Systems of Systems and Statically Defined Dynamic Architecture Evolution

Robert Watson, Sutirtha Bhattacharyya
Dewayne E Perry
Center for Advanced Research in Software Engineering (ARiSE)
The University of Texas at Austin
Introduction

→ Context: NASA Manned Space exploration
  ➔ Earthbound control center no longer feasible
  ➔ Will need control center and other systems onboard
  ➔ Will need to dynamically evolve the systems of systems

→ Systems of components
  ➔ Compositions of components
  ➔ Component interaction
  ➔ Component and system/architectural evolution
  ➔ Typically single thread of execution

→ Systems of Systems
  ➔ Compositions of Systems of components
  ➔ Systems interactions
  ➔ Systems of Systems and architectural evolution
Research Background

→ Simulation language and System
  ➡ NASA specific, but sufficient to task
  ➡ Governed by a very flat architecture/design description
    ➢ For components
    ➢ For interactions
    ➢ For topology
    ➢ For scheduling
  ➡ Automatically generates simulation system and schedules
  ➡ Provides execution and visualization environment

→ Goals of research
  ➡ Reverse engineer existing simulations to create architecture models
    ➢ SDP - an analysis tool for reverse-engineering existing flat simulation descriptions to provide
      ✓ Relationship descriptions
      ✓ Visualizations of the concrete architecture of the simulation
  ➡ Create architecture model and support to create simulations via architecture descriptions
    ➢ Archpad - a graphical architecture modeling system
      ✓ Tailor to creating simulation architecture models
      ✓ Basis for generating simulations
Abstract Architecture Model

→ Model consists of three abstract constructs
  ➜ Arch-element
    ➢ A component (data or processing) or a connector
  ➜ Arch-composition
    ➢ Represents the substructure of an arch-element
  ➜ Arch-region
    ➢ An arbitrary collection of arch elements
    ➢ Arch-regions may overlap, contain or be contained in other arch-regions

→ Arch-element is the basic architecture component
  ➜ Arch-element =
    ( name,
      {service-specifications},
      {general-constraints},
      {dependency-specifications}
    )

→ General constraints apply to the arch-element as a whole
Abstract Architecture Model

→ An arch-composition is a set of elements together with mappings as to how they relate to each other

Arch-composition
( name,
{arch-elements},
{mappings}
)

Mappings accomplish several things

- Map an sub-arch-element service-specification to the arch-element service specification
  ✓ ie, indicate which service specifications are used to satisfy the arch-elements interface
- Map internal satisfaction of dependency-specifications to their associated service-specifications
- Map unsatisfied service-specifications to the arch-element interface specification
- Map general-constraint satisfaction
- Map unsatisfied general-constraints to the arch-element interface
Elements in the Simulation World

→ Basic and composite elements
  â May be both depending on use in a particular architecture configuration
  ➢ In one simulations may be treated as a basic component (eg, the Crew Exploration Vehicle - CEV)
  ➢ In another it may be that we need to consider its constituent component (CEV as stage 1 rockets and astronaut capsule, eg)

→ Basic architecture elements are physical objects
  â Such as the CEV or earth

→ Basic elements are active or passive
   Eg, the CEV is active, earth passive
   Passive elements often contexts for active elements
   Passive elements often sources of constraining influences on active objects (as the earth is on the CEV, eg)
Elements in the Simulation World

→ A product line style-like organization
  - Commonality: architecture components used in a variety of simulations
  - Variability: architecture components specifically for certain simulations

→ Schedules are critical for simulations
  - Envisioned as a general-constraint
  - Also for real-time systems
  - Schedules for various levels of the simulations
    - Micro-level schedules for individual components
    - Macro-level schedules coordinating multiple components
    - Over all schedules governing sequencing phases of a simulation

→ Motivation for the notions of configurations
  - Specific physical events when the “world” changes
    - Eg, failures, transitioning from earth to space
    - Eg, de-coupling the rocket stage from the capsule
    - Eg, docking at the space station
  - May need to represent sequences, trees or graphs of events
Specializations of AAM

→ Architecture Transition Connectors
  ➜ Define the interactions between architectures
  ➜ Governs the transition of control and data between architectures

→ Graphs of architectures represent a (projected) history of the simulation
  ➜ Sequences represent sequences of events
  ➜ Trees represent sequences of events that include choices
  ➜ Graphs represent sequences of events with choices/merges

→ An architecture of architectures graph (AAG) represents a complete simulation
  ➜ Arch-archs-graph =
    ( name,
      {arch-configurations}
      {thread-bindings}
      {schedules}
    )
Specializations of AAM

→ The thread bindings of an ACG
  ➤ Tie individual AC threads together across the architectural configuration graph
    ➤ Some threads stop executing
    ➤ Some threads continue
    ➤ Some threads start up
  ➤ Defines the actual execution of threads where the AC threads bindings merely define the potential threads in a configuration

→ Schedules in an ACG
  ➤ Define when the ACs begin and end

→ Execution semantics assumptions wrt data:
  ➤ All data is “current” with in a thread
  ➤ No “own” data
  ➤ Shared data between threads is “read only”
  ➤ If want writeable “global” data, need critical sections
Architecture-of-Architectures

→ Problem domain: NASA M.E. simulators
  ✇ Exhibit a varying architecture as simulated vehicles reconfigure in-flight
  ✇ Each architecture describes the simulator and simulated system over an interval of time
  ✇ Architectures share common sub-architectural elements
  ✇ An architecture-of-architectures approach allows common elements to be defined once
  ✇ Changes to one element propagate to all architectures
Dynamic Architectural Change

- Examples of architectural change from Apollo and Shuttle
Relationships Among Sub-architectures

→ Most sub-architectures are the product of a physical transformation of an existing architecture
  ➔ Differences tend to be incremental derivations
  ➔ Substantial redundancy exists among sub-architectures

→ Long duration missions will exhibit many sub-architectures requiring considerable effort:
  ➔ In development of sub-architectures
  ➔ In maintenance of sub-architectures
Architectural Transitions

→ To avoid development and maintenance of highly redundant sub-architectures we propose connectors among sub-architectures: architectural transitions
   ✤ Transitions describe how one sub-architecture differs from another
   ✤ Descriptions can be minimal—they describe one temporal change exhibited by a vehicle in flight

→ Transitions are reusable—they can be applied to more than one source architecture
Architectural Transitions

→ Architectural transitions reduce redundancy
  ⚫ Potentially, only the initial vehicle configuration has a full sub-architectural description
  ⚫ Other sub-architectures are derived by applying transitions to the initial and derived architectures

→ Example:
  ⚫ Initial configuration, vehicle on-pad (pre-launch)
  ⚫ A transition describes differences from post-launch configuration
  ⚫ Another transitions describes changes incurred by stage 1 booster separation
Example Architecture Graph

onpad
  ↓ launch
    ↓
stage1
  ↓ LAS_abort
  ↓ stage1_separation
  ↓ LAS_abort
    ↓
launch_abort
  ↓ LAS_separation
    ↓
launch_abort.LAS_sep
  ↓
reentry
    ↓ landed
  ↓
LEO
  ↓ SM_jettison
  ↓ station_dock
  ↓ station_undock
  ↓
docked
Elements of a Transition

→ Transition predicate and effector function

→ Predicate:
  ✷ Selects architectures valid for application of the transition
  ✷ Iff the predicate of transition t holds for some sub-architecture c, then there is another sub-architecture c' defined by the application of the effector function of t to c.
Elements of a Transition

→ **Effector function:**
  - Defines a sub-architecture as a variation on an existing sub-architecture
  - Captures only the differences between a source architecture and a derived architecture
  - May not be idempotent

→ *A single transition may apply to more than one source architecture*
  - Increases transition complexity but reduces redundant specification
  - Example: the launch abort transition can be initiated from multiple vehicle configurations
Implementation

→ Implemented with an architectural meta-language
→ Currently utilizes a procedural description
→ Meta-language will allow non-procedural descriptions
→ Currently, predicate and effector computations are not separated
Example Transition

→ **conf**: architecture to be transformed
→ **rename**: Provides a name for derived
→ **return**: Provides a predicate value. Predicate holds if true
→ **replace**: Carries out a transformation on conf

```plaintext
transition stage1_separation {
    global var conf;
    conf = rename(conf, stage2);
    if (!has_component(conf, fullstack)) return(false);
    if (has_constraint(conf, onpad)) return(false);
    conf = replace(fullstack, {...stack.stage_two, ..stage1});
    return (true);
}
```
Summary

→ Began with our abstract architecture model
   ➲ Useful for modeling architecture elements in simulations
   ➲ Schedules for individual architecture elements describable as constraints on the elements

→ Initial extensions to model needed
   ➲ Differentiation of data, processing and connecting elements?
   ➲ Further development of connecting elements beyond typical use

→ To model complex simulations where physical changes take place, propose the ideas of architecture configurations and configuration graphs
   ➲ Notions of locus of control, threads
   ➲ Higher levels of scheduling
   ➲ Binding and rebinding of data
   ➲ Binding of threads to actual execution threads