PSL and Flow Models

Conrad Bock
Michael Gruninger
8/2004
Overview

- Approaches to system specification
  - Model vs instance-based
  - Example from structural specification
- PSL introduction
  - Why PSL is not yet another “L”.
  - Basic PSL concepts
  - How PSL is used
- PSL application
  - Behavior Classification
- Conclusions
Ontology Languages

- Terms
- Thesauri
- Formal Taxonomies (OKBC)
- Frames (OKBC)
- Description Logic-based (DAML+OIL)

Glossaries & Data Dictionaries
- ‘Ordinary’ Glossaries
- Data Dictionaries (EDI)

Thesauri
- Structured Glossaries
- Principled, informal hierarchies

XML DTDs
- XML Schema

DB Schema
- Formal Taxonomies

Data and Process Models (UML, ORM, EXPRESS)
- FOL, OCL, PSL

Formal Languages & Automated Reasoning
Left of Red Line (User view)

- **OWL:**
  ```xml
  <owl:Class rdf:ID="Mammal"/>
  <owl:Class rdf:ID="Dog">
    <rdfs:subClassOf rdf:resource="#Mammal"/>
  </owl:Class>
  </owl:Class>
  ```

- **UML:**
  ![UML Diagram]

- **C++:**
  ```
  struct Dog : Mammal {} 
  ```

- **English:** Dog is a kind of Mammal (or UML repository)
Left of Red Line (Machine view)

- **OWL:**
  ```xml
  <owl:GWJK rdf:ID="LHGY"/>
  <owl:GWJK rdf:ID="OUYT">
    <rdfs:LNCGWJKYO rdf:resource="#LHGY"/>
  </owl:GWJK>
  ```

- **UML:**
  ![UML Diagram]
  (same for repository)

- **C++:**
  ```cpp
  eghc OUYT : LHGY {}  
  ```

- **English:** OUYT er a bfvc yo LHGY
Specialized Interpreters

- Interpreters built for each LORLL ...
- ... by humans who “know” the meanings.
- “Consensus” achieved by:
  - Documentation, runtime examples, model theories, RORLL’s.
- LORLL’s are fundamentally:
  - Not self-documenting.
  - Don’t say what they mean.
- Result: Interoperability problems.
Right of Red Line

- **FOL:**
  
  \[
  \forall x \ (\text{Dog } x \implies \text{Mammal } x)
  \]

- Self documenting because it refers to *instances* of domain concepts (\(x\)).

- Still need interpreter for “forall”, etc.

- Small set of highly reusable and composable constructs.
Right of Red Line

- Simple things can be hard to say:

\[
\text{(forall } (?x) \\
\text{ (implies } (\text{Pet } ?x) \\
\text{ (exists } (?y) \\
\text{ (and } (\text{Person } ?y) \\
\text{ (own } ?y ?x) \\
\text{ (forall } (?z) \\
\text{ (implies } (\text{own } ?z ?x) \\
\text{ (= } ?z ?y))))))\]

\[
\text{ Person} \quad \text{ owned_by } \quad \text{ own } \quad \text{ Pet}
\]
Right of Red Line

- And some things impossible:

Each one expanding to increasingly complicated expression

Each person owns:

{ 0 pets, or
{ or 1 pet,
{ or 2 pets
{ or 3 pets . .
Left/Right Comparison

- No silver bullet

- Left of Red Line (modeling):
  - Usually more concise.
  - Easier to add concepts.
    - Except for updating tools.
  - Difficult to interpret correctly.

- Right of Red Line (instance-based):
  - Self-documenting.
  - Sometimes very difficult to add concepts.
    - Once done, tools understand the new concepts.
  - Usually more verbose.
Flow models: **LORLL**

- **UML 2:**

  - ChangeColor
    - Paint → Dry

- **BPEL:**

  ```
  <process name="ChangeColor">
    <sequence>
      <invoke operation="Paint"/>
      <invoke operation="Dry"/>
    </sequence>
  </process>
  ```

- **C:**

  ```c
  void ChangeColor
  {
    Paint();
    Dry();
  }
  ```
Specialized Interpreters

- Interpretation is needed to know:
  - Can any other activities occur between Paint and Dry?
  - What behaviors can occur concurrently with painting?
  - How soon after painting must drying occur?
  - Is it possible under exceptional conditions for drying not to happen?
PSL: RORLL (instance-based)

- Instances of processes
  - Individual executing processes.
  - ChangeColor executed at 10:21am ET 9/1/2003 at factory 1.

- Execution sequence
  - Sequences of executing steps in the process, perhaps some concurrently.
  - Paint executed at 10:22am, then Dry at 10:40am, etc.

- Small set of highly reusable constructs.
Basic PSL Concepts

- **Occurrence** is an execution of an **Activity**
  - like Paint executed at 10:22am ET 9/1/2003 at factory 1.

- **Activity** is a RORL-like
  - like Paint or Dry.
Basic PSL Concepts

- **In FOL:**

\[
\forall \text{a} \ \forall \text{occ} \\
\quad \exists \text{a} \ (\text{activity}\_\text{occurrence} \ ?\text{occ} \ ?\text{a}) \\
\quad (\text{activity} \ ?\text{a}) \\
\quad (\text{occurrence}\_\text{of} \ ?\text{occ} \ ?\text{a}) \\
\quad (\text{activity}\_\text{occurrence}\_\text{of} \ ?\text{occ} \ ?\text{a}) \\
\quad \forall \text{occ} 
\quad (\text{occurrence}\_\text{of} \ ?\text{occ} \ ?\text{a}) \\
\quad (\text{activity}\_\text{occurrence} \ ?\text{occ}) \\
\quad (\exists \text{a} \\
\quad (\text{activity} \ ?\text{a}) \\
\quad (\text{occurrence}\_\text{of} \ ?\text{occ} \ ?\text{a})) \\
\quad (\forall \text{occ} \ ?\text{a1} \ ?\text{a2}) \\
\quad (\text{occurrence}\_\text{of} \ ?\text{occ} \ ?\text{a1}) \\
\quad (\text{occurrence}\_\text{of} \ ?\text{occ} \ ?\text{a2}) \\
\quad (\text{equal} \ ?\text{a1} \ ?\text{a2}))
\]

- PSL is an execution-based way of describing processes.
- PSL happens to be expressed in FOL, but it is not bound to FOL.
Basic PSL Concepts

- Executions happen one after another.
- Covers all activities happening anywhere.
- Occurrence has multiple successors, one for each (theoretically) possible next occurrence.
Occurrence Tree

- Tree of all possible execution sequences, including those that
  - are not physically possible.
  - are not specified by the user.

- Not stored anywhere, just referred to.
Process Specification in PSL

- Constraints on the occurrence tree.
- Example: drying immediately follows all painting.
Process Specification in PSL

- Constrain occurrences of Paint to be followed by occurrences of Dry:

\[
\text{forall} \ (?\text{occPaint}) \\
\text{implies} \\
\text{and} \ (\text{occurrence_of} \ ?\text{occPaint} \ \text{Paint}) \\
(\text{legal} \ ?\text{occPaint}) \\
(\text{and} \ (\text{legal} \ (\text{successor} \ \text{Dry} \ ?\text{occPaint}))) \\
(\text{forall} \ (?\text{otherSuccessor}) \\
\text{implies} \\
(\text{not} \ (\text{equal} \ ?\text{otherSuccessor} \\
\ (\text{successor} \ \text{Dry} \ ?\text{occPaint}))) \\
(\text{not} \ (\text{legal} \ ?\text{otherSuccessor}))))))))
\]
Process Specification in PSL

- Above says that Dry happens after Paint under executions of ChangeColor.
- Other processes may use Paint without Dry.
Complex Processes in PSL

- Paint happens immediately after Dry under executions of ChangeColor.
- ChangeColor specification does not constrain OtherProcess above
Complex Processes in PSL

- Complex occurrences and activities composed of primitive ones:

  - Successor moved down to PrimitiveOccurrence.
  - Occurrence tree covers every step at finest grain.
Complex Processes in PSL

- Execution sequencing within complex activity:

  Executions immediately following (under a complex occurrence)
  \[ \text{next\_subocc} \]

  Executions following sometime (under a complex occurrence), not necessarily immediately.
  \[ \text{min\_precedes} \]

- \text{min\_precedes} defined in terms of successor.
- \text{next\_subocc} in terms of \text{min\_precedes}:

  \[
  \text{forall} (\ ?s1 \ ?s2 \ ?s3) \\
  (\text{iff} (\text{next\_subocc} \ ?s1 \ ?s2 \ ?a) \\
  (\text{and} (\text{min\_precedes} \ ?s1 \ ?s2 \ ?a) \\
  (\text{not} (\text{exists} (\ ?s3) \\
  (\text{and} (\text{min\_precedes} \ ?s1 \ ?s3 \ ?a) \\
  (\text{min\_precedes} \ ?s3 \ ?s2 \ ?a))))))
  \]
Complex Processes in PSL

- Constrain occurrences of ChangeColor to be composed of sequential occurrences of Paint and Dry:

\[
(\forall \text{occChangeColor} \quad (\text{implies}
  \quad (\text{occurrence_of} \ \text{occChangeColor} \ \text{ChangeColor})
  \quad (\exists \text{occPaint} \ \text{occDry})
  \quad (\text{and}
    \quad (\text{occurrence_of} \ \text{occPaint} \ \text{Paint})
    \quad (\text{occurrence_of} \ \text{occDry} \ \text{Dry})
    \quad (\text{subactivity_occurrence} \ \text{occPaint} \ \text{occChangeColor})
    \quad (\text{subactivity_occurrence} \ \text{occDry} \ \text{occChangeColor})
    \quad (\text{next_subocc} \ \text{occPaint} \ \text{occDry} \ \text{ChangeColor})))
\]
Process Specification in PSL

- Simple things can be hard to say:

- 6 nonoverlapping orderings
- 6 partially overlapping orderings
- 1 complete overlapping order
Process Specification in PSL

- **Benefits:**
  - Self-documenting (says what it means).
  - Small set of highly reusable concepts.
  - Improved interoperability by reducing ambiguity.

- **Disadvantages**
  - Sometimes difficult to add concepts.
  - More verbose in many cases.

- **Additional benefit to process modeling:**
  - More flexible constraints (classification, rules).
Advertise the Distinction

- Common to think of PSL as yet another “L” (UML, BPEL, etc).
- PSL is a semantic foundation for all LORR flow/process models.
- Even KBSI substitutes flow models for PSL (PDS).
- More expressive and less ambiguous than flow models.
How to Get Best of Both Worlds?

- Research topic
- Translate models to instance-based
  - Not enough: Users ignore instance-based
- Instance-based aid to example testing
  - Check examples (user-defined or actual) against instance-based semantics.
  - Generate examples from instance-based specs to be checked by users or system.
- Annotate modeling languages with instance-semantics.
Behavior Classification

Classification of process executions:

\[
\text{(forall } (?\text{occFFS}) \\
\text{(implies } (\text{occurrence\_of } ?\text{occFFS} \text{ FastFoodService}) \\
\text{(exists } (?\text{occFS}) \\
\text{(and } (\text{occurrence\_of } ?\text{occFS} \text{ ?FoodService}) \\
\text{(forall } (?s) \\
\text{(implies } \text{(subactivity\_occurrence } ?s \text{ ?occFFS}) \\
\text{(subactivity\_occurrence } ?s \text{ ?occFS})))))
\]
How to abstract commonality?
Behavior Classification

- Food Service has these steps:
  - Order, Prepare, Serve, Eat, Pay

- With these constraints:
  - Order, Prepare, and Serve always happen before Eat.
  - Serve happens after Prepare and Order.
  - Pay can happen anytime in the process.

- Need to partially specify a process as incrementally-defined constraints.
Behavior Classification

- Flow models are not expressive enough:
  - Prepare and Order are not concurrent.
  - Pay is not concurrent with other steps.
Behavior Classification

- Prepare sometime before Eat under FoodService:

\[
\text{forall } (?\text{occFoodService}) \\
\text{implies} \\
(\text{occurrence_of } ?\text{occFoodService} \text{ FoodService}) \\
(\exists (?\text{occPrepare} \ ?\text{occEat}) \\
(\text{and} \\
(\text{occurrence_of } ?\text{occPrepare} \text{ Prepare}) \\
(\text{occurrence_of } ?\text{occEat} \text{ Eat}) \\
(\text{subactivity_occurrence } ?\text{occPrepare} \ ?\text{occFoodService}) \\
(\text{subactivity_occurrence } ?\text{occServe} \ ?\text{occFoodService}) \\
(\text{min_precedes } ?\text{occPrepare} \ ?\text{occEat} \text{ FoodService})))))
\]
Behavior Classification

- Possible enhancement to UML notation.
Behavior Classification

- **FastFoodService**: Prepare sometime before Order

  \[
  \forall \text{occFastFoodService}.
  \exists \text{occPrepare, occOrder}.
  \begin{align*}
  \text{occurrence_of(occFastFoodService, FastFoodService)} \\
  \text{occurrence_of(occPrepare, Prepare)} \\
  \text{occurrence_of(occOrder, Order)} \\
  \text{subactivity_occurrence(occPrepare, FoodService)} \\
  \text{subactivity_occurrence(occOrder, FoodService)} \\
  \min_precedes(occPrepare, occOrder)
  \end{align*}
  \]
Behavior Classification

- Execution traces classified by process specifications (constraints).
Behavior Classification

- Possible enhancement to UML notation.
- Requires updating tools and services.
Abstraction vs Ambiguity

- Both omit information.
- One does it intentionally and explicitly, the other doesn’t.
- Example:
  - Did the modeler intend that no other step occur between Paint and Dry?
  - Design intent is lost.
- A proper abstraction would say what the modeler actually meant.
- PSL does this with the occurrence tree.
PSL Myths

- Too precise
  - Can write “partial programs”
  - Can make useful distinctions
    - Weak and strong ordering
    - Weak and strong concurrency
    - Activity viewpoints
    - Occurrence, activity, activity class
  - Distinctions provide power

- Can’t say everything
  - Some things too complicated
More Information

- See paper “PSL: A Semantic Domain for Flow Models”.
- More applications of PSL to flow modeling.
- Parameterized activities.
- Inputs and outputs, see NISTIR.
- Concurrency and external activities.
- Decision points/merges.
- Closure.