Software Architectures: What are we Building?

March 1999

Roger J. Dziegiel, Jr
AFRL/IFTD
525 Brooks Rd
Rome, NY 13441-4505
(315)330-2185
dziegielr@rl.af.mil
**PROBLEM**

- Paradigm shift to the development of application families and an increased reliance on component reuse necessitates advanced mechanisms for representing system architecture
- No commonly accepted definition of architecture beyond notions of components and connectors; emerging ADLs, but terminology is not fixed and competing languages and logics are used
  - Difficulty predicting/analyzing functional/extra-functional characteristics of integrated components
  - Architectural archeology (e.g., recovery of architectural information from legacy systems)

**APPROACH**

- Develop common architecture specification notions/languages to facilitate multiple views through interrelated ADLs
- Develop integrated architectural analysis, design, and measurement capabilities to assist in understanding functional and extra-functional characteristics
- Automate composition/synthesis of application implementations from architectural specifications
- Preserve all aspects of system being developed including behavioral and architectural specifications to enable subsequent examination, modification, and/or transformation

**PAYOFF**

- Architecture is formally specified and used to generate/develop/evolve source code
  - Architecturally consistent implementation
  - Cost-effective
  - Easily evolvable
Architecture & Generation

Participants

- Carnegie Mellon University (CMU)
- Michigan State
- Oregon Graduate Institute (OGI)
- Rice University
- Stanford University (2 projects)
- USC/ISI (2 projects)
- University of Indiana
- University of Texas
- Vanderbilt University

Software Generators

- USC/ISI
  - R. Balzer
- Vanderbilt
  - J. Sztipanovits
- University of Texas
  - D. Batory
- OGI
  - J. Hook

Constraints

- CMU
  - M. Shaw
- USC/ISI
  - R. Balzer
- Stanford University
  - G. Wiederhold
- University of Indiana
  - D. Friedman

ADL Semantics

- USC/ISI
  - D. Wile/D. Garlan
- Lockheed-Martin
  - R. Creps

Modeling & Analysis

- CMU
  - D. Garlan
- Michigan State
  - B. Cheng
- Stanford University
  - D. Luckham
- U Mass
  - G. Avrunin
  - L. Osterweil
EDCS Architecture Vision

ObjecTime Evaluation

• Features
  – End to End Support
  – Highly Integrated
  – Intuitive
  – Responsive

• Limitations
  – Single Architectural Style
  – Single Modeling Formalism
    • (Finite State Machines)
  – Fixed Analysis Tool Set
  – Fixed set of constraints

  Effect
  Limited Applicability
  Incomplete Coverage
  Limited Feedback
  Unchecked Requirements
EDCS Architecture Vision
EDCS Value Added

Leading Edge
Commercial
Limitations

Effect
Single Architectural Style

Limited Applicability
Single Modeling Formalism

Incomplete Coverage
Fixed Analysis Tool Set

Limited Feedback
Fixed set of constraints

Unchecked Requirements

EDCS Objectives
Multiple Architectural Styles
Multiple Architecture Langs.
Multiple Modeling Formalisms
Extensible Set of Analyzers
Dynamic Constraints
Conformance Testing

Develop Technology
needed to extend
Leading Edge
ADAM - Architectural Design, Analysis and Measurement

**Architectural Design**
- Evaluating Trade-offs among alternative designs
- Formal Event-based modeling of system behavior
- System design and evolution based on architectural styles

**Architectural Analysis**
- Style-specific analysis and constraint enforcement
- Simulation and animation of SW architectures to explore behavioral
- Fault Tolerance analysis based on Monte Carlo simulation

- ACME-Based Analysis Tools
- Armani
- UniCon2
- Rapide
- Web-Based ACME Browser
- Style-Specific Constraints
- Domain-specific Analyses
- Performance
- Fault Tolerance
- Real-time Scheduling
- System Construction
- Simulation
- Synchronization

Lockheed Martin
ACME developed as a common interchange mechanism
- Supports common static analysis services
- Provides tools access through ADL translation
- Facilitates development of domain-specific notations
Armani: A Configurable Software Architecture Design Environment
- rapid construction of customized software architectures
- configurable design environment and language for describing the vocabulary of SW architecture design domains, the structure and properties of software systems, constraints on the evolution of system designs, and design heuristics

UniCon2: An Architecture Description Language and Toolset
- create and manipulate high-level architectural abstractions
- generates support and glue code and invokes compilers as necessary to translate the descriptions into running systems

Wright: Formal Modeling of Software Architectures
- ADL that permits the behavioral specification and analysis of a software architecture
- supports rules for checking the consistency and completeness of architectural designs
Event pattern language expressing both event causality and event timing, for rapidly configuring event filters, aggregators and constraints.

Complex event processing based on event patterns, for enhancing the capabilities of existing distributed systems.

Java extensions to express architecture concepts with an event-based semantics.

Complex event processing and architecture conformance checking, hosted on commercial middleware.
Impact

- Automated component composition, **not custom coding**
- Shift engineering focus to designing for change
- Months of setup time reduced to hours
- AEDC estimated 1996 return – **$10M**
Jakarta: A Tool Suite for Constructing Software Generators
- Component Composition and Software System Evolution
- Compiler Construction
- Metaprogramming
- Self-Adaptive Software

Capabilities
- Extensible Java
- Automated Evolution of Object-Oriented Software
- Language support for compositional and transformational components, and dynamic and static components
Architecture & Generation

Work Avoidance, Incrementality, and “Rightness” by:

• Enabling automatic analysis and early detection of errors

• Enabling reuse and product line development
  – Transferable, reusable abstractions
    • High-level, domain specific analysis and composition
    • Basis for ensuring validity of implementations
  – Analyze Risks of Alternative Approaches

• Supporting incrementality
  – Automated construction or modification of systems from specs

• Supporting optimization (non-functional attributes)

• Providing basis for software process improvement
Key Technology: Automatic Analysis and Early Detection of Errors

Example analyses -- show that:

• Given sequence of events can (or cannot) occur
  – Deadlock, livelock conditions
  – Sequence of processing steps in distributed applications (e.g., authorization completed before data is accessed)

• Components can (or cannot) be composed with predictable properties, e.g.:
  – Timing
  – Resource use, starvation

• Dynamic interaction (reconfiguration behavior) ensures, e.g.:
  – Components exist before invocation
  – Links don’t exist to deleted components
ARCHIE
ARCHIitectural DEsign and Analysis Toolkit

Architectural Design
- Evaluating Trade-offs among alternative designs
- Formal Event-based modeling of system behavior
- System design and evolution based on architectural styles

Architectural Analysis
- Style-specific analysis and constraint enforcement
- Simulation and animation of SW architectures to explore behavioral
- Fault Tolerance analysis based on Monte Carlo simulation