Supporting Dynamic Reconfiguration in Ground Systems Architectures

Phillip Schmidt
Information Technology Department
schmidt@aero.org

The Aerospace Corporation
El Segundo, CA 90245-4691
Outline

- Definition
- Motivation
- Requirements/Design Issues
- State of the Art Assessment
- Some Techniques
- An Architectural Approach
- Observations/Recommendations
Definition

- A **configurable, reconfigurable, or adaptive** computing system is one that is capable of modifying its behavior based on changes in the environment.
- **Dynamic reconfiguration** permits change while system is executing.
- This adaptiveness is usually designed to support one or more of the following:
  - Changes in operations (e.g. sustainment, new missions)
  - Failures (e.g. processor or communication)
  - Performance

Environment Changes

<table>
<thead>
<tr>
<th>uncertain</th>
<th>Fault Tolerance</th>
<th>Adaptive Systems</th>
<th>controlled changes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Configurable</td>
</tr>
</tbody>
</table>
Motivation

• Traditional ground system architectures inflexible
  – Fixed configurations in well-prescribed scenarios
  – High O&M (sustainment) costs

• Formal verification is not the cure

• Future requires operation in a changing environment
  – Addition, removal, migration, replacement
  – 24 x 7 operation => controlled dynamic reconfiguration
  – New customized roles
  – Increased complexity (space autonomy)

• Object service architectures technology foundation
  – Replication of services not good enough (restrictions on service use)
  – New component-based architectures support object evolution
  – Configurable infrastructure is needed
  – Agent-based computing: execution, monitoring, planning, coordination

• “Information warfare”
  – Situation assessment module
  – Security
Requirements

• Support communication across heterogeneous hosts
  – Avoid changes to the OS
• Current configuration must be accessible.
  – Identify components to be configured and their current state
  – Uniform interface
• Bindings (interconnections) are not compiled into modules
• No covert communication among modules
• Provide ability to add/remove modules and bindings
• Provide access to messages in transit
• Synchronize activities
• Client support for interface discovery
• Support for component evolution (versioning)
• Provide adequate system performance
Design Trade-offs in Dynamic Reconfiguration

- Location transparency vs client-controlled binding
- Synchronous vs. asynchronous applications
- Deferred interaction (message-based) vs non-deferred (realtime)
- State vs stateless contexts
- Run-time reuse vs compile-time reuse
- Aggregation/delegation vs class-based inheritance
- Idempotent vs nonidempotent actions
- Transaction vs nontransaction
- Semantic component knowledge
- Security/access control
- Operating system vs middleware design
- Performance vs overhead
- Legacy vs new applications
State of the Art Assessment

- Early focus: Process Migration
  - Created new OS. (Full state context, but limited to homogeneous environments.)
  - DEMOS/MP, Charlotte, CONIC
- Today’s focus: Layered solutions for heterogeneous environments
  - Improving the OS kernel: x-kernel, Regis, Kernel Tool Kit, Kea
  - Middleware: e.g. Prgmr Playground, Polylith bus, ISIS, ORB proxies, filters
  - Structure of applications for reconfiguration: recon interface
- Component-based infrastructures evolving
  - Commercial offerings - principally stateless models, document/office oriented
  - Agent-based computing - promise for greater flexibility, autonomy
- Challenge: Interface discovery and semantic information
Techniques

• Separate computation and communication
• Separate computation and reconfiguration
• Dynamic interfaces and semantic information:
  – Part of the architecture definition: Configuration languages identify what a component requires/provides. (Gerel, Darwin, Rapide)
  – Part of the programming language - Java Beans retrospection, introspection
  – Part of the infrastructure - ActiveX/DCOM interface, CORBA DII (registry, repository)
  – Part of the environment - KQML/ KIF ontologies, reconfiguration context information (rule sets)
• Consistency and coordination/monitoring
  – Recovery and avoidance
  – State save/restore mechanisms
  – Reconfiguration points (resource-lock)
  – Adaptive situation assessment (agent-based computing model)
• Event-based communication (multicast, publish/subscribe)
Architectural Approach
(Basic Notion)

• **Construct an architecture to support dynamic object evolution during normal operations.**
  - Identify a reconfiguration target
  - Place system in a “safe” state
  - Save any internal state information of the reconfiguration target
  - Preload the new service
  - Perform state mapping to restore state information
  - Redirect communications of reconfiguration target
An Architectural Approach

- Reconfiguration Execution Points
- State Save/Restore Methods
Observations/Recommendations

- IDL alone is not sufficient. Work needs to be done on understanding ADLs better to generate appropriate specifications.
- Recognize need for a reconfiguration interface to support a common configurable infrastructure.
  - Object service architectures and application structure
  - Agent-based computing
- Design for configurability
  - Study applicability of dynamic reconfiguration. Dynamic does not mean instantaneous and uncontrolled.
  - Manage the evolution of interfaces
  - Recognize need for a supporting infrastructure
  - Source code may not always be available.
- Legacy systems will likely need to be restructured.
  - Wrappers
  - Compiler augmentation
- Accurate state representation crucial
- Watch performance