# Extending and Strengthening the Tool Chain

Peter Linington

University of Kent at Canterbury

# Some Tools Issues

We are familiar with tools that manage engineering, and, to a lesser extent, computational specifications

What is the next step?

* What sorts of things can we do by transforming specifications?
* What kind of specification is needed for an enterprise?
* How can we develop suitable tools and evaluate them?

# Given an Enterprise Specification

If a design were to progress top-down, then the enterprise specification might be used

* to provide some sort of skeleton design from a list of roles
* to control and validate computational designs as they progress
* to control confirm that maintenance changes do not violate earlier policies
* to select middleware components (transparencies) and provide values for parameters.

# Given a Complete Design

If a complete, consistent set of specifications exists, changes to the enterprise specification could

* still change parameters and automatically rework parts of the design (particularly engineering)
  - but subject to need to check consistency is maintained
  - what about managing the transition? Can it be automated?
* flag parts of the specification that now violate policy
* provide initial cost estimates for proposed enterprise changes?

# The Re-working Problem

One of the problems with changes of an abstract specification is what to do about skeletons generated from it

* need to restructure detail based on the skeleton
* need to operate at a structural level - decorations on a syntax tree?
* examine anti-plagiarism tools?
* this is one of the problems which is facing the patterns movement

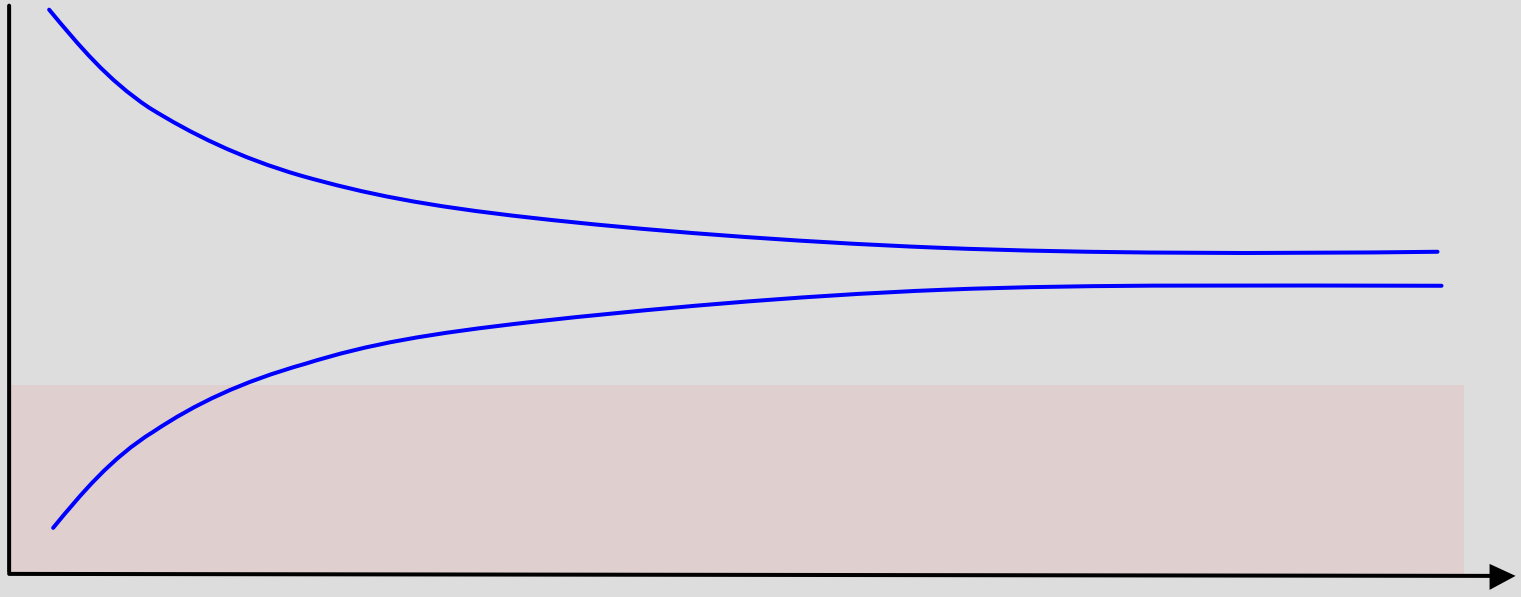
# Performance Prediction

UKC has been working with BT (and ERA) in the PERMABASE project, aimed at early estimation of the performance of distributed systems.

* Input from standard design tools
* support for what-if experiments
* aim to identify performance problems early, while designs are still at the outline stage
* reduce requirement for performance modelling expertise

# Accuracy of Prediction

The more information you have about a design, the more accurately you can predict performance, but the more it cost you to get there.



# PERMABASE(1)

User inputs are taken from

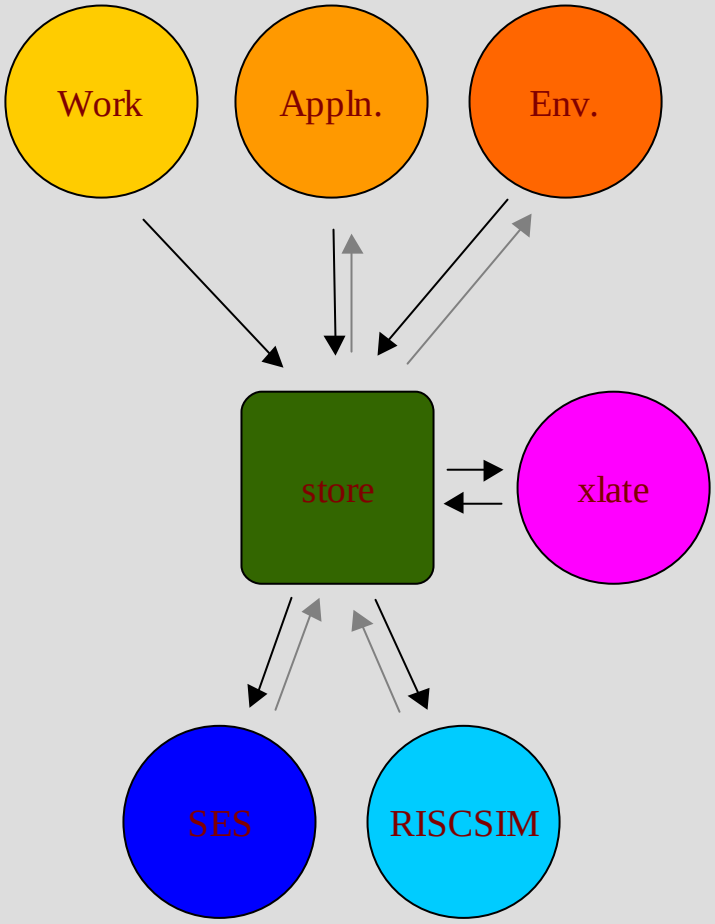
* requirements - yielding expected workloads and performance targets;
* application designs - in UML (Rational Rose);
* details of the environment - a BT network planning tool (configurator)

These inputs are processed by the tool to generate a unified data model, from which performance model can be built.

# PERMABASE (2)

## Processing steps:

* unify inputs
* check completeness
* apply defaults
* build performance model
* generate model definition for suitable engine
* translate results back to user terms



# PERMABASE (3)

Constructing models from UML

* focus on interaction diagrams
* add statechart information where available
* glean extra information from various constraints and decorations
* need good unifying algorithms

Even so, the designs are generally far from complete; even where there is notation for removing ambiguity, it is often not used.

# PERMABASE (4)

* Current prototype has used two discrete event simulators, with different levels of sophistication
* Hybrid techniques being investigated
* Have performed three case-studies
* Need a light-weight simulator to get reasonable response
* Direct feedback on parameter sensitivity is helpful
* Need your tools to be repository based and have open command interfaces

# Lessons for Tool Builders

* Specifications are still incomplete at the stage where important decisions are being taken
* need good defaults and heuristics to fill in the gaps
* these can be guided by knowledge of enterprise-wide standards and policies
* don’t get in the way of the design process - whisper hints, but don’t block progress
* reorganizing a design may take it through quite unrealistic intermediate stages

# Objectives

This paper fleshes-out parts of the ODP Enterprise Viewpoint Language

* It concentrates on the concepts of community and policy
* It shows the direction that current standardization activities should take
* It establishes a direction for detailed work on the way application design and middleware management can be made to reflect enterprise policy.

# The Enterprise Viewpoint

The ODP enterprise viewpoint is concerned with:

* the organizational framework within which an ODP system is to operate
* the general decisions which should guide its design and operation
* sufficient of the behaviour of the enterprise to support their definition.

An enterprise specification can be parameterized by the definition of policies.

# Communities and Roles

## A Community is:

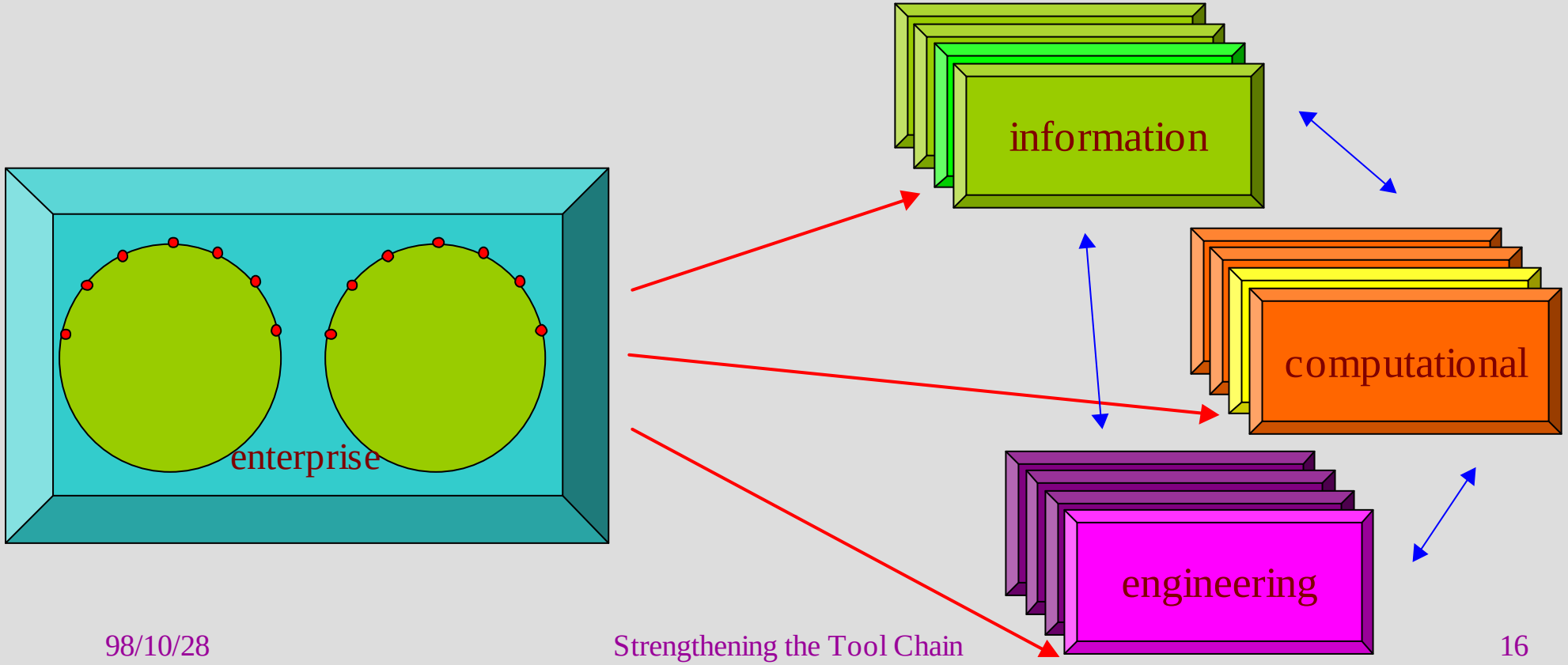
* a configuration of objects
* formed to meet an objective
* expressed in terms of a contract

## A Role is:

* a formal parameter of a community
* associated with required behaviour (a type)
* actualized by a suitable object

# Relation between Viewpoints

Enterprise policies may constrain the form of, or provide parameters for, other viewpoints.



# What a Contract Says

A community contract can specify:

* behaviour of the community as a whole; its obligations
* the roles in the community
* behaviour associated with each of the roles in the community
* constraints on the objects which can fulfil the roles in the community; e.g. filling by distinct objects
* authority claimed by the community

# Deontic Logic

Logic expressing the relation of permissions, prohibitions and obligations.

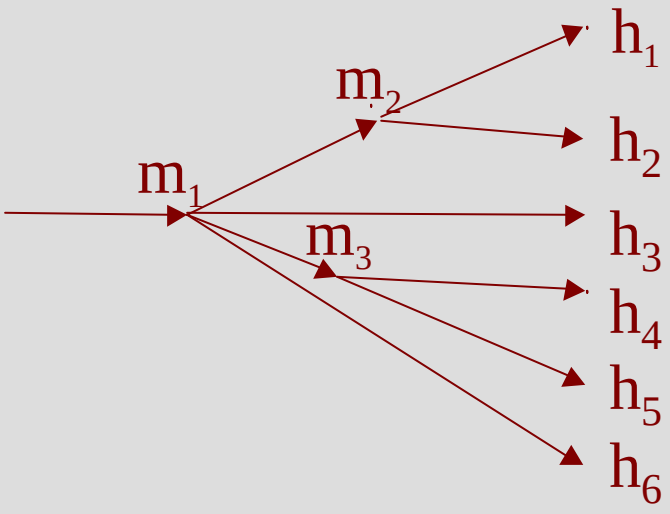
* a very active research area
* Standard Deontic Logic concentrates on the consistency of static situations
* deals with “ought to be”, rather than “ought to do”
* has difficulty with e.g. shared responsibilities.

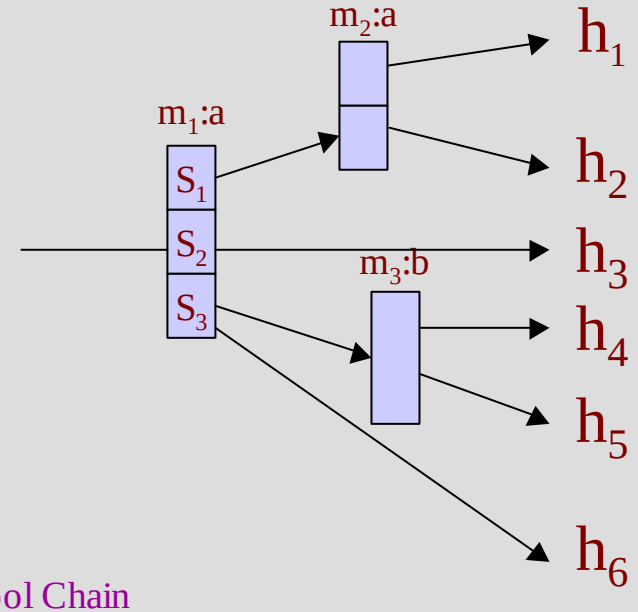
Need to enhance SDL with a notion of agency.

# Model of Obligation

Model based on the work of Belnap et al, using forward branching time

* moments represent decisions by specific agents
* series of moments is a history; history has a cost



# Enterprise Behaviour

An enterprise object can engage in a wide range of behaviour:

* it can make unpredictable choices;
* it can do things it ought not to do.

We distinguish:

* physical behaviour - the full range of things an object could, in principle, do
* social behaviour - behaviour constrained by the object itself as a result of its being in some community.

# Enterprise Actions

An enterprise action can be an interaction involving a number of roles in a contract.

An action happens if the community is:

* ready; all preconditions are satisfied
* willing; consistent with the social behaviour of all participants
* able; consistent with the physical behaviour of all participants

# Social Actions

The rules, objectives and policies of a community may be modified by social interactions within it.

* objects may negotiate changes to their obligations
* objects may modify their goals so as to change priorities
  - planning is based on incomplete information about the social state of others
  - each object may make estimates of the reliability of the objects in other roles
  - there will generally be at least some inconsistency.

# Forms of Policy

Policy can be expressed by

* permissions (weakening of obligations)
  - implies that, by default, some constraint exists
* prohibitions (strengthening of obligations)
  - implies that, by default, behaviour was unconstrained
* obligations; which have:
  - defining behaviour
  - enabling conditions (predicate or start/end conditions)
  - satisfaction or violation conditions (may dispose of obligation)

# Permissions

There are two senses of permission:

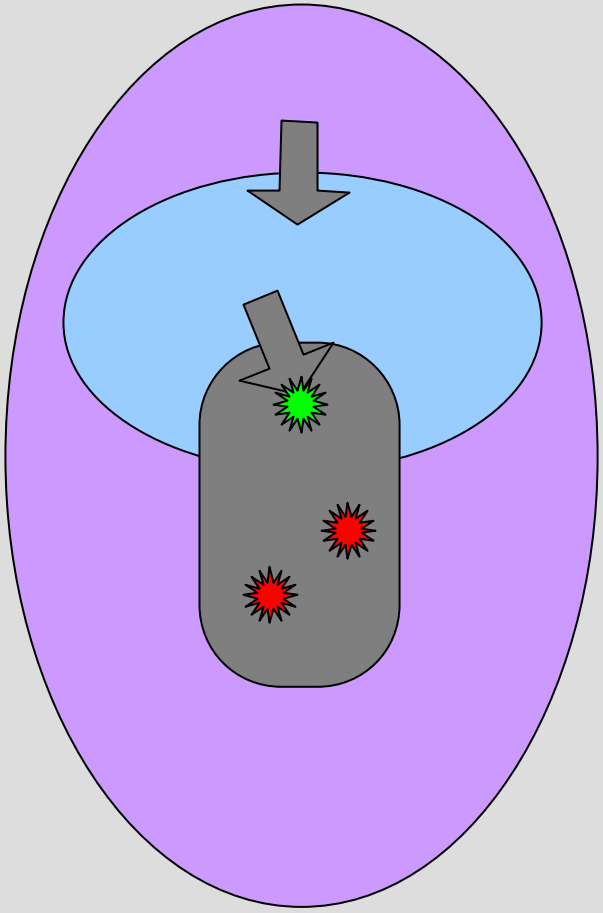
* having a permission; describes the social state of the object, in terms of what it may choose to do if it wishes
* granting a permission; an interaction of two objects
  - results in the grantee having the permission;
  - only valid if the grantor is a proper authority for the permission
  - places an obligation on the grantor not to frustrate the permission granted

The second meaning extends a mesh of obligations.

# Nesting of Communities

A communities has context provided by surrounding or overlapping outer communities.

* these provide the basis for its contract;
  - when no obligation is placed, the constraints outside apply
  - new obligations must not conflict;
* an outer community may delegate authority to an inner one



# Enforcing Policy

There are two styles:

* pessimistic enforcement:
  - involves checking of permissions for every action by some trusted agent
  - requires active prompting for obligations
  - used if trust is low and costs of violation high
* optimistic enforcement:
  - relies on objects controlling their social behaviour
  - backed up by auditing and a system of penalties
  - used if trust is high and cost of violation low

# Conclusions

A framework for definition of policy is a vital part of the Enterprise Language. It must define

* how communities express policy
* how the behaviour of objects is modified when they undertake roles in communities
* how obligations and authorities are inherited from outer or overlapping communities

It can be used to steer information or computational designs and configure middleware engineering.

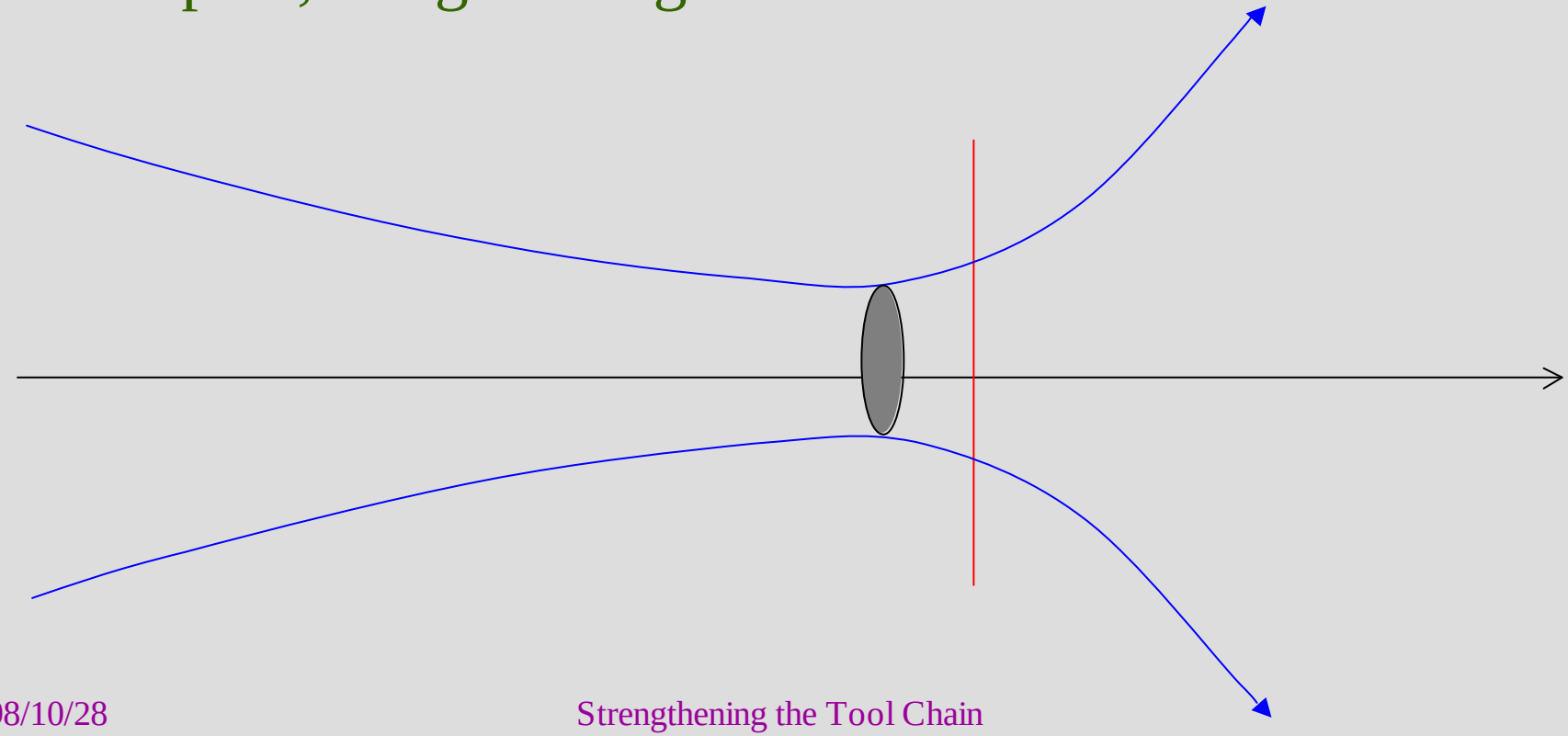
# Using the Enterprise Spec.

How can the information in the enterprise specification be used?

* to select from a library of known solutions
  - transparencies or protocols
  - at design or system-build time
* to select applicable tactics
  - feed into adaptive components
* to guide fine-scale game playing
  - run-time guidance direct from a model of obligations

# A Component's Knowledge

* as information arrives, a component's knowledge of its peers' recent state increases; further into the future or the past, things diverge



# Playing the Game

* use information about the state of peers, and their physical and recent social behaviour to build a tree of options
* use policies as weights to assess the costs of the various courses of action
* use the strongest policies first, and prune the tree as soon as possible
* possible simple test cases; policy based:
  - data transfer
  - transactional systems

# Conclusions

* lengthening the tool chain and linking its components together promises greater automation at each stage;
* having explicit enterprise information allows better and earlier checks on the design process
* the enterprise information might be applicable to short time-scale decision, not just to the design stages, making the chain from enterprise policy to operations complete
* the environment for integrating tools can be provided by repositories
* better framework for enterprise specifications is needed