Towards a logical framework for tightly coupled monitoring and evolution of software components

Howard Barringer

in collaboration with Dov Gabbay, David Rydeheard

School of Computer Science
University of Manchester

email: howard.barringer@manchester.ac.uk

5th January 2007

Health Warning

Logicians at work!

What follows may seem rather

abstract, simplistic and too far from reality.

Please bear with us as we attempt to develop foundations!

Outline

Motivation

- Evolvable systems

Motivation

- Evolvable systems
- Software architecture based on Evolver-Producer pairs

Motivation

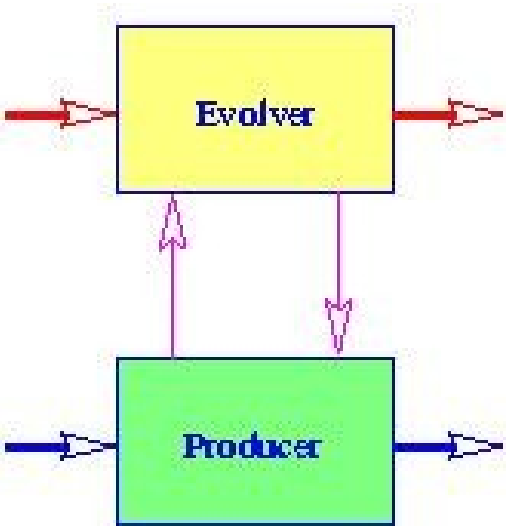
- Evolvable systems
- Software architecture based on Evolver-Producer pairs
- Software component-based systems

Motivation

- Evolvable systems
- Software architecture based on Evolver-Producer pairs
- Software component-based systems
- Logical modelling and reasoning

Evolver-Producer Pairs

Warboys et al. introduced these as basic building blocks.



The evolver process may stop, decompose, modify, recompose and restart the producer process

Hierarchical assemblies of such pairings — few constraints on connectivity.

Outline

Imagine . . .

. . . a blocks world theory in a typed 1st order logic.

Use an observation predicate

$$on : Block \times (Block \cup Table)$$

Specify an action, say,

$$Move : Block \times (Block \cup Table) \times (Block \cup Table)$$

as a revision (e.g. as in STRIPS) on observation states (i.e. a set of ground atoms)

Use theory axioms/constraints to characterise size of tables/blocks, etc., as well as properties of observation predicates. Abstraction predicates can be introduced, e.g.

$$free : Table$$

Move action specification

$Move(x : Block, y, z : (Block \cup Table))$	
pre	$\{on(x, z), free(x), free(y), x \neq y\}$
add	$\{on(x, y)\}$
del	$\{on(x, z)\}$

Now imagine ...

. . . another theory meta to the blocks world, also defined in a typed 1st order logic.

Use observation predicates

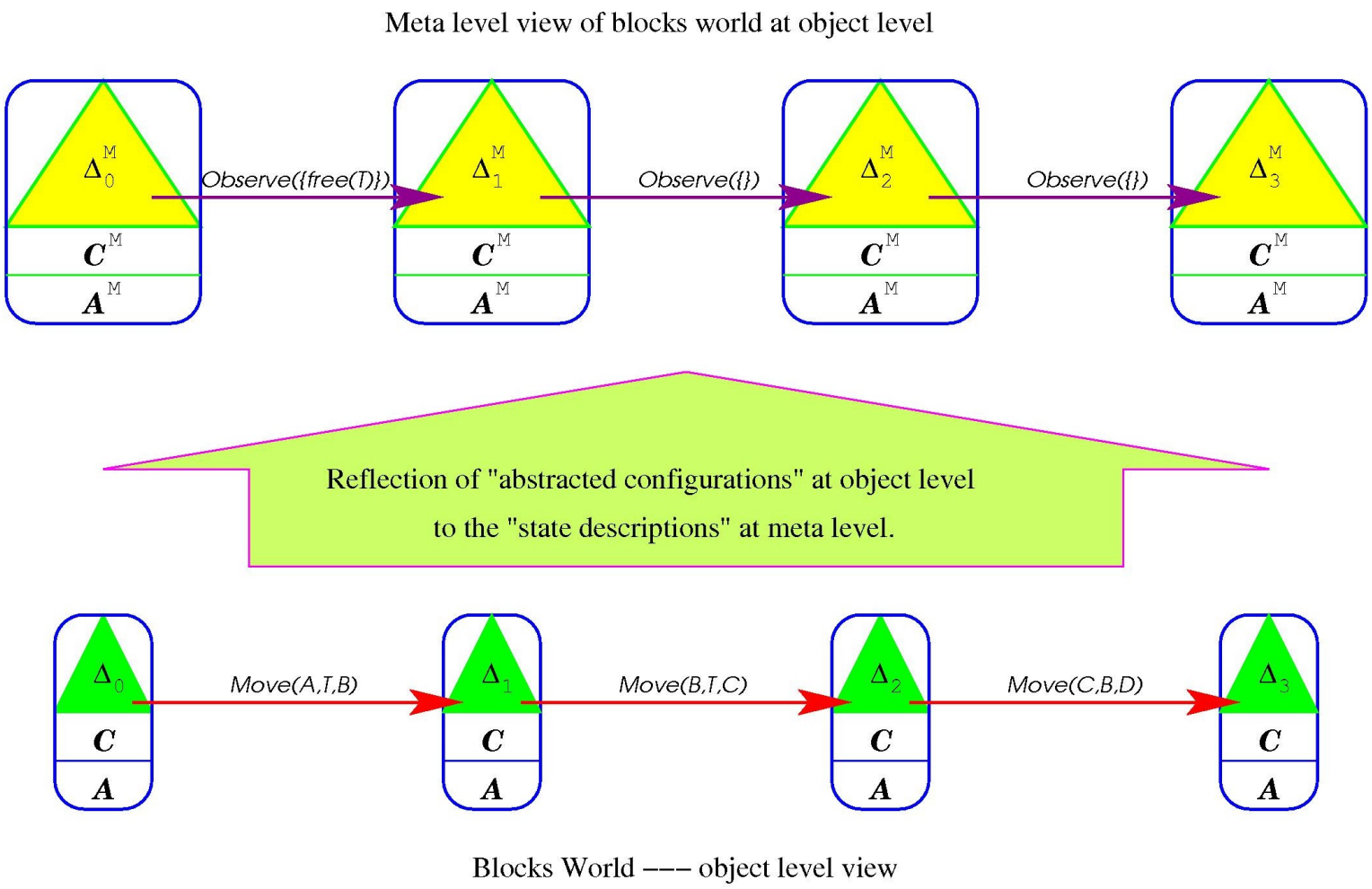
$$\begin{aligned} &\mathit{holds} : \textsc{Formula} \times \mathit{ConfigurationName} \\ &\mathit{current} : \mathit{ConfigurationName} \\ &\mathit{constraint} : \textsc{ConstraintName} \end{aligned}$$

Specify an action

$$\mathit{Observe} : \textsc{Formulae} \times \mathit{ConfigurationName}$$

to revise the observation state by the addition of formulas given as argument.

Observing blocks world moves



State Meta View

Formally characterises the relationship between

meta level state observations

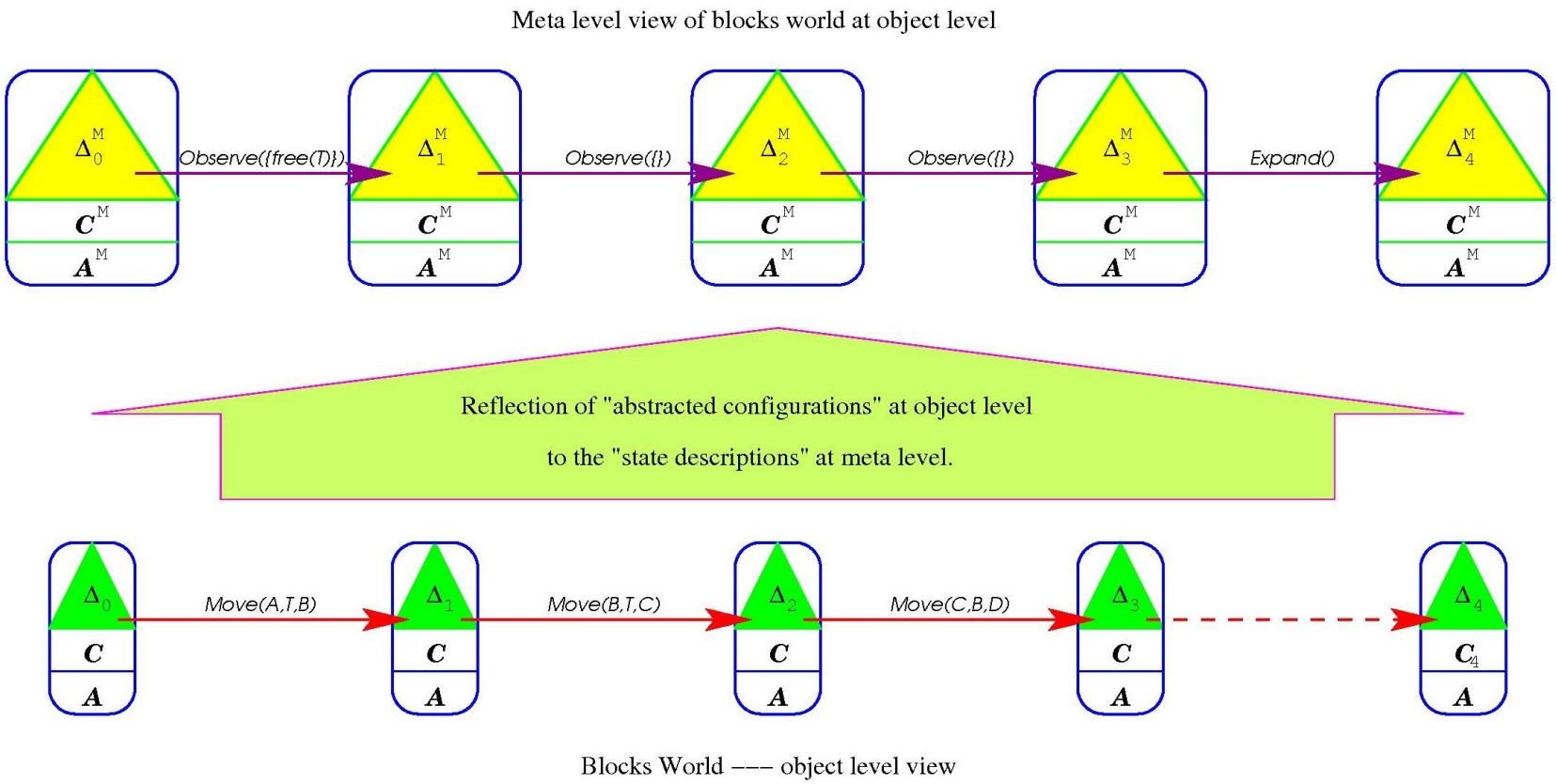
and

object level configuration

through the use of the *holds*predicate.

E.g. if $holds(on(A, T), c_0)$ exists in meta-level state observation then $on(A, T)$ must exist in observation state of object-level configuration corresponding to c_0.

Evolving the blocks world



Transition Meta View

Formally characterises the relationship between *PAIRS* of

meta level state observations

and

object level configurations

through the use of an *evolve* predicate.

Table expansion — from the meta level

In the meta-level theory, specify an action:-

$Expand(m, n, c)$	
pre	$\{current(c), constraint(TableSize(T, m))\}$
add	$\{current(s(c)), holds(free(T), s(c)),$ $constraint(TableSize(T, n))\}$
del	$\{current(c), constraint(TableSize(T, m))\}$

Key points so far

- A configuration describes a logical execution state together with its associated theory instance

Key points so far

- A configuration describes a logical execution state together with its associated theory instance
- Meta-level and object-level systems are specified in the same way

Key points so far

- A configuration describes a logical execution state together with its associated theory instance
- Meta-level and object-level systems are specified in the same way
- Meta-level and object-level systems are viewed as being executed in lock step synchrony

Key points so far

- A configuration describes a logical execution state together with its associated theory instance
- Meta-level and object-level systems are specified in the same way
- Meta-level and object-level systems are viewed as being executed in lock step synchrony
- The meta-level systemhas the capability to view anything of the object-levelconfiguration

Key points so far

- A configuration describes a logical execution state together with its associated theory instance
- Meta-level and object-level systems are specified in the same way
- Meta-level and object-level systems are viewed as being executed in lock step synchrony
- The meta-level systemhas the capability to view anything of the object-levelconfiguration
- Actions are specified as being logically atomic via a revision process.

Key points so far

- A configuration describes a logical execution state together with its associated theory instance
- Meta-level and object-level systems are specified in the same way
- Meta-level and object-level systems are viewed as being executed in lock step synchrony
- The meta-level systemhas the capability to view anything of the object-levelconfiguration
- Actions are specified as being logically atomic via a revision process.
- At the meta-level, actions may observe, or induce change at the object-level.

Outline

Component Theories

- Treat a component instance as an instance of a particular logical theory

Component Theories

- Treat a component instance as an instance of a particular logical theory
- Have extended the above approach to hierarchical assemblies of component instances

Component Theories

- Treat a component instance as an instance of a particular logical theory
- Have extended the above approach to hierarchical assemblies of component instances
- Use a simple unique naming mechanism to enable straightforward composition of logical theories

Component Theories

- Treat a component instance as an instance of a particular logical theory
- Have extended the above approach to hierarchical assemblies of component instances
- Use a simple unique naming mechanism to enable straightforward composition of logical theories
- Communication between component instances modelled by joint actions

A Buffer theory example

$BUFFER(N : Int)$

OBSERVATION PREDICATES

$content : Value\text{-}list$

ABSTRACTION PREDICATES

$free$

CONSTRAINTS

$$Uniqueness \stackrel{dfn}{=} \forall l_1, l_2 : Value\text{-}list \cdot content(l_1) \wedge content(l_2) \Rightarrow l_1 = l_2$$

$$Size(M : Int \text{ INITIALLY } N) \stackrel{dfn}{=} (\exists l : Value\text{-}list \cdot content(l) \wedge (|l| < M)) \Leftrightarrow free \wedge \forall l : Value\text{-}list \cdot content(l) \Rightarrow (|l| \leq M)$$

ACTIONS

$Send(v)$	
pre	$\{content(l :: v)\}$
add	$\{content(l)\}$
del	$\{content(l :: v)\}$

$Receive(v)$	
pre	$\{free, content(l)\}$
add	$\{content(v :: l)\}$
del	$\{content(l)\}$

A coupled buffer

New logical theories can be constructed through combining instances of other theories.

BUFFERING

COMPONENTS

$B_1, B_2 : BUFFER(2)$

ACTIONS

$Send(v) \stackrel{dfn}{=} B_2.Send(v)$

$Receive(v) \stackrel{dfn}{=} B_1.Receive(v)$

$Internal \stackrel{dfn}{=} B_1.Send(v) || B_2.Receive(v)$

The specified actions are the only actions that an instance of a BUFFERING theory has.

Combining Meta-level and Object-level theories

- A meta-level theory is specified relative to an object-level theory

Combining Meta-level and Object-level theories

- A meta-level theory is specified relative to an object-level theory
- The combination of a meta and associated object level theory is special

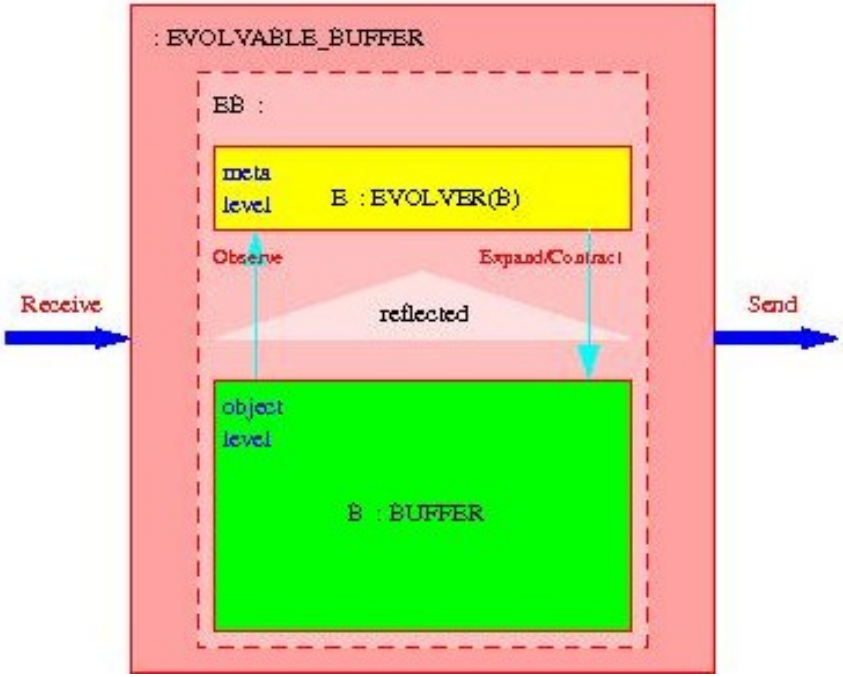
Combining Meta-level and Object-level theories

- A meta-level theory is specified relative to an object-level theory
- The combination of a meta and associated object level theory is special
- It requires identifying how actions can be coupled

Combining Meta-level and Object-level theories

- A meta-level theory is specified relative to an object-level theory
- The combination of a meta and associated object level theory is special
- It requires identifying how actions can be coupled
- Meta-level actions that specify object level theory change occur in isolation

Pictorially ...



As a schema ...

EVOLVABLE_BUFFER

COMPONENTS

$EB : (E : BUFFER\_EVOLVER \text{ META TO } B : BUFFER(2))$

ACTIONS

$Send(v) \stackrel{dfn}{=} EB.\langle E.Observe(Q), B.Send(v)\rangle$

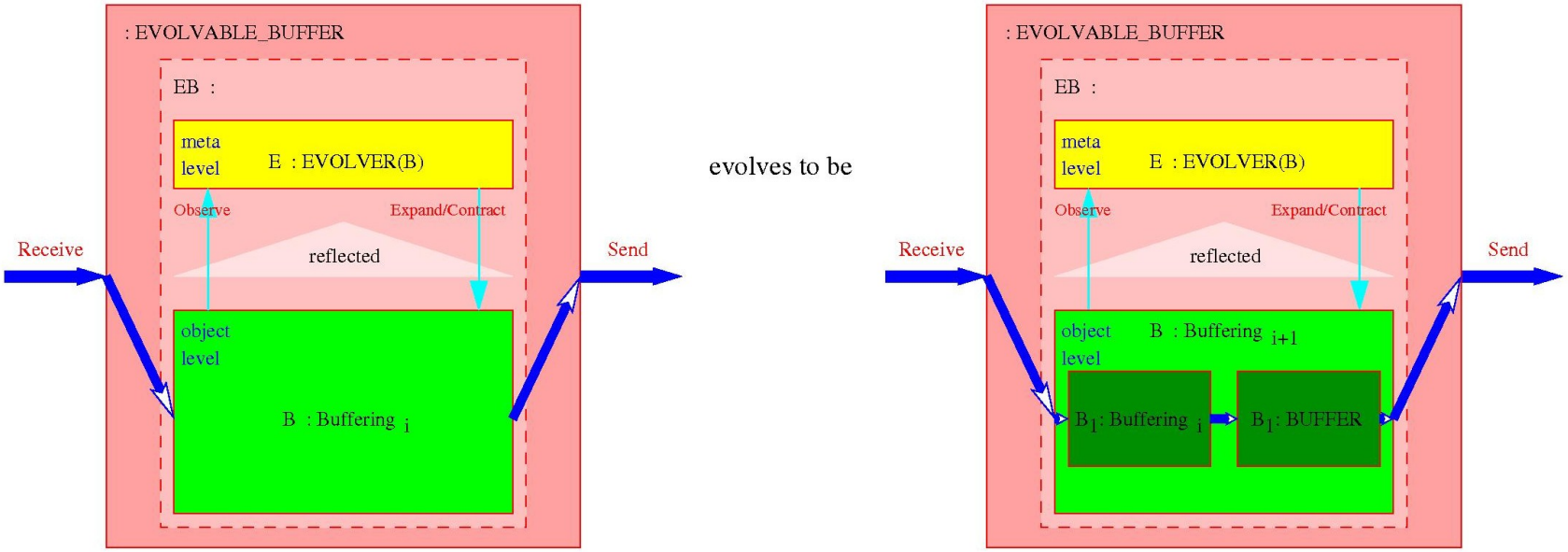
$Receive(v) \stackrel{dfn}{=} EB.\langle E.Observe(Q), B.Receive(v)\rangle$

$Internal \stackrel{dfn}{=} EB.\langle E.Expand, \rangle$

EB names an instance of the special combination of meta and object-level component theories

The actions of the *EVOLVABLE_BUFFER* component are specified as combinations of the meta and object component actions.

A more structural object-level change



For components, so far ...

- We have a logical framework for supporting the specification of hierarchical assemblies of supervised components

For components, so far ...

- We have a logical framework for supporting the specification of hierarchical assemblies of supervised components
- Configuration structures for such assemblies are fairly complex

For components, so far ...

- We have a logical framework for supporting the specification of hierarchical assemblies of supervised components
- Configuration structures for such assemblies are fairly complex
- Revision is no longer a simple process of augmenting and/or restricting sets of formulae — complex tree manipulations may be required.

For components, so far ...

- We have a logical framework for supporting the specification of hierarchical assemblies of supervised components
- Configuration structures for such assemblies are fairly complex
- Revision is no longer a simple process of augmenting and/or restricting sets of formulae — complex tree manipulations may be required.
- Standard revision logics consider only simple revisions — considerable work to be done on proof theory

For components, so far ...

- We have a logical framework for supporting the specification of hierarchical assemblies of supervised components
- Configuration structures for such assemblies are fairly complex
- Revision is no longer a simple process of augmenting and/or restricting sets of formulae — complex tree manipulations may be required.
- Standard revision logics consider only simple revisions — considerable work to be done on proof theory
- Components communicate *vertically* through evolutionary processes

For components, so far ...

- We have a logical framework for supporting the specification of hierarchical assemblies of supervised components
- Configuration structures for such assemblies are fairly complex
- Revision is no longer a simple process of augmenting and/or restricting sets of formulae — complex tree manipulations may be required.
- Standard revision logics consider only simple revisions — considerable work to be done on proof theory
- Components communicate *vertically* through evolutionary processes
- and *horizontally* through joint actions defined as disjoint revisions on the subcomponent states

For components, so far ...

- We have a logical framework for supporting the specification of hierarchical assemblies of supervised components
- Configuration structures for such assemblies are fairly complex
- Revision is no longer a simple process of augmenting and/or restricting sets of formulae — complex tree manipulations may be required.
- Standard revision logics consider only simple revisions — considerable work to be done on proof theory
- Components communicate *vertically* through evolutionary processes
- and *horizontally* through joint actions defined as disjoint revisions on the subcomponent states
- The interaction of component hierarchies with evolutionary behaviour appear to yield a powerful methodology for the development of adaptive and evolvable software systems.

Outline

Encapsulating programs

- Where and how do programs over the actions of a component fit in?

Encapsulating programs

- Where and how do programs over the actions of a component fit in?
- Have been experimenting with the view that components are active and come equipped with a single main program?

Encapsulating programs

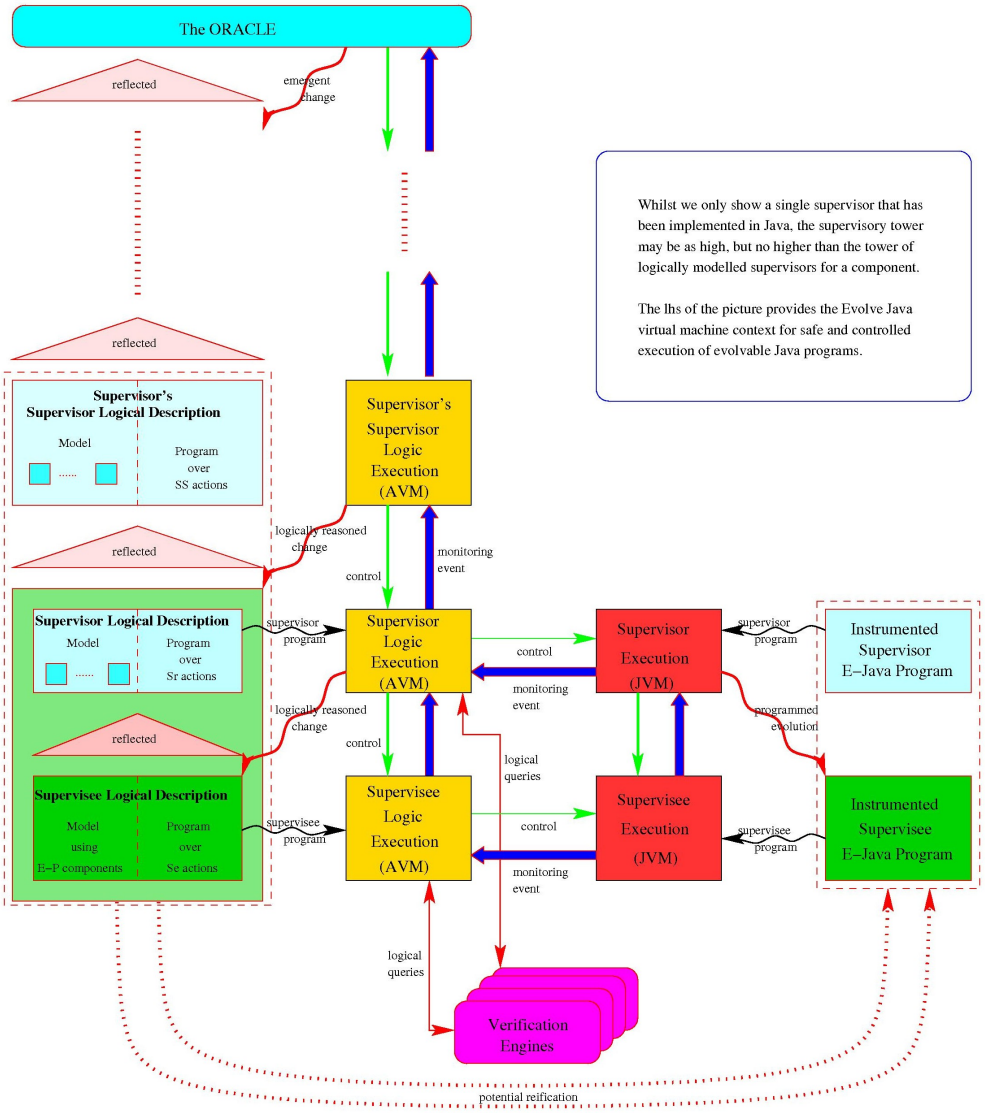
- Where and how do programs over the actions of a component fit in?
- Have been experimenting with the view that components are active and come equipped with a single main program?
- Components may have different associated languages!

Encapsulating programs

- Where and how do programs over the actions of a component fit in?
- Have been experimenting with the view that components are active and come equipped with a single main program?
- Components may have different associated languages!
- Meta-level components may thus not only change programs, but also programming language of associated object-level systems.

Outline

An implementation aim!!!!



Whilst we only show a single supervisor that has been implemented in Java, the supervisory tower may be as high, but no higher than the tower of logically modelled supervisors for a component.

The lhs of the picture provides the Evolve Java virtual machine context for safe and controlled execution of evolvable Java programs.

There is an evolving page

http://www.cs.manchester.ac.uk/evolve

Thank you for your patience