System Development at Run Time

Dr. Christopher Landauer, Dr. Kirstie Bellman
Topcy House Consulting
topcycal@gmail.com, bellmanhome@yahoo.com

Models@Run.Time Workshop (after)
29 September 2015

Outline

Space System Context: Why Do We Want Reflection?

Space systems are the most complex systems humans build that work
- Hundreds of organizations
- Thousands of people
- Millions of components
- Development and sustainment process can last for decades
- Resulting systems are expected to last for years to decades
  . with no planned hardware changes in space
  . with occasional complete replacement of ground hardware and software
  . responsive software changes to accommodate hardware degradation

There is no development methodology that can effectively address this scale
of problem up front; errors will occur

For sufficiently remote, complex or hazardous environments
- Communication delays with system precludes hands on problem solving
- Operators must rely on processes internal to the system

The system needs continual access to all development and operational aspects
- Monitoring behavior and state, Reporting to developers / users, Analyzing
- To catch problems as soon as they occur
- To predict problems before they occur

What Is Reflection?

System reasoning about itself and its own behavior
- Observing that behavior, creating and adapting models and other knowledge
- For our applications, all parts of the system are subject to this kind of reflection

Interpreting and analyzing knowledge embodied in the models
- Mapping the external and internal worlds into its internal context,
  so the data can be used to adapt and guide its own behavior
  (according to the goals or purposes provided or selected)
- Analysis, evaluation, arbitration, selection

Mapping the decisions into action internally or in the world
- None of the rest matters if the system can’t do anything about it
  . Asking for help is doing something
  . Being able to say ‘‘I do not know what to do’’ is essential

Leads to a need for activity loops to close the processing cycle explicitly
- Data through decisions to changes
- Assistance from processes that have internal knowledge that users do not have
- Models of all parts of the system that may need to change
- Coordination of internal assessment activities with external
  . Adaptation based on internal or external performance criteria

How Do We Implement Reflection?

Wrappings

- Knowledge-Based Integration Infrastructure for Reflective Systems
  . Explicit context management

- Problem Posing Programming Paradigm PPPP
  . Separate problems from resources

- Wrapping Knowledge Bases WKB
  . Machine-interpretable mappings from problems to resource uses in context
  . These are models of appropriate use for all resources

- Problem Managers PM coordinate the use of resources
  Coordination Manager CM
  Study Manager SM
  Activity loops

Problem Posing

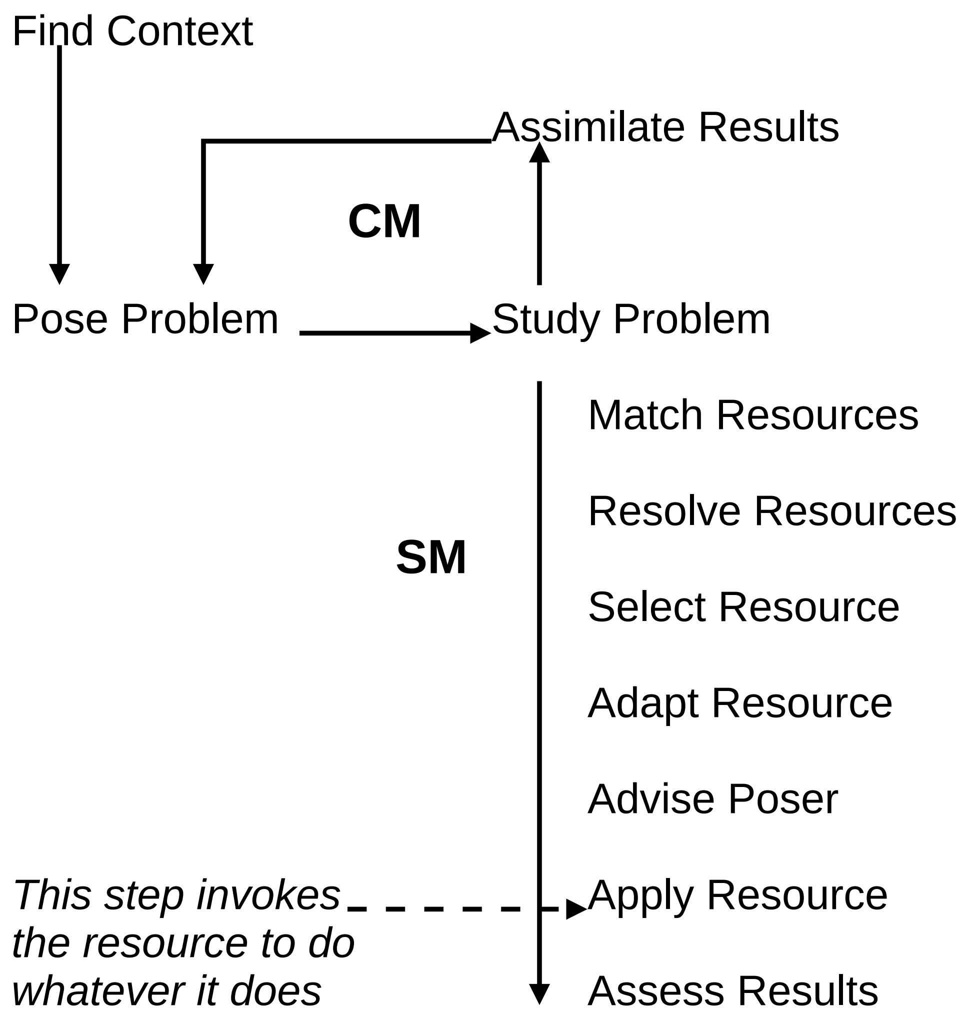
The map is not the territory (plato)

The model is not the system
- This separation is important for more effective modeling
  - . Develop multiple model hypotheses
  - . Compare multiple model behaviors
  - . Evaluate model effectiveness in context
  - . Use different models for different analyses
  - . Use different models for different situations

The problem is not the resource (problem posing)
- All information service requests are called *Problems*
  - . Posed problems replace function calls and message sends
- All information service providers are called *Resources*
  - . Resources replace function and method definitions

This separation allows tremendous flexibility in the reconnection processes
- System organized as a large number of computational resources
- Resources are selected to apply to problems in context
  - . Based on knowledge in the Wrapping Knowledge Bases (WKBs)

Default Activity Loop



Process (Activity Loops)

Activity loops are used to organize system tasks
- Read, Eval, Print (LISP)
- OODA = Observe, Orient, Decide, Act (DoD, Boyd)
- MAPE = Monitor, Analyse, Plan, Execute (Autonomic, Kephardt)
- ELF = Elementary Loop of Functioning (AI, Meystel)

All can be seen as instances of CM / SM
- Most activity loops are not intended to be used meta-recursively
  - . They are directed outward towards control tasks
- The CM / SM are directed inward towards any and all resource use
  - . Relegate all actual work to resources
  - . This means they can be used meta-recursively (on themselves)

Activity loops include other kinds of main control
- Any main program style corresponds to a CM
  - . We described the single loop version in the default CM
  - . There is also a multi-agent distributed version in a Mud CM
  - . There is also a multi-agent swarm version

Meta-Meta- = Meta-

There is only one level of resources, some of which act on others,
some of which act on themselves, all based on provided Wrappings

This choice prevents an infinite tower of meta-processes,
but it means we have to address self-reference carefully

Every resource is managed by an SM, including all of the PMs themselves,
so every resource has at least one other that can control its selection and application

This is what we mean by completely reflective (we don’t often do that, but we can)

- How much reflection we use is an engineering judgment
  about what needs to be flexible and what can be fixed
- Almost always more needs to be flexible than you expect

All control activity is the same process, run by the PMs

- Fit the resource use to the context and problem, according to the WKB
- Apply it to the problem

Reflective Modeling

It is trivial to add new models and other resources in a Wrapping-based system
- Models and all other resources are selected at run time:
  - . Add new WKB entries for new resource uses
  - . Replacements or customized special cases
  - . Different problems, different contexts use different Wrappings
- It automatically becomes available for selection in the proper context
- We might want the system to performs some acceptability analyses
  - . Assumptions, parameter settings, compatibility assessments

Our systems
- Need no inherent or a priori knowledge of the problem set
  - . That is defined entirely by what problems are posed
- Build the space within which they operate, as defined by their models of
  - . The operational environment (inferred from observation)
  - . Their own behavior (specified and inferred from observation)
  - . The concept of operations
    (specified rules about how the system will be used / commanded)
- Adjust it according to observed environment behavior
  - . Using goals, deficiencies, real-time measurements (DDDAS)
- Newly added problems, new resources also adjust operational space

Reflective Modeling

Self-modeling systems interpret run time models of behavior to generate that behavior
- These systems do not USE models @ run.time
- These systems ARE models @ run.time

One kind of inherent adaptation
- System builds behavior models via grammatical inference
  - . Inference of other complex event structures
- These observation models live in an event trajectory space
  - . But we expect that they occupy a very small fraction of it
- Manifold discovery identifies constraints on the observed trajectories
  - . Leads to simplified behavior models that can be compared to originals
    (for special cases, diagnostics, anomalies, etc.)
  - . May lead to simpler notational expressions for behavior
    (only express what changes, leave what does not as context)

We expect our systems to have many more resources than most applications use
- Dozens to hundreds is minimal, thousands more likely)
- It is very hard to create, refine and analyze models without access
  to behavior observations or simulations or external knowledge

We therefore want the designers and operators and the system to better support
each other in recognizing issues and providing resource solutions

How Do We Use Reflection?

To support system co-development by human designers / users and system
- From early in development through deployment and even run time

Sources of issues
- Human finds deficiency in expected behavior
- System detects divergence from expected behavior

Humans often provide resources to address the issue
If not, then system must
- Generate hypotheses
- Design experiments
  . User-provided resources shortcut these two steps,
    but the rest are needed for compatibility checking
    (to make sure the resources address the issue effectively)
- Run simulations on appropriate selected scenarios
- Evaluate results and assess improvements, if any
- Update resource, if acceptable

This is a seemingly different activity loop, but it is also an instance of the CM loop,
a kind of meta-activity loop that organizes the resources that manage activities
(for problems like ‘‘add new resource’’, ‘‘evaluate resource’’, ‘‘add new Wrapping’’, etc.)

Conclusions and Prospects

There are many research issues remaining

Computational semiotics
- Study of the use of representational mechanisms in computing systems
- The ‘‘Get Stuck’’ Theorems imply that symbol systems must be refactored
    . If systems act over long times in a complex environment
- How that refactoring should or can be done is difficult
    . Manifold discovery on event trajectories is promising
    . Develop usage-appropriate notation
    . Adjust all object and process representations

Self-reference
- Go:del’s theorem is not really relevant (we are not looking for completeness)
- We want to discover sets of mathematical conditions under which self-reference
    can be considered, or even properly defined
- We are looking for criteria for acceptable use
    . Analogous to the many convergence theorems for infinite series
        (‘‘If you retain this property, things will be OK’’)
        (‘‘If you stay away from this problem, things will be OK’’)
- Circular coinductive rewriting (Joseph Goguen) shows that circular sets
    of definitions can define abstract structures uniquely
    (by analogy with but more general than Context-Free Grammars)

Appendix

Wrapping Details

Application: Reuse Without Modification
- Function Call Diversion

Application: Migration To Common Standards
- Separation, Integration
. Transitioning Legacy Systems

Self-Modeling Systems

Hard Modeling Issues: Objects and Events

Wrapping Details

ALL About Wrappings

Components of Wrappings

Data (Wrapping Knowledge Bases)

Process (Problem Managers)

Default Problem Manager

Multiple Resources from Same Problem

ALL About Wrappings

ALL parts of a system are **resources**
- Resources are computation or information service providers
- Everything that does anything is a resource

ALL activities in a system are **problem** study
- The application of resources to posed problems in context
- Posed Problems are computation or information service requests

The Problem Posing Interpretation is this separation of problems and resources
- It can be applied to any programming language
- It greatly improves the flexibility of behavior of programs
- It leads to a number of interesting applications
    . Reuse without modification
    . Migration to common standards
    . Self-modeling systems
    . Generic programming
    . Delaying language semantics to run-time

ALL About Wrappings (continued)

ALL maintenance of system state is done with **context**
- Invocation environment provides initial context
- System operation updates dynamic context from internal and external sources
- Context is essential for mapping problems to appropriate resources

ALL information connecting problems to resources is in **Wrapping Knowledge Bases**
- WKBs define mapping of posed problems to applicable resources in context
- Defined by developers and provided with the resources
- A Wrapping defines a use of a resource for a problem in context
- One resource may have many Wrappings for the same or different problems

ALL interpretation and performance activities are managed by **Problem Managers**
- Problem Managers are also resources, and are also Wrapped
    . Computational Reflection
- Coordination Manager is the basic cyclic program heartbeat
- Study Manager is a planner that organizes the resource applications

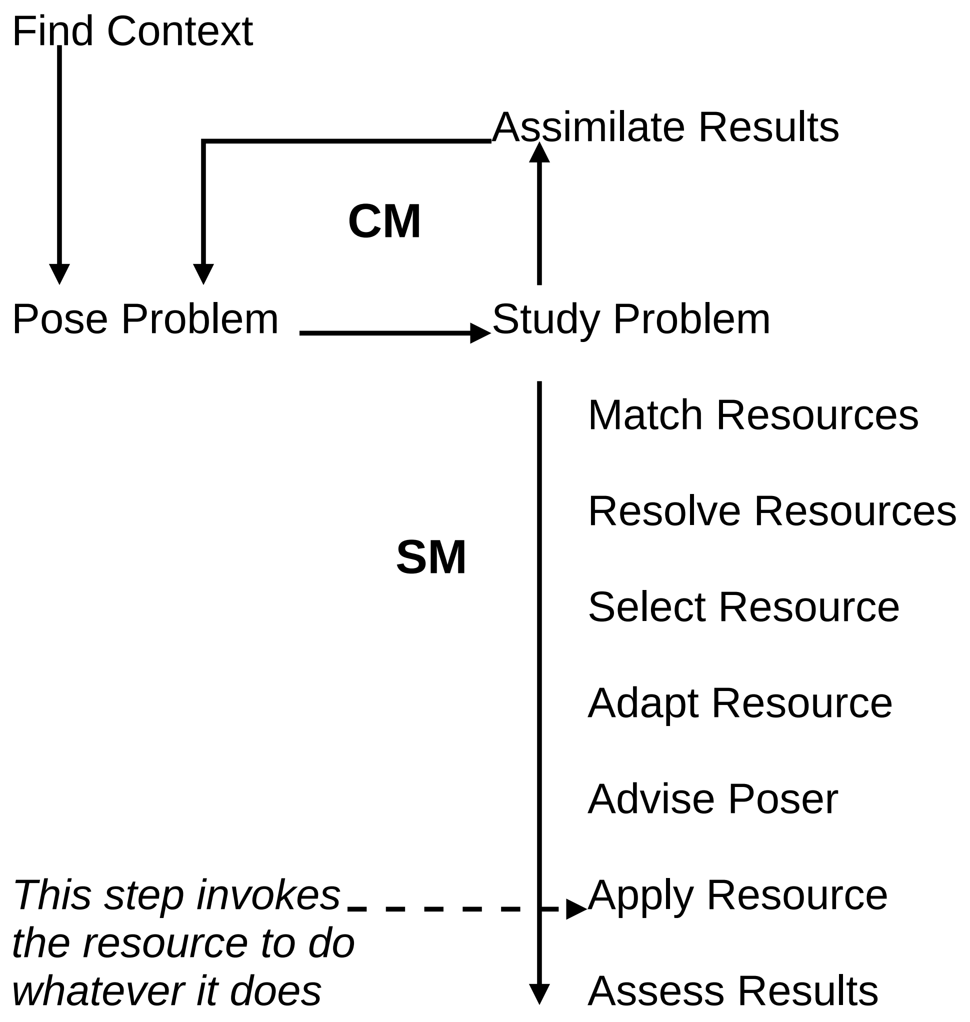
Process (Problem Managers)

The processes that use Wrappings are called Problem Managers PMs
- There are two distinguished ones: CM and SM

```
Coordination Manager CM
    Find (initial) Context
    loop
        Pose Problem
        Study Problem
        Assimilate Results

Study Manager SM
    Match Resources
    Resolve Resources
    Select Resource
    Adapt Resource
    Advise Poser
    Apply Resource
    Assess Result
```

Default Problem Managers



Multiple Resources from Same Problem

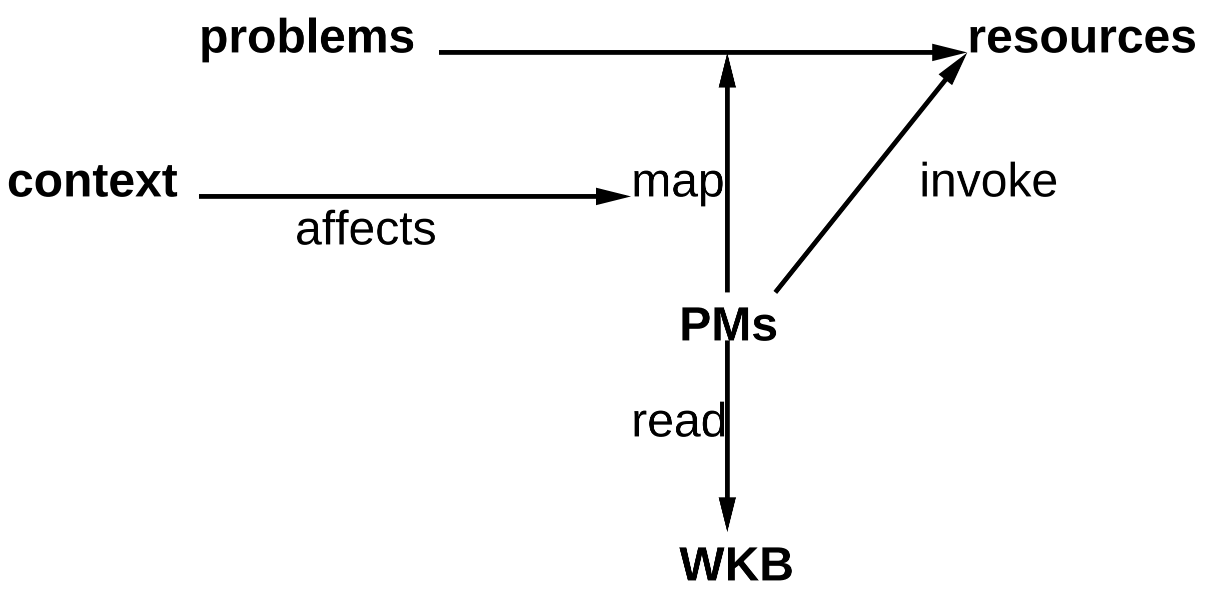
Wrapping processes map problems to resources in context using a Knowledge Base
- That means that different contexts can lead to different resources

Exactly the same code can be used in different ways
- One version of the resource uses behavior assertions in a formal model and includes timing and other observations for a simulation
- Another has the basic machine code with additional timing for a simulation
- Yet another has just the basic machine code to insert into an execution image

This lets us examine and adjust the timing properties of the system as we build it
- And as we learn more about the execution environment

It also allows us to delay the problem semantics to run time
- Via selection of different resources using different contexts

Components of Wrappings



Data (Wrapping Knowledge Bases)

The system connects problems to resources via mappings called Wrappings

A Wrapping is a map from a problems in context to a particular use of a resource

- When it is appropriate to use this resource
  - . Context and problem parameter constraints
- How to use the resource
  - . Parameter binding
- Where to find the resource
  - . Object code map

Designers have to answer all of these questions anyway

- Wrappings record the answers in a machine-interpretable way

Application: Reuse Without Modification

Start with legacy code that is still needed (or at least wanted)
- Remember, tomorrow you will start writing more legacy code

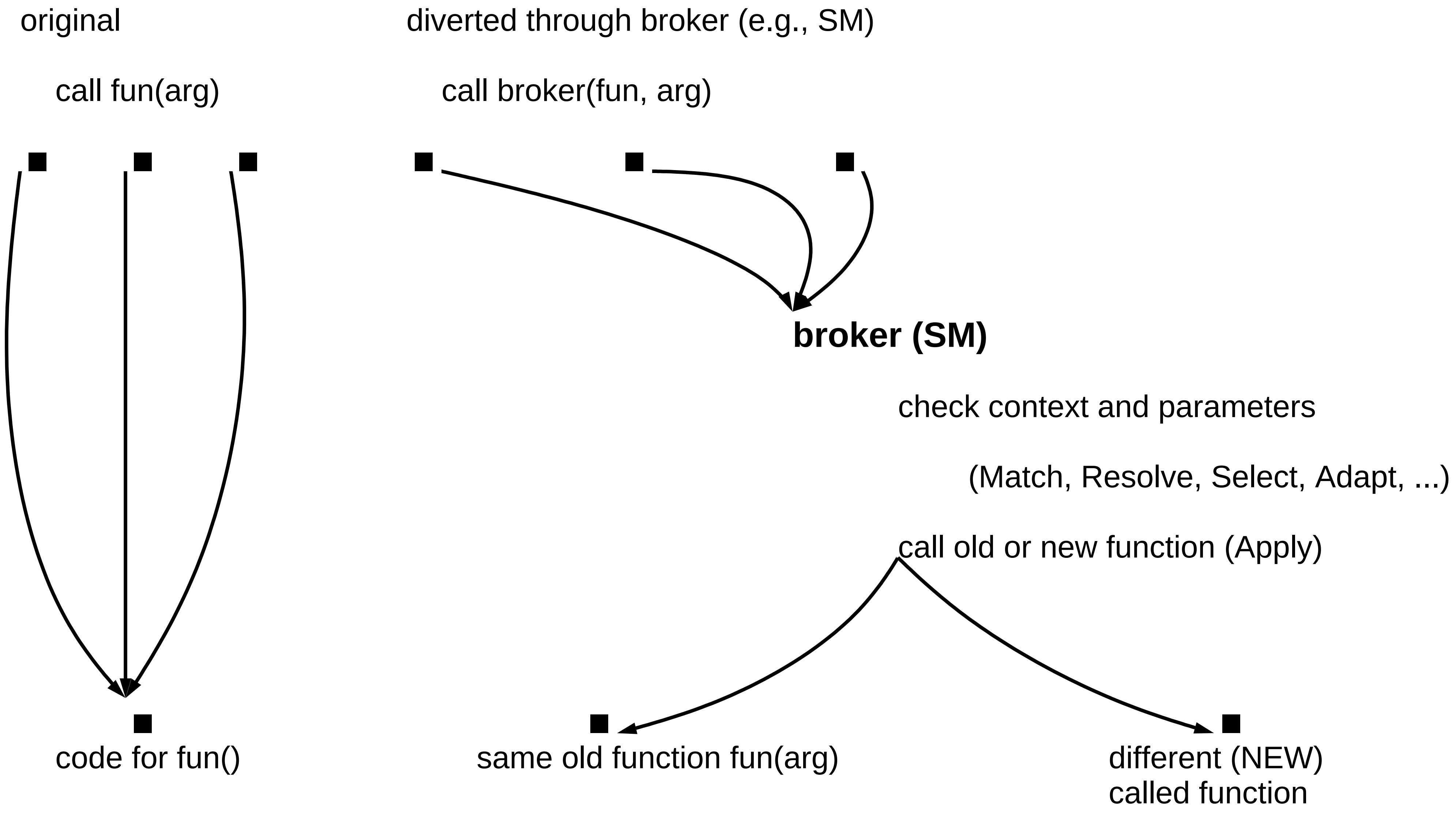
Software Disintegration to identify points at which to introduce flexibility
- Engineering judgment

Write a new compiler to intercept the function calls and call an SM instead
- With the specified function call as a problem to be studied
- Writing compilers is now relatively easy (master’s thesis level)

Define Wrappings to divert the function calls to other resources
- Can also map to old function as before, in the appropriate context
- Can even introduce time delays to maintain same timing characteristics

Code reuse with no changes to the code at all
- Old code is always available, even when new code partially overlaps in function
- Gradual weaning from old code until it is not needed any more

Function Call Diversion


original
call fun(arg)
code for fun()
diverted through broker (e.g., SM)
call broker(fun, arg)
broker (SM)
check context and parameters
(Match, Resolve, Select, Adapt, ...)
call old or new function (Apply)
same old function fun(arg)
different (NEW)
called function

Application: Migration to Common Standards

Start with multiple interactive systems for similar functions

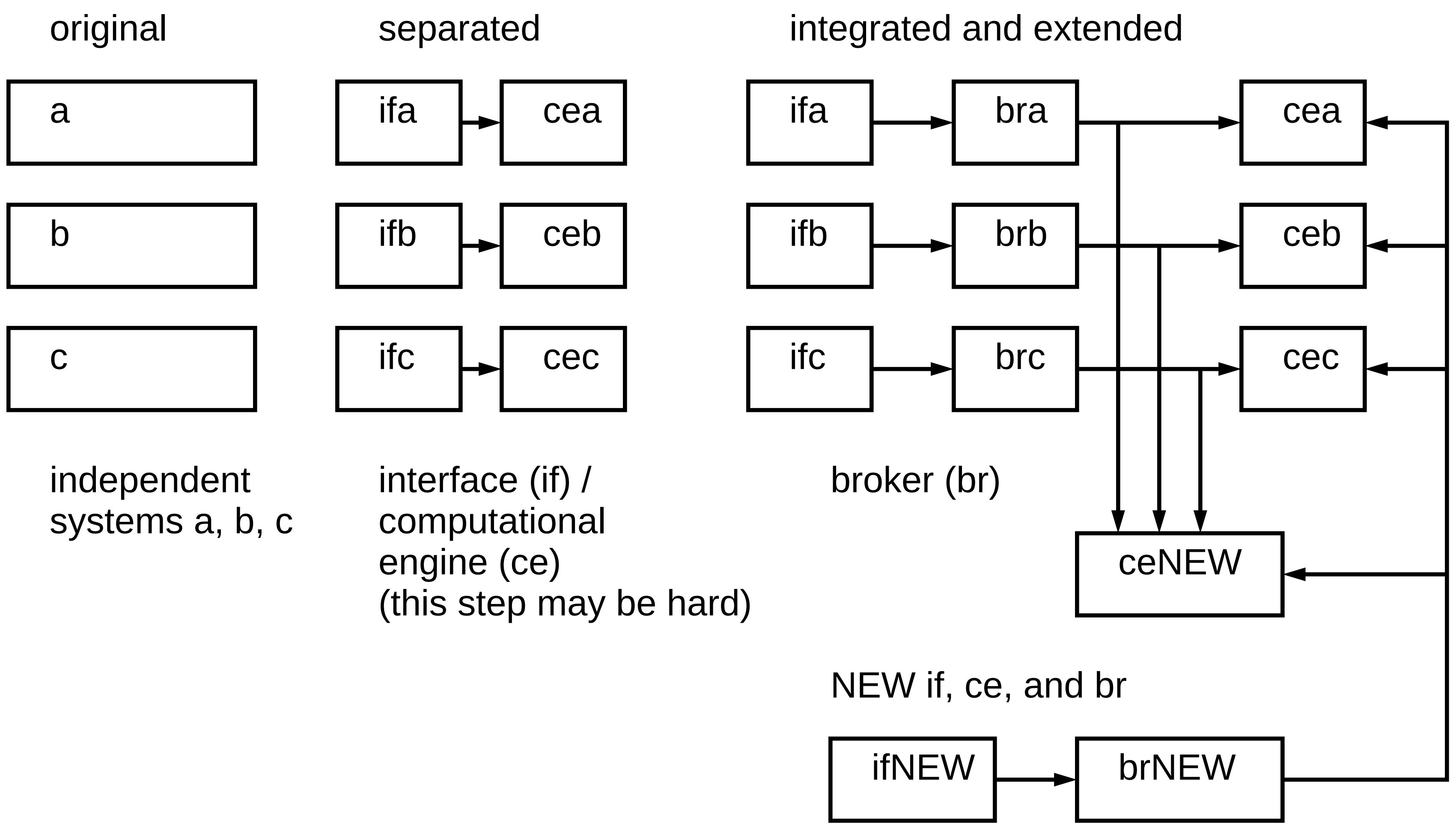
Software Disintegration to identify internal interfaces
- Separate user actions from computational responses

Define Wrappings to divert the function calls to other resources
- Can also map to old function as before, in the appropriate context
- Can even introduce time delays to maintain same timing characteristics

Users of old interfaces can continue to use them
- Old data remains in the existing legacy databases
- Old applications remain in the existing legacy processes
- New data and applications go into the new standard database and process
- Wrappings determine which to use

New standardized interface also accesses both old and new data
- Old interfaces are used until they are no longer needed
- Old databases are used until they are no longer needed

Separation, Integration



Transitioning Legacy Systems

Software Disintegration

Choose instrumentation points
- Where to change
  . Anywhere choices of algorithm or control may change
- What to change
  . Computations to function calls
  . Function calls to posed problems

Create resources and Wrappings
- How to change
  . Invent problem and resource names and parameter descriptions
  . Split out many code fragments as resources
  . Define conditions under which they should be used
    (context and problem parameter)
  . Write Wrappings for the resources

Self-Modeling System

Processes and descriptions

- Processes are everything that happens (active resources)
- Descriptions are models of the processes
    - . Every process has at least one description sufficiently detailed to be executed
- Descriptions can be models of other constraints and relationships
- All description notations need interpreters or translators, which are also processes

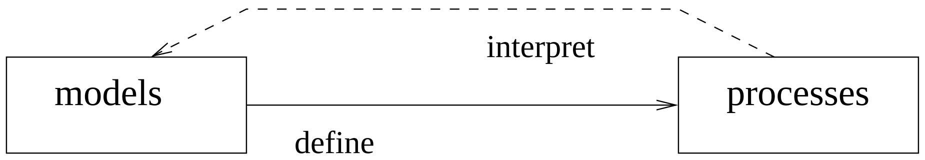
Architecture

- Wrappings describe the use of all resources
- PMs use those descriptions to organize and apply combinations of resources
- Descriptions of all processes and their uses
    - . Active resources are processes; passive ones are descriptions
    - . Dictionaries map resource names (symbols as string data) to program addresses (symbols in object files)
- Rich set of notations and notational fragments to produce these descriptions
    - . Tuple templates for Knowledge Bases
    - . Compilers and translators for all of the notations
    - . Integrators that define the role of fragments

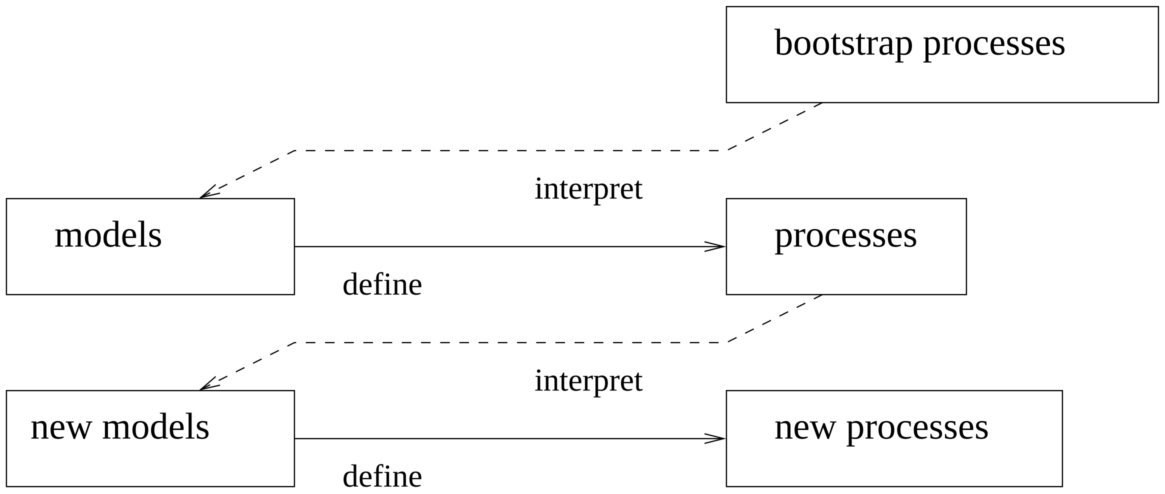
Need at least one bootstrap program to start the process

- To compile process description interpreters into executable processes

Self-Modeling System



Starting a Self-Modeling System



Hard Modeling Issues: Objects and Events

Finally, perhaps the hardest problem of all

What are objects and events in the real world?
- We know that they are learned concepts (even in humans)
  . Not inherent or instinctive

What is an object?
- How do we identify boundaries?
- How do we identify persistence?
- How do we identify identity?

What is an event?
- How do we decide beginning and end?
- How do we separate from concurrent activities?

There are some deep issues about perception and representation in these questions