Flexible Architectures for Custom Ground Stations

Christopher Landauer
National Systems Group, The Aerospace Corporation
E-mail: cal@aero.org, Phone: (703) 318-1666

Kirstie L. Bellman
The Aerospace Corporation
on loan to: DARPA / ITO
E-mail: kbellman@arpa.mil, Phone: (703) 696-2219

27 February 1997

GSAW’97: Ground Systems Architecture Workshop

Outline
Design Goals - What we want to do
Infrastructure Requirements - What we need to do
Implementation Mechanisms - What we can do
Architectural Standardization

Many architecture efforts fail
- Overly rigid resulting architecture requires absolute compliance
- Operational systems must continue operating while they
  . Incorporate any new technology
  . Adopt new practices

Common Framework
- Different users require different exceptions and specializations

Treat architecture as an "agreement"
- Agreements grow over time and deepen if they are mutually satisfactory
- Agreements are two-way
- Good agreements respect individual needs and differences

Our Goal: create "limited buy-in" architectures
- Allow heterogeneity under a common framework
- Allow different levels of commitment and use
- Provide sufficiently flexible technology
  . Infrastructure
Infrastructure

Our research in complex systems has shown the importance of infrastructure
- Explicit components and activities of the system
  . Whose function is to help organize the rest
  . Resources performing actions on other resources
    script generators, activity monitors, and planners
  . Resources reasoning about other resources’ capabilities and behavior
    before, during, and after applying the other resource
- No matter what kinds of computational models are used
  . The system needs infrastructure for complex interactions

Our "wrapping" approach
- Intelligent integration infrastructure for constructed complex systems
- Provides a natural means of reuse and customization
  . Of all computational resources
  . Use the same mechanisms as with all other parts of the system
Wrapping Approach to Intelligent Infrastructure

The Wrapping Approach to Heterogeneous System Infrastructure
- Explicit, machine-processable information about resources
  AND
- Active processes that use the information
Wrap Everything (Everything!): tools, data, UIs and other resources
- Wrap different uses of complex resources
  . Many wrappings for one resource
- Wrap combinations of resources that apply together
  . Many resources for one wrapping
Processing the Wrappings
- Intercept all component interactions, wrappings help make connections
  . Requests for information need not know information source
Advantages of Wrapping
- Simplified development of large application environments
- Provides principled infrastructure organization
- Descriptions available for other analyses
- Can generate code interfaces (often called "wrappers") according to context
Simplifying Uniformities of Approach
- Uniformity of Description (wrapping information)
- Uniformity of Processes (wrapping processes)
- Same methods used for describing and selecting processes
  . Both infrastructure and application processes
Wrapping Processes (Problem Managers)

Problem Managers (PMs) can pose problems and organize their solution
- Share common workspace for problem study
- Special purpose PMs: SMs and CM

Study Manager (SM) implements the basic steps for problem study within a context
- Organizes the problem study sequence (simple task planner):
  . Interpret Problem:
    Match Resources,
    Resolve Resources,
    Select Resource,
    Adapt Resource,
    Advise Poser
  . Apply Resource
  . Assess Results
- Recursive!
  . Same method used to select and apply wrapping processes, (e.g., SM)
  . Criteria for bottoming-out the recursion

Coordination Manager (CM) provides the initial context and problem
- Poses first problem: "Find Context"
- Simple analogue of LISP’s "read-eval-print" loop
- SMs are resources that apply to problem "Study Problem"
Wrappings Allow Customization

An Architecture IS a Model
- Defines roles and responsibilities of components
- Defines interconnections and information exchanges
Models of all components as computational resources
- Custom versions for custom applications

Wrappings
- Define component’s role in the architecture
  . Context requirements and use
  . Limitations, assumptions about behavior, including performance
- Select and compile custom versions according to target platform
  . Knowledge-Based Polymorphism
- No performance hit
  . Can compile the fixed decisions out of the run-time system

Problem Posing Interpretation
- New interpretation of any programming language
  . Pose problems and apply resources instead of call functions
- Allows reuse without code change
  . Needs a modified compiler (we know how to do this)
- Can intercept all function calls (resource requests)
- Insert measurements, consistency checks
  . Divert function call to new resource implementations
Infrastructure Summary: To Put Pieces Together, You Need

The Right Pieces
- Resources and Models of their behavior

The Right Information About the Pieces
- Wrapping Knowledge Bases

The Right Mechanisms to Use the Information
- Study Manager, Coordination Manager and other Problem Managers
- Problem Posing interpretation

Wrapping is a Knowledge-Based Approach to Software Composition
- Explicit meta-knowledge about ALL resource uses
  . Assumptions, limitations, styles of use
- Integration is more than assembly
  . Not just "how", but "whether" and "when"

What if the Right Pieces are Unavailable?
- Software Disintegration of existing systems
- New model and resource development
References on Wrapping


References on Wrapping (continued)


Wrapping Information

Wrapping Knowledge Base (WKB) Information
- Not just about how to use the resource
- Also when and whether: assumptions, limitations, scope, styles of use

For Each Resource, a Sequence of Entries
- Problem Identification
- Context under which this resource might be appropriate for this problem
- Requirements (information or services) for using this resource
- Products (information or services)

Use of Wrappings (Intelligent User Support Functions)
- Selection (which resource to use)
- Assembly (how to use it: data formats and protocols, mainly syntactic)
- Integration (whether and when to use it: more semantics, also context)
  . The keys to Use
- Adaptation (how to set up resource for problem)
  . The key to Re-Use (hard!)
- Explanation (why the resource was used)
  . The key to Understanding (hard!)
Problem Posing as a Programming Paradigm

New declarative programming style that unifies most major classes of programming
- Subsumes functional, imperative, logic, and message programming
- Programs do not “call functions”, “issue commands”, “set constraints”, or “send messages”
  . They “pose problems”
- Programs are not written as “functions”, “modules”, “clauses” or “object methods”
  . They are written as “resources” that can be applied to problems
  . Invocation references become problems to which resources may apply
- Programs do not “invoke functions” or “dispatch messages”
  . They “study problems”
- Change in attitude about sources of control
  . From user “issues commands” to user “poses problems”
  . Allows more creativity in system response
- Programs in _any_ programming language can be interpreted in this style
  . Same syntax, completely new interpretation
    (reuse is greatly facilitated)
  . References are connected to definitions through knowledge bases
    (Knowledge-Based Polymorphism)
  . Allows interconnecting multiple languages
Provides support for prototyping within same system
- Programming in the large in addition to programming in the small
**Complexity Management**

Layering to reduce complexity
- Quite successful for computer networks
- Other systems (especially simulations) are not as clearly hierarchical

Locality and Distribution
- Conceptually and physically

Reductionist approaches lead to ‘component-based’ architectures
- Large numbers of small components providing information services
- We extend this notion to all resources
  - Resources: ALL Computational and Informational Elements
  - Includes architectures, scripts and plans that organize other resources

Resource composition occurs
- At Different Times
  - Definition time (APGs, Little Language Compilers, ...)
  - Construction time (Linkers, ...)
  - Load time (Loaders, ...)
  - Run time (Dynamic loaders, ...)
- At Different Frequencies
  - Just once
  - Repeatedly (periodic, occasional, triggered by events, ...)
- Need different kinds of knowledge for different times and frequencies
Variables

System organization should have “variables all the way down”
- Variability is coordinated by the system itself
  . Maintains all models and resources
  . Manages all architecture layers

Variability allows multiple models to coexist in a system
- Accommodates model analyses for different levels of detail
- Allows explicit comparison and simultaneous monitoring of model behavior

The key to maintaining the variability: treat ALL parts of the system as resources
- Resources may be selected and used in interesting combinations
  . Computational processes
  . Relationship and other constraint enforcing and observing processes
  . All of the models
- Includes architectures, scripts and plans that organize other resources
- New resources and new combinations may be added easily

Several important kinds of “information services”
- Computation, Information, Distribution, Interaction
Spontaneous Activity and Reflection

System will have many different kinds of spontaneous activity
- Self-maintenance functions
  . Monitoring
  . Analysis
  . Reconfiguration
  . Safety checking and introspection
- Monitor can say “stop this line of reasoning” or “stop this resource task”

System will keep activity information around for examination
- For offline and background checking and repair of data structures
- Should be some redundancy of structures links to facilitate the repair process

Introspection incorporates system experience

System monitors will examine two questions
- What have I been doing? (an induction process)
- What could I be doing? (a deduction process)

Continual contemplation
- Simulations of interesting problems
Programming for Autonomy

Autonomous systems in complex environments must be complex systems
  - Repertoire of possible alternative behaviors
    . Processes for selecting them
    . Fallback choices when situation estimates are wrong
    . Quick partial selections to reduce decision time
  - All activity is situated, strongly dependent on context
    . Different decision processes in different situations
  - Integration of many different kinds of models
    . Of external and internal environment, architecture, behavior,
      and system itself, in context
    . Both predictive and empirical, continual validation processes

Wrapping information and processes provide:
  - Flexibility and coordination
  - Simplifying uniformities of expression
    . Computational resources (uniformity of description)
    . Problem Study (uniformity of processing)
  - Multiplicity of responses
    . Problem Managers organize resources
  - Computational Reflection (recursion in meta-direction)
Application to System Development

Wrappings unify and simplify the treatment of system development
  - Model-Based System Development

System Construction
  - Make the code less important by having explicit models
  - System construction artifacts
    . Usually discarded once a program is delivered
    . Some are not generally constructed explicitly at all

System Maintenance
  - Artifacts support maintenance (explanation of models)

System Re-use
  - Artifacts support re-use (selection of models)

System Re-engineering
  - Reconstruct those artifacts from existing programs
    . This step is generally very hard
    . It is necessary for _any_ re-engineering process
      (only the artifacts will be different)

Some of these processes must be at least partly automatic for an autonomous system