Autonomous, Self-Adaptive Software: Architecture-based Tools, Techniques, and Methods

John Georgas, (Eric Dashofy)
Institute for Software Research
University of California, Irvine
May 6, 2004
Outline

- **Software Dynamism**
- Software Architecture
- Architecture-Based Approach
  - Evolution Management
  - Adaptation Management
- Summary
What is dynamism?

- The ability to change the structure or behavior of a software system at run-time.
  - Generally, in ways not explicitly planned for in the initially deployed system.
- Dynamism is essential for high-availability systems.
  - Medical devices
  - Space probes
  - Emergency response systems
- Dynamism is desirable for all systems.
  - PC security patches, virus updates
  - Service packs and other functionality upgrades
  - MMORPGs
- Dynamism is necessary for self-adaptive systems.
Examples of Dynamic Systems

- Dynamic load/install plugins in Internet Explorer/Netscape
  - Generally, these work without shutting down the browser.

- Not-so-dynamic systems
  - Windows Update
    - Only works without a reboot if resources weren’t in use.
  - JPL Space Probe system updates
    - Require restart of many non-core systems.
  - Application patches
    - Do not require a full reboot but generally require application restart.

We would like to move from this…

To this…
Techniques for Dynamism

- **Plug-in Mechanisms (e.g. Netscape/IE)**
  - Generally, specific extensions to a core platform.
  - Core usually remains unchanged.

- **Dynamic code loading (e.g. Java ClassLoaders)**
  - Handle loading new code and unloading old code.

- **Dynamic component instantiation (e.g. CORBA)**
  - Generally, handles unloading poorly.
  - Makes understanding and managing changes difficult
    - Little change visibility.
Outline

- Software Dynamism
- **Software Architecture**
- Architecture-Based Approach
  - Evolution Management
  - Adaptation Management
- Summary
Architecture: A New Perspective

- Architecture views software systems at the level of components and connectors.
  - Not lines-of-code or modules.
  - Not objects.
- Architecture generally leverages explicit software models that depict at least:
  - Software Components
    - Including provided and required interfaces.
  - Explicit (generally) Software Connectors
    - Provided and required interfaces.
  - Explicit links between the two.
    - Links form various system configurations.
Example of an Architecture-level Depiction:

```
ls -l ➔ grep "foo" ➔ more ➔ stdout
```
Example of an Architecture-level Depiction:

- ls -l
- grep "foo"
- more
- stdout

- Components
Example of an Architecture-level Depiction:

- Components
- Connectors (all pipes)
Example of an Architecture-level Depiction:

- Components
- Connectors (all pipes)
- Provided interfaces
Example of an Architecture-level Depiction:

- Components
- Connectors (all pipes)
- Provided interfaces
- Required interfaces
Example of an Architecture-level Depiction:

- Components
- Connectors (all pipes)
- Provided interfaces
- Required interfaces
- Links

```
ls -l
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
grep "foo"
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
more
```

stdout
A slightly larger example

Clock → Ports → Warehouses → Vehicles

Connector 1
- Planner
  - Port artist
  - Connector 2
    - Warehouse artist
    - Vehicle artist
  - Connector 3
    - Router artist
    - Router
  - Connector 4
    - Graphics
Outline

- Software Dynamism
- Software Architecture
- **Architecture-Based Approach**
  - Evolution Management
  - Adaptation Management
- Summary
Can we use architecture to manage and enact dynamism?

Leverage architecture-level models to:

- Understand and visualize the structure of the system.
- Depict, visualize, and understand changes to that structure.
- Guide automated tools in making changes to modeled components.

Leverage the above concepts to:

- Serve as the basis for self-healing/self-adaptive systems that make decisions and changes based on architecture-level models.
A Vision for Architecture-based Adaptation: The Figure-8 Diagram
Outline

- Software Dynamism
- Software Architecture
- **Architecture-Based Approach**
  - Evolution Management
  - Adaptation Management
- Summary
First Focus: Bottom Half

Key Insight: Keep the model and the implementation in-sync: a change to one automatically results in a change to the other.
Assumptions Implicit in the Figure-8 Diagram

- There is a modeling language.
- It can be accessed programmatically.
- Change descriptions can be expressed and deployed to (multiple?) sites.
- There is an implementation framework that supports dynamic changes.
- There is a tool that can maintain model ↔ implementation consistency.
A Modeling Language

Traditionally architectures are expressed in an Architecture Description Language (ADL):

- A formalism that allows you to ‘write down’ architectures.
  - At minimum, must support:
    - Components
    - Connectors
    - Interfaces
    - Links
  - For our purposes, must also support some mapping to implementation.
- Ideally, flexible enough to support many domains.
Problems with current ADLs

- **Too broad.**
  - Example: Acme
  - Supports arbitrary properties on elements, but only basic support for these properties

- **Too narrowly-focused.**
  - Examples: Rapide, Wright, Darwin, Meta-H, etc.
  - Support one domain or set of concerns well, others poorly.
  - Often lack implementation mappings.

- **Not extensible.**
  - Too hard to extend existing ADLs (and their tool-sets) to add information.
Our Solution: xADL 2.0

- An extensible, XML-based ADL.
  - Modeling features all expressed in language modules (XML schemas).
  - A composition of XML schemas make up an ADL.
  - Schemas available from UCI to support:
    - Design-time & run-time structural modeling.
    - Implementation mappings.
    - Product-line architectures (allows managing model evolution over time).
Change Descriptions

- Required to express and understand architectural changes.

- Different levels of change to consider:
  - Basic ‘diffs’
    - Describe changes between Model1 and Model2.
  - Product-Line ‘diffs’
    - Describe changes between Product-Line1 and Product-Line2.
  - Pattern-based ‘diffs’
    - Describe changes to patterns found in Model1 and patterns found in Model2.
Our Change Descriptions

- We currently support:
  - Basic ‘diffs’
  - Product-line ‘diffs’
- Both implemented as extensions to xADL 2.0.
- Accompanied by automated tools:
  - Automatically generate diff documents from two architectures or product lines.
    - (the architecture equivalent of ‘diff’ on UNIX)
  - Automatically merge a diff into an architecture or product line.
    - (the architecture equivalent of ‘patch’ on UNIX)
Architecture Frameworks

- Bridge the gap between elements found in architectural styles.
  - (components, connectors)
- ...and programming languages.
  - (classes, objects, procedure calls)
- Often support a particular architectural style or family of styles.
- For our purposes, should support run-time dynamism primitives (add/remove component, add/remove link, etc.).
- Potential candidates:
  - Component frameworks like COM, EJB, CORBA...
c2.fw: One such framework

- **Architectural style(s):**
  - Component- and message-based styles.
  - Special support for C2 style.

- **Programming languages:**
  - Java
  - (Other frameworks available for other languages)
    - C++, Embedded C++, Ada95, etc.

- **Dynamism Primitives**
  - Exposes a single, unified interface for adding/removing components, connectors, links, interfaces, etc.
Maintain Consistency

- Tool must monitor both architectural model and running system:
  - When model changes (e.g. due to patching a diff), must modify the implementation to match.
  - When application changes (e.g. due to component failure or shutdown) must modify the model to match.

- Algorithms to accomplish this with different kinds of models and dynamism primitives are still being researched.
Architecture Evolution Manager

- A component of our architecture-based development environment that performs this function.

- Currently supports local changes, will evolve to support distributed changes and things like maintaining component state across replacements/upgrades.
Open Dynamism Research Issues

- Distributed systems
  - Encounter many new types of failures—network failure, host failure, etc.

- Infrastructure adaptation
  - Can be partially addressed with a multi-level approach (AEMs running inside other AEMs).
  - We have a proof of concept in our current infrastructure.

- Maintaining state across component upgrade/replacement.

- Assessing/maintaining reliability.
A Vision for Architecture-based Adaptation: The Figure-8 Diagram

Feedback and Planning

Implementation Issues
Outline

- Software Dynamism
- Software Architecture
- **Architecture-Based Approach**
  - Evolution Management
  - *Adaptation Management*
- Summary
Key Insight: Managing and planning adaptations is done at the architectural level, independent of the application semantics.
Implicit Assumptions

- Enact changes and collect observations
  - Changes can be enacted and observations collected.

- Evaluate and monitor observations
  - Observations can be evaluated for their meaning.

- Plan changes
  - Modifications can be planned according to some criteria.
Planning Changes

- Interesting questions:
  - *Who* is responsible?
    - System designers, administrators, users.
  - *When* should changes be enacted?
    - Pre-planned situations, user discretion.
  - *What* are the specifics?
    - Pre-planned change scripts, user-defined modifications.
Self-Adaptive Software

- Software that can modify itself in response to:
  - Software faults.
  - Changing deployment conditions.
  - New behavioral requirements.

- Modifications do not need human intervention.

- The system itself decides...
  - ...when changes need to take place.
  - ...what the specifics of these changes are.
Various Approaches

- Changes are pre-programmed into software components.
  - Little visibility, close coupling with implementations.

- Pre-planned change scripts.
  - Static responses for a non-static world.
  - Limited to the foresight of the system designer.

- Adaptive algorithms
  - Domain-specific solutions in a constrained environment.

The challenge lies in developing an approach that ensures high visibility, strict decoupling, and dynamic evolution.
A Knowledge-Based Approach: Overview

- An architecture-centric, knowledge-based approach which reasons about change based on observations and policies.
  - Observations comprise known information.
  - Policies define when modifications should take place and what the responses should be.

- Features:
  - High visibility
    - Knowledge and policies are specified as part of the system’s architectural description.
  - Decoupled
    - Policies are strongly-decoupled from component implementations.
    - Components need not have any knowledge of adaptation.
  - Dynamic
    - Observations may be transient.
    - Policies may be added, removed, and composed.
Knowledge-based Adaptation Policies

- Policies determine the timing and specifics of adaptations.
- Knowledge-based policy structure:
  - Observation+ → Response+
- Adaptation policies are specified at the architectural level, and can be dynamically modified at run-time.
- Representational support using xADL 2.0, and expert system implementation using the Java Expert System Shell (JESS). Again, fully extensible.
Adaptation Observations

- Observations express architectural knowledge.
  - Events indicating non-nominal operation.
    - Component or connector failure.
  - Events indicating the structure of the architecture has changed.
    - Components and connector addition, link removal, etc.
  - Events which may indicate composition errors.
    - Requests and notifications go unanswered or ignored.

- These observations are supported by:
  - xADL 2.0 modeling extensions.
  - c2.fw implementation framework.

- But, they are easily extended to accommodate domain-specific information.
Collecting Observations

- May be emitted by components themselves.
- Collected using independent software probes.
  - May be dynamically inserted into the running system.
  - Primarily observe communication patterns.
Adaptation Responses

- Responses indicate architectural modifications.
  - Addition of architectural elements (components, connectors, or links).
  - Removal of architectural elements.
  - Addition and removal of observations or adaptation policies.
  - Composite operations.

- Using these responses, the system can modify both:
  - Its structure, and therefore its behavior.
  - The policies guiding adaptations themselves.

- Again, supported by xADL 2.0 extensions and the c2.fw framework but also fully extensible.
Enacting adaptations

- Modifications due to adaptation responses are not directly enacted. May need to...
  - Maintain architectural constraints.
  - Log and publish modifications.
- Architecture Adaptation Manager (AAM)
  - Point of coordination for these “value add” services.
- AAM (to be) included in the ArchStudio 3.0 toolkit.
  - Currently, coordinates constraint maintenance facilities.
A short example

- Unmanned Air Vehicle (UAV) with limited on-board resources.
- Operates software components supporting various tasks.
  - Nominal navigation.
  - Threat avoidance navigation.
  - Image processing.
  - Inter-networking management.
- In certain situations, some of these tasks take precedence.
An example policy

- Policy giving threat avoidance precedence.
  
  ```xml
  <AdaptationPolicy id="Avoid_threats">
    <Description>Replace normal navigation.</Description>
    <Observation id="Threat_Detected" />
    <Response id="Replace_Component"
      old="Nominal_Nav" new="Threat_Avoidance_Nav"/>
  </AdaptationPolicy>
  ```

- Observations
  - Domain specific: Threat Detected.

- Responses
  - Composite operation:
    - Remove Nominal navigation component.
    - Adding Threat Avoidance component in its place.
Open Research Issues

- **Distributed systems**
  - Can local adaptation decisions give rise to global adaptive behavior?

- **Expressiveness**
  - Is this knowledge-based approach expressive enough?

- **Safety and Predictability**
  - Given the non-deterministic nature of the approach, can guarantees about the system’s architecture be made?
  - Are constraints sufficient for this?
Outline

- Software Dynamism
- Software Architecture
- Architecture-Based Approach
  - Evolution Management
  - Adaptation Management
- Summary
Summary

- *Architectural* models are central not only to software development but also evolution.
- Architecture provides a promising approach for:
  - Dynamic, run-time system evolution.
  - Developing self-adaptive capabilities.
- “Proof of concept” techniques and tools:
  - xADL 2.0 architecture description language.
  - ArchStudio 3 environment.
  - Knowledge-Based Architecture Adaptation Management (KBAAM).