VDM at Newcastle: applications, methods and tools

Jeremy Bryans, Joey Coleman, John Fitzgerald, John Hughes, Cliff Jones

Centre for Software Reliability
School of Computing Science

Introduction

1. Background
2. Applications:
   1. Dynamic Coalitions
   2. Access Control
3. Methods:
   1. Partial Functions
   2. Proof
4. Tools:
   1. Open Proof Support

Background

- Large Dependability Research Group
- Fault tolerance, dependability, formal methods.
- Unusually interdisciplinary approach
- Big recent projects included
  - DIRC
  - RODIN
  - BAESYSTEMS DCSC

Introduction

FT for “Ambient Systems”

- ♦ Dynamic
- ♦ No centralised control
- ♦ Heterogeneous

Applications

Mainly from a Security Perspective

- ♦ Collaboration with DSTL
- ♦ And with the GOLD project in Chemical Engineering.
- ♦ Dynamic Coalitions
- ♦ Access Control
- ♦ (Access Control in Dynamic Coalitions!)

Applications: Dynamic Coalitions

- Virtual Organisations, Strategic Alliances, Virtual Enterprises, Dynamic Coalitions, …
- Approach: (lightweight formal modelling in VDM-SL)
  - identifying "dimensions". Focus on dimensions relevant to information flow
- We mapped out a space of possibilities for dynamic coalitions using these dimensions. It includes an initial case study from the chemical engineering domain.
- Long-term: helping design-time decision-making, using the validation techniques of VDMTools. (want API & scripting for exploring information flow control policies)

Dimensions of Dynamic Coalitions, *Bryans, J. W., Fitzgerald, J. S., Jones, C. B., Mozolevsky, I.* Tech. Report **CS-TR: 963** School of Computing Science, University of Newcastle, Jul 2006

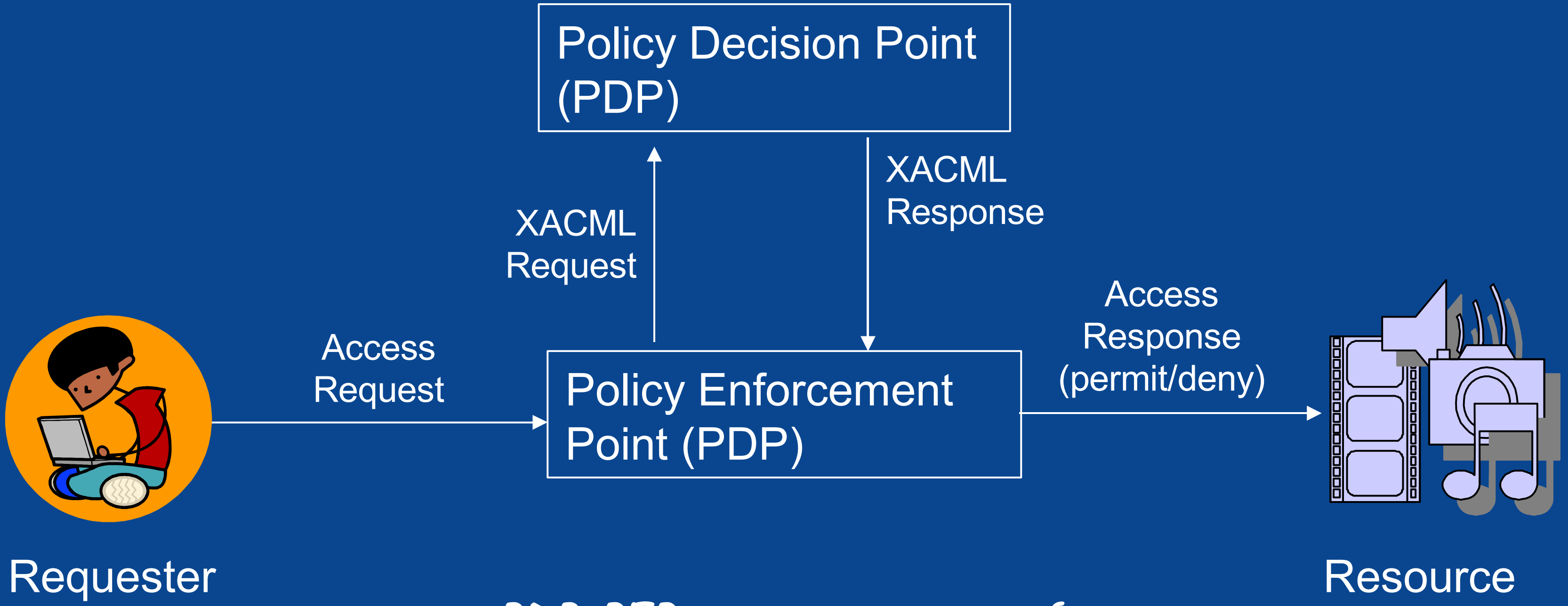
Applications: Access Control

- Aim – make an integrated suite of validation techniques *available* to access control system designers
- Design choices
  - build on a model with a formal semantics, and
  - remain faithful to XACML (eXtended Access Control Meta-Language) – a de facto standard for access control system description
- This means we want the *structure* of the XACML model replicated in our formal model, as well as providing a faithful semantic interpretation

Applications: Access Control

Policy embedded in the PDP

Full XACML admits context-dependent decisisons


Policy Decision Point (PDP)
XACML Request
XACML Response
Access Request
Policy Enforcement Point (PDP)
Access Response (permit/deny)
Requester
Resource

PDP+PEP serve as a sort of execution monitor

Applications: Access Control

Some key concepts (simplified)

```
PDP :: policies : set of Policy
       policyCombAlg : CombAlg;

Policy :: target : Target
          rules : set of Rule
          ruleCombAlg : CombAlg;

Rule :: target : Target | <Null>
        effect : Effect;

Effect = <Permit> | <Deny> |
         <Indeterminate> | <notApplicable>;

CombAlg = <denyOverrides> |
          <permitOverrides>;
```

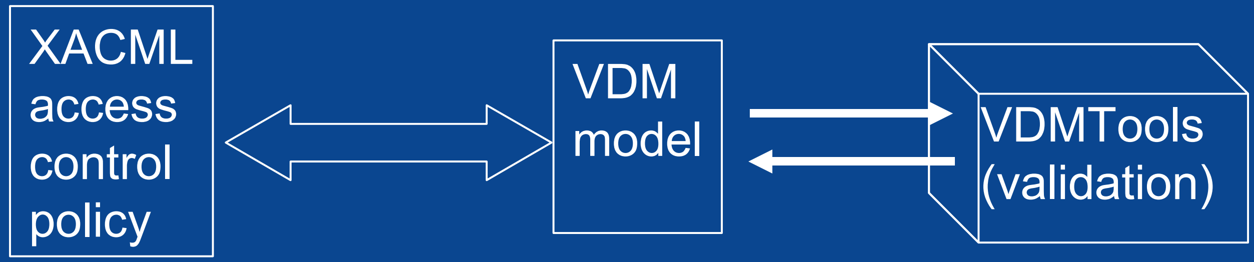
```
Request :: target : Target;

Target :: subjects : set of Subject
          resources : set of Resource
          actions : set of Action;

Action = <Assign>|<View>|<Receive>;
Subject = <Anne> | <Bob> |
          <Charlie> | <Dave> ;
Resource = <Int>|<Ext>;

Role :: set of Subject;
```

Applications: Access Control



- We want a machine implementable translation in both directions, so the XACML can be automatically updated when changes are made in the model.
- What validation techniques?
  - testing
    - against set of individual access requests, or more detailed scenarios
  - internal consistency
    - do rules contradict each other?
  - comparison with earlier versions of policies
    - to check for unwanted effects of updates to policies

Applications: Access Control

Where are we?

- VDM model of (simplified) XACML language (context-free rules only)
- translation
  - XACML to VDM (in part)
  - VDM to XACML
- validation techniques – through VDMTools
- context-dependent rules
- automatic derivation of economical test suites
- workflow descriptions to derive test suites
- least privilege
- translation of full XACML to VDM and
- model checking?

done

to do

Model Based Analysis and Validation of Access Control Policies, *Bryans, J. W., Fitzgerald, J. S., Periorellis, P.*
Tech. Report **CS-TR: 976** School of Computing Science, University of Newcastle, Jul 2006

Methods

- We take the view that validation through execution is great but not as far as we can go in exploiting the formal semantics of a modelling language.
- Methods work is mainly geared around advanced formal analysis via proof.
- Actually we are doing little on testing & model checking and need Overture colleagues to collaborate on these.

Methods: handling partial functions

Partial functions arise routinely in models and in code.

Modelling languages intending to support proof need to address this issue in the logic.

Which of the following do you expect to be true:

```
5/0 = 1 or 5/0 <> 1

forall i:int & fact(i) >=0 or fact(-i) >= 0

hd [] = 5
```

The Logic of Partial Functions is one way of handling undefined terms.

Different decisions in different formalisms (e.g. Z)

Also different decisions in VDMTools (McCarthy conditional interpretation)

Methods: handling partial functions

e1	e2	e1 and e2
T	T	T
T	F	F
T	*	*
F	T	F
F	F	F
F	*	F
*	T	*
*	F	F
*	*	*

If e1 then e2 else false

e1	e2	e1 cand e2
T	T	T
T	F	F
T	*	*
F	T	F
F	F	F
F	*	F
*	T	*
*	F	*
*	*	*

Weaker than Classical logic, but, e.g.

cand is not commutative

Methods: handling partial functions

Typed LPF has been implemented in a (over-restrictive?) logical framework.

Next Steps:

• How do differing approaches “trade off” in specification, refinement, interpreter-based validation and coding?

• What are the consequences for program design where specification annotations are built in to code (c.f. ESC/Java, Spec#)?

• Can we completely implement typed LPF in the frames of major provers such as PVS and HOL?

The Typed Logic of Partial Functions and the Vienna Development Method, *Fitzgerald, J. S.*
Tech. Report **CS-TR: ???** School of Computing Science, University of Newcastle, Aug 2006

Methods: proof

We have been exploring correctness proofs based on using structural operational semantics.

$$\boxed{\text{If-T}}\ \frac{(b,\sigma) \xrightarrow{e} \mathbf{true}, \quad (th,\sigma) \xrightarrow{s} \sigma'}{(\mathbf{if}\ b\ \mathbf{then}\ th\ \mathbf{else}\ el\ \mathbf{fi}, \sigma) \xrightarrow{s} \sigma'}$$

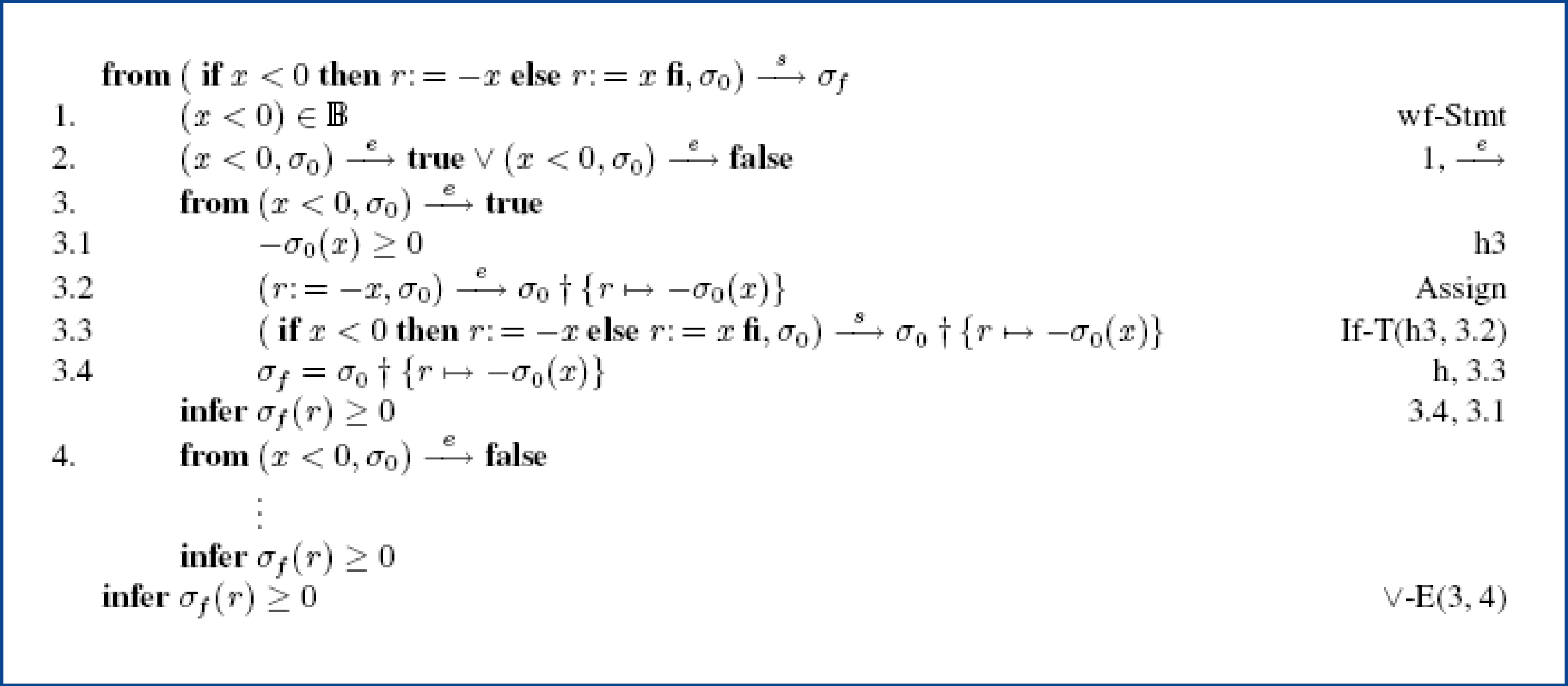
$$\boxed{\text{If-F}}\ \frac{(b,\sigma) \xrightarrow{e} \mathbf{false}, \quad (el,\sigma) \xrightarrow{s} \sigma'}{(\mathbf{if}\ b\ \mathbf{then}\ th\ \mathbf{else}\ el\ \mathbf{fi}, \sigma) \xrightarrow{s} \sigma'}$$

$$\boxed{\text{Assign}}\ \frac{(rhs,\sigma) \xrightarrow{e} v}{(lhs := rhs, \sigma) \xrightarrow{s} \sigma \dagger \{lhs \mapsto v\}}$$

Rules are based on transition relations describing (here) stmt and expr evaluation/execution.

These rules *are* the PL semantics.

Methods: proof



```
      from $(\textbf{if } x < 0 \textbf{ then } r := -x \textbf{ else } r := x \textbf{ fi}, \sigma_0) \xrightarrow{s} \sigma_f$
1.        $(x < 0) \in \mathbb{B}$                                                                          wf-Stmt
2.        $(x < 0, \sigma_0) \xrightarrow{e} \textbf{true} \vee (x < 0, \sigma_0) \xrightarrow{e} \textbf{false}$       $1, \xrightarrow{e}$
3.        from $(x < 0, \sigma_0) \xrightarrow{e} \textbf{true}$
3.1           $-\sigma_0(x) \geq 0$                                                                          h3
3.2           $(r := -x, \sigma_0) \xrightarrow{e} \sigma_0 \dagger \{r \mapsto -\sigma_0(x)\}$                          Assign
3.3           $(\textbf{if } x < 0 \textbf{ then } r := -x \textbf{ else } r := x \textbf{ fi}, \sigma_0) \xrightarrow{s} \sigma_0 \dagger \{r \mapsto -\sigma_0(x)\}$   If-T(h3, 3.2)
3.4           $\sigma_f = \sigma_0 \dagger \{r \mapsto -\sigma_0(x)\}$                                               h, 3.3
          infer $\sigma_f(r) \geq 0$                                                                      3.4, 3.1
4.        from $(x < 0, \sigma_0) \xrightarrow{e} \textbf{false}$
              $\vdots$
          infer $\sigma_f(r) \geq 0$
      infer $\sigma_f(r) \geq 0$                                                                          $\vee$-E(3, 4)
```

A proof using the SOS rules allows us to prove properties of programs using the semantics directly.

Methods: proof

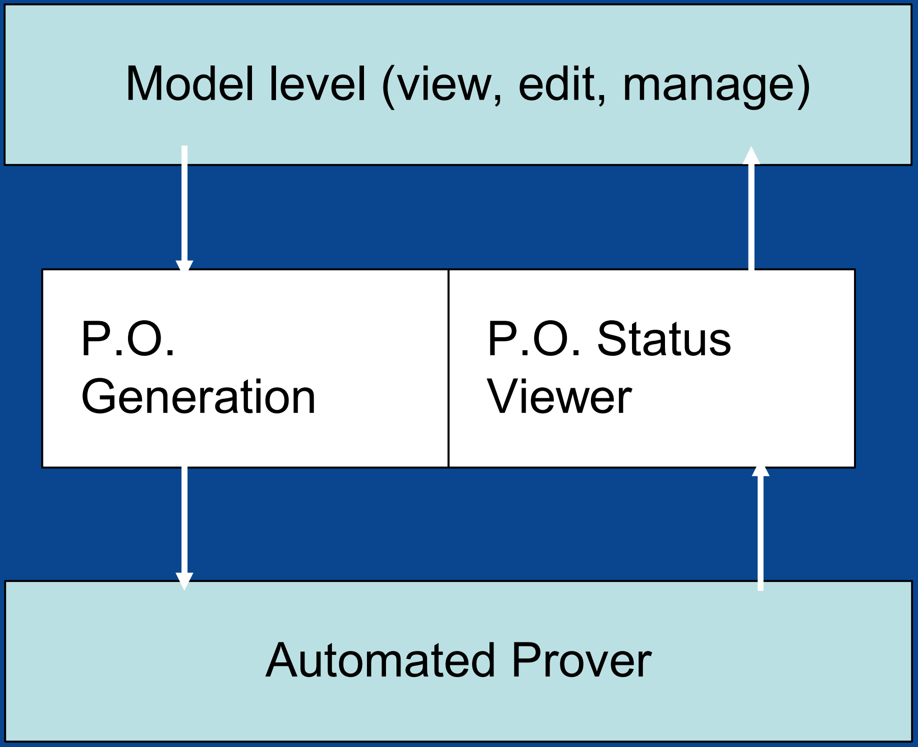
- Satisfaction proofs in terms of the language semantics are possible.
- … but complexity becomes intractable
- **Rely/Guarantee rules** trade off completeness for ease of use (do not expose internals)
- But must be proven sound wrt Language Semantics
- Actual use of R/G rules is similar to Hoare Rules
- Nice side-effect; semantic gap between prog & spec reduced
- So how do we take advantage of this?

Tools ☺

- Our tools experience in VDM is limited to the mural tools.
    - User guided proof and limited specification management
    - The mural core routines have been respecified in modern VDM (in VDMTools) and a implemented in Java. Aim is to develop a modern implementation of mural for fun.
- Recently done quite a bit on tools interoperability & Eclipse plug-ins in Rodin. Includes fine extensions to B toolset handling automated proof obligations.
- So, Tools for Proof remains a major interest. What could we do in Overture to promote this?

Tools ☺

High Automation Scenario (working for Proof Obligations)



Envision automated discharging of POs via a generation and management system.

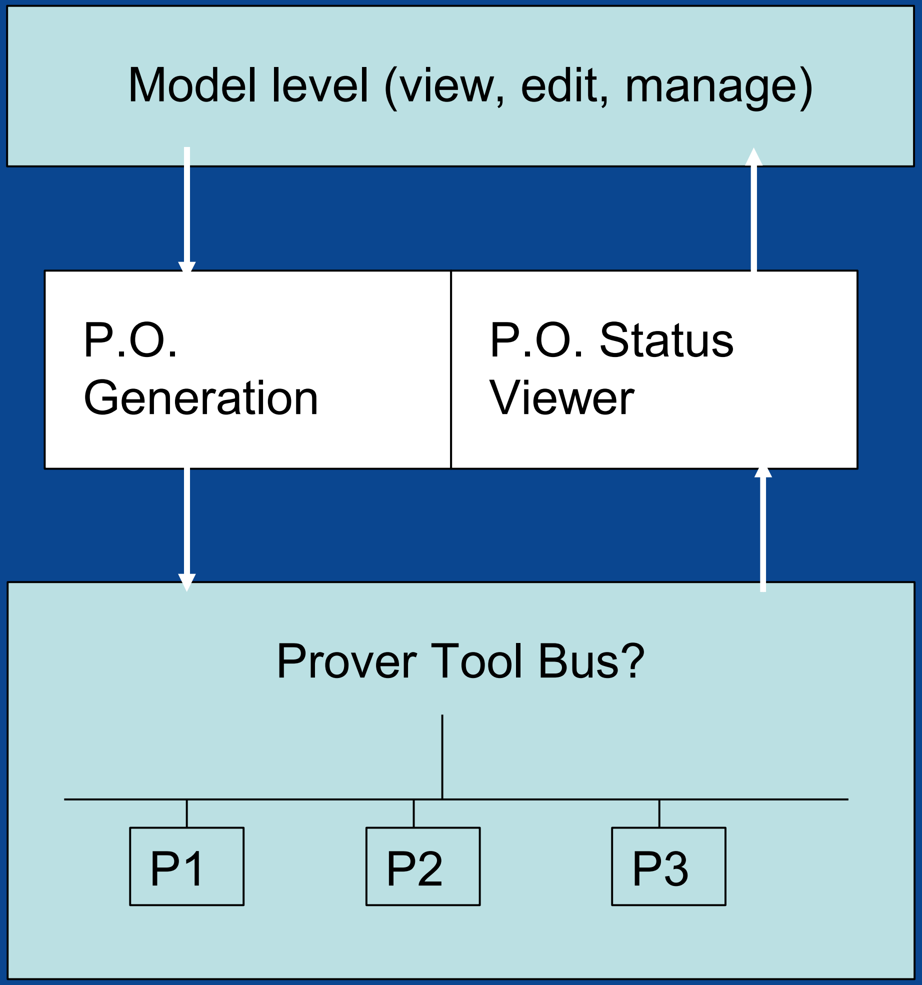
(c.f. Prosper Toolkit)

Need automated provers populated with theories consistent with model's semantics.

Need management tools for maintaining project status?

Tools ☺

High Automation Scenario (working for Proof Obligations)

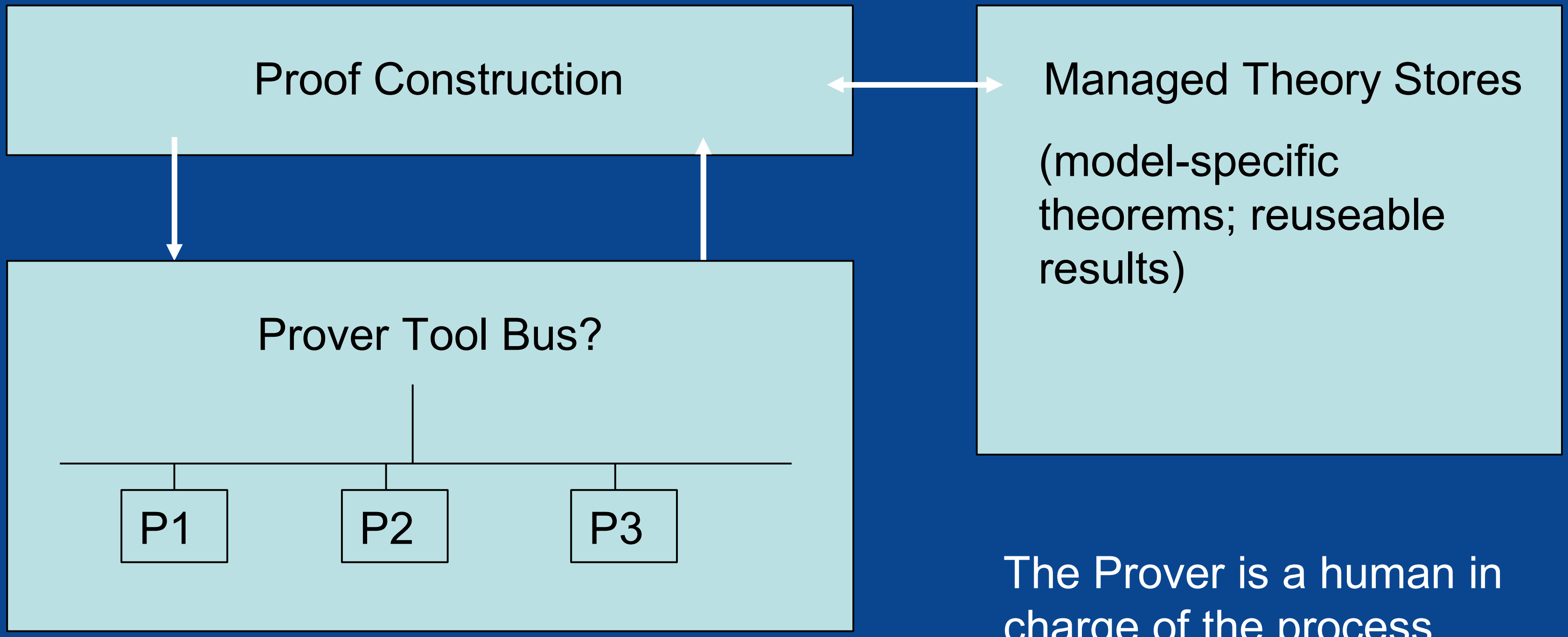


The tool bus concept is applicable (subject to semantic compatibility).

Implementing over a WS- architecture might mean stateless interaction with verification tools?

Tools ☺

Low Automation Scenario (Validation Conjectures and Exploratory Proof)



The Prover is a human in charge of the process, choosing to “accept” lemmas or send them to automated tools. Mural-like top level construction tool.

Tools ☺

- We have mentioned the need for animation and interface development support tools in the style of VDMTools.
- For proof, I want:
    - Theories of Typed LPF implemented in automated support systems
    - Tools managing the validation process (tracking discharged Pos etc.)
    - Lower-automation proof management, guidance tools using modern human interfaces and semantically well-defined interfaces with underlying proof tools.