precise Semantics

- Process definition is a hierarchical decomposition
- Think of steps as procedure invocations
  - They define scopes
  - Copy and restore argument semantics
- Encourages use of abstraction
  - Eg. subprocess reuse

A key feature in distinguishing this from less formal languages (e.g. workflow)
“Step” is the central Little-JIL abstraction

- Interface Badge
  - (parameters, resources, agent)

- Prerequisite Badge
- Postrequisite Badge

- TheStepName

- Substep sequencing
- Artifact flows
- Exception type
- continuation

- Handlers
Top level of Little-JIL Scrum process definition
Top level of Little-JIL Scrum process definition

creates “sprint backlog” from “product backlog”
Top level of Little-JIL Scrum process definition

creates “sprint backlog” from “product backlog”

Executes tasks From the “sprint backlog”
Top level of Little-JIL Scrum process definition

- Creates “sprint backlog” from “product backlog”
- Reviews sprint and updates “product backlog”
- Executes tasks from the “sprint backlog”
Elaboration of “Execute Tasks” step

After the task is completed, perform review

If the review fails, rework the task
The Basis for Engineering

- Such definitions can then be the subjects for sound analyses
- They can be executed
  - To provide user guidance
- They can be support education and training
- They form the basis for disciplined improvement
A Continuous Process Improvement Environment

- Process, property, resource definition languages with rigorous semantics
- Collection of analysis capabilities: error detection and security analysis
- Multiple derived representations
- Resource Specification And Management
- Process Improvement decisions based on technology assessment
- Property Specifications
- Finite-State Verification
- Fault-Tree Analysis
- Discrete-Event Simulation
- Role-Based Analysis

To perform this step, the provider must have the patient-name. The provider should first order test(s) on computer, and then order test(s) on patient chart. During any of these steps, if the required resources are not available, order test(s) is considered to have failed.
Finite-State Verification

Process Engineer → System Translator → Property

Property Holds on All Paths Through the Model

Domain Expert

Property Generator

Property Does Not Hold: Counterexample

PROPEL

Property Representation

FLAVERS

Reasoning Engine

Little-JIL

Process Definition

Process Engineer

Property

System Translator

Domain Expert

Property
Finite State Verification of Properties

Process property: Task must be reworked if the review fails

After the task is completed, perform review

If the review fails, rework the task
Using Propel to define property “Task must be reworked if the review fails”
Corresponding (Disciplined) English Description of Property

QuickTime™ and a decompressor are needed to see this picture.
FLAVERS-generated trace showing how the property can be violated

- Task is performed
- Review fails
- Task is reworked
- Review fails again

Suggesting a correction to the process-technology-driven process Improvement
Fault Tree Analysis (FTA)

- A well accepted and widely practiced hazard analysis technique
- Systematically identifies and reasons about all possible events that could lead to a given hazard
  - Create fault tree for a hazard
  - Analyze each fault tree
- Analysis results can be used to improve the process => process improvement
Hazard: Artifact "sprintbacklog" from "Sprint" is wrong
Minimal Cut Sets Can Be Generated Automatically

<table>
<thead>
<tr>
<th>Minimal Cut Set</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(ReviewFailed thrown by &quot;Execute Task&quot;) &quot;Execute Task&quot; produces wrong sprintbacklog</td>
<td>2</td>
</tr>
<tr>
<td>!(ReviewFailed thrown by &quot;Execute Task&quot;) productbacklog to &quot;Sprint&quot; is WRONG</td>
<td>2</td>
</tr>
<tr>
<td>&quot;Plan for Sprint&quot; produces wrong sprintbacklog !(ReviewFailed thrown by &quot;Execute Task&quot;)</td>
<td>2</td>
</tr>
<tr>
<td>sprintbacklog to &quot;Sprint&quot; is WRONG ReviewFailed thrown by &quot;Execute Task&quot;</td>
<td>2</td>
</tr>
<tr>
<td>&quot;Execute Task&quot; produces wrong sprintbacklog</td>
<td>1</td>
</tr>
</tbody>
</table>

Single Point of Failure
Location of Single Point of Failure

Single point of failure!
Discrete-Event Simulation

- Use the Little-JIL process models, combined with a resource manager to drive discrete-event simulation
  - Evaluate alternative resource allocations
    - More architects, more programmers, or more testers
Little JIL Interpreter Architecture

Step Sequencer

Resource Manager

Parameter Manager

Agenda Item

Agenda Manager

Which step next?

Who does it?

What is it done to?

Outputs

Agendas

Human Agents

Non-Human Agents
Little JIL Simulator Architecture

The diagram illustrates the architecture of the Little JIL Simulator. It includes components such as:

- **User**
- **TimeLine**
- **Agent Behaviors Specification**
- **Event Arrivals**
- **Next Event**
- **Agenda Item**
- **Agendas**
- **Step Sequencer**
- **Resource Manager**
- **Parameter Manager**
- **Simulated Human Agents**
- **Non-Simulated Non-Human Agents**
- **Simulation Results**

The flow of the process involves:
- User specifying the arrival distribution.
- TimeLine determining the next event.
- Agent Behaviors being sequenced.
- Step Sequencer deciding the next step.
- Resource Manager assigning tasks.
- Parameter Manager tracking outputs.
- Simulation Results completed.
Life Cycle Process Engineering: An engineering discipline applied to the domain of processes

- Integrated approach to process
  - Definition
  - Analysis
  - Simulation
  - Execution
  - Education
  - Improvement
Toolset Status

- Little-JIL language 1.5 is defined
- LASER currently distributes
  - Visual JIL graphical editor
  - Propel property specification system
  - FLAVERS finite state verification system
  - Fault tree generator and analyzers
- Working, but not distributed yet
  - Juliette runtime execution system
  - ROMEO resource manager
  - JSim finite state simulation system
Toolset Integrated through Eclipse
A SERC-relevant Application: Agile/Adaptive Software Development

- Applying this approach to processes for agile/adaptive system development
- Some examples can be drawn from Agile Methods, Extreme Programming
- Case in point: Scrum-oriented development
  - Define Scrum
  - Analyze Scrum
    - Identify and fix weaknesses
  - Train and educate
  - Provide automated guidance in doing Scrum
Some Research Areas

- What semantic features should a microprocess definition language have?
- How to specify its semantics?
- What analysis approaches should be explored?
  - What can be learned from each
- What is the architecture of a microprocess execution system?
  - What components?
  - How integrated?
- Software artifact provenance
  - What is needed?
  - How to provide it?
Questions
and
Discussion
Backup Slides
Four parts to a Little-JIL Process

- Coordination diagram
- Artifact space
- Resource repository
- Agents
An Articulate Process Can Help Answer Questions Like These

- Where does output go?
- What to do when reviews fail?
- What causes this rework?
- What portion of activity should be done?
- How do we break this cycle?
High-Level Process

Software Development

Requirements → High-Level Design → Low-Level Design → Coding
Requirements Process

Emphasizing Rework

1. Declare Rqmt Element
2. Develop Rqmt Element
3. Define Rqmt Element
4. +
5. Declare and Define Rqmt
6. ∆
7. ~ Rqmt OK
8. Develop Rqmt Element
9. =
10. Requirements

Rqmt OK
Develop Rqmt Element

In/Out: Rqmt Spec, Rqmt History
Out: Rqmt Elt

Rqmt OK
In: Rqmt Elt
Out: Rqmt Rpt

In: Rqmt Spec, Rqmt History
Out: Rqmt Elt \<-(\{Rqmt Elt\} U Rqmt Elt)

In: Rqmt Spec, Rqmt History
Out: Rqmt Elt

Declare and Define Rqmt

In/Out: Rqmt Spec, Rqmt History
Out: Rqmt Elt

Declare Rqmt Element

Define Rqmt Element
1. Declare HLDesign Element

2. Declare HLDesign Elements

3. Define HLDesign Elements

4. Develop Rqmt Element

5. High-Level Design

6. Declare and Define HLDesign Elements

7. Requirements

8. A Rqmt OK

9. Rqmts OK

10. X
Defining Module Interfaces

Define a Module Interface

Code All Modules

Requirements

High-Level Design

Low-Level Design

Coding

Develop Rqmt Element

Develop Code Modules

Coding

Code

Interface

OK

~Rqmts OK

~HLD OK

~LLD OK

~Code OK

~A Rqmt OK
FSV Using Propel and FLAVERS