Hierarchical Dynamic Models for Verifying Parallel Distributed Real-Time Systems

Heinz Schmidt
Centre for Distributed Systems and Software Engineering
Monash University
11/2005
Overview

- Architectural Models and ADLs
- Darwin, Koala, RADL
- Applications: what do we model or predict?
- Context-dependent analysis and parameterized components
- Work in Progress
Example Component-Based System
Example Model

Production Cell Animation
Reliable/Rich Architecture Definition Language (RADL)

- Many ADLs now available [See M.Shaw ICSE’01, Medvidovic&Taylor TSE1/2000]
- UML2 includes ADL in so-called superstructure standard
- RADL is based on ICL’s DARWIN [joint project 94-96]
Our extension of Darwin’s Software Architect Assistant: TCAT [98]

- From object-based to object-oriented
- Beyond Objects: Components and Architectural Assemblies

<table>
<thead>
<tr>
<th>ADL Features</th>
<th>SAA / DARWIN</th>
<th>TCAT / RADL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objects</td>
<td>Object-based</td>
<td>Object-oriented</td>
</tr>
<tr>
<td>Classes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>subtyping</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>inheritance</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>component protocols</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>connector protocols</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Language binding</td>
<td>CORBA</td>
<td>Java</td>
</tr>
</tbody>
</table>
KOALA - Darwin spin-off

- Industrial-strength spin-off at Philips (Eindhoven)
- Now applied to all consumer electronics architectures/ product lines

Additions to Darwin

- C modules: breaking out of ADL in primitive components
- Switches: mini-components for complex connector scripting s.a. dispatch, delegation of calls etc but without separate binary black-box component

Koala compiler to C - 10-20% of CE code ADL derived

- Strength: in modelling, reuse, documentation, compilation
- Weakness: no behaviour specification or analysis
Meanwhile...

KOALA: ongoing work on ADL - IDE
- initial prototype with graphical and textual capabilities
- exploration of Eclipse integration
- first analysis support s.a. tool for concurrency analysis

DARWIN: richer dynamic models
- UML2: Scenarios, MSC, UML2 state charts
- Semantic extensions: FSP revisions and target of UML translation, fluent calculus and probabilistic LTS
- Improved analysis: LTSA2 and beyond

RADL: extra-functional analysis and industrial applications
- WCET prediction and consistency analysis
- Parameterised models and probabilistic prediction
- ... move away from graphic models to Java API strength
One of the early motivations for **specifying** software architectures explicitly is the use of architectural design information for structural **reasoning** about component, assembly and system properties.

[Mary Shaw: The Coming of Age of Software Architecture ICSE’01, IEEE]
RADL Rich Dynamic Models

- Extended DFAs underlying DFSMs
  - variables capturing shared external and internal state and folding dynamic models
  - hierarchical state refinement
- Generic extra-functional property modelling
  - real-time, space and probabilities (reliability, availability...)
  - different type iterators walking dynamic models for ease of extra-functional analysis
  - aspect-oriented EFP weaving
- Sophisticated model composition operators
  - architectural component assembly
  - dynamic model manipulation and analysis in component dependency networks
Architectural Design & Composition in RADL

Component composition 1: Connectors for component „wiring“

Connectors: two-way interaction or flow; direction indicates dominant control, e.g. initiation of sequence

Hierarchical architectural components

Component composition 2: protocol expressions
Robot = (Arm1 . Base . Arm2) | WeakSync
including constraints & properties

Gate: interface object of defined interaction protocol type

requires gate
provides gate
Dependent FSMs and Architectural dependency networks

Connector DFAs (extended dependent finite state automaton / trace languages)
Component DFTs (extended dep. fin. state translator / trace translations)

R DFA       P DFA
Component DFT
DFT computed for composite
RADL Hierarchical Dynamic Models

Protocol and/or property models “canned” with packaged components and built into architectural dependency network at configuration or deployment.
Interaction and Translation Protocol Types

Interaction protocol types (port interface abstraction)
- Defined by trace languages and automata based in Petri nets and automata theory
- True concurrency based on partially ordered traces over interactions

Translation protocol type (component abstraction)
- Dependencies between interaction ports
- Distinguish input, output, hidden subalphabets $\Sigma = I + O + H$
- Hierarchical protocol name spaces following architectural decomposition

Petri nets covered by atomic (sequential) subnets (S-components)
- Closed under rational operators
- Lattice of interaction protocol type $\Rightarrow$ subtyping
- Time, reliability, usage probabilities and other extra-functional property annotations
Postponing state space explosion

Partial order traces and vectors of statemachines composing underlying Petri net

- vector product preserves local ordering and synchronisation constraints
- concatenation and shuffle as special case based on distinction of dependency D and independence I relation between labels
- additional fairness constraints sufficient for decidability (see trace)

\[
\begin{align*}
\text{a} & \quad \text{b} & \quad \text{c} \\
\text{d} & \quad \text{e} & \quad \text{g} \\
\text{a} & \quad \text{b} & \quad \text{c} \\
\text{d} & \quad \text{e} & \quad \text{g}
\end{align*}
\]

if \quad \text{dDe}
Application: eCAP ABB Collaboration

- IEC 6 1131-3 code
  - Function Block Diagrams & ST (imperative)
  - restrictions: assume few loops, no pointer arithmetic, no recursion, no suspend, etc..

- Currently 800xA includes almost no support for performance analysis
  - benchmark data for example components on various CPUs;
  - benchmarking methods manually driven, time consuming
  - client problem resolution may require specialist support at client sites for days/weeks
eCAP - ABB Collaboration

Goal: tool to diagnose extra-functional properties (load, WCET) of IEC 6 1131-3 component software designs for ABB AC800 Controllers

Benefits: reduce design, testing & maintenance costs

Results

- prototype running on small examples
- patent application (filed May 2005)
- deployment for in-house evaluation requested
- publications: WORDS 2004 (Fall), ERCIM Workshop on Software Intensive Embedded Systems
ECAP Polymetric views

- Multiple metrics
  - BCET/WCET/Deviation
  - Code complexity
  - ...

- Beyond summary tables

- Visual cues
  - following source structure
  - basic and aggregate entities
  - length/height outliers
  - color coding

![ECAP Polymetric views Diagram]
ABB Industrial™ (TM) 800xA Extension protected by joint patent: proprietary feasibility checking; analyser plugins .NET/C# (open-source license); commercialisation gate 5 passed 12/2005
Production Cell - press “provides” gate
Context-Dependent Analysis

- Automation library component software in industry becomes more and more ‘capable’ and ‘configurable’ for different contexts to maximise reuse without loss of quality.

- Software components are used in restricted deployment environment guaranteeing much stronger assumptions than those made for generic library components.

- Predicted times and reliability based on context-free property models (adding resource usage of components without regard for embedding in context) are not sufficiently accurate except for the simplest, lowest-level, components.
WCET: reuse vs. context dependence

- traditionally whole-of-system approach
  - context-free or global analysis
  - overly pessimistic for many contexts
  - results not easily reused
  - requires full source
- need “context-dependent” approach
Context-dependent property analysis

Context of component type T is restriction of T's stimuli-response ‘translation’ protocols

qualify “guard” property G of T with a context C: <G,C>

the universal context of T is simply its DFSM's original provided protocol: <true,P>

<G1,C1> only valid in a given context C2 if C2 equals or (further) restricts C1

analysis techniques must be adapted to handle context

properties reusable only if guards satisfied
Parameterised Components
- beyond provides and requires interfaces -

Parameter contexts
- development, e.g. reuse settings
- configuration, e.g. product line variation
- deployment, e.g. installation settings and services
- execution, e.g. initialization and runtime services

Extra-functional annotations in parameters
- runtime usage probabilities
- architecture-dependent WCET
- policies (such as fault-tolerance)
Probabilistic Component Parametrisation

- **Provided Gate protocol types**
  - unknown usage profile
  - usage probabilities for component services
  - load probabilities: passive - active, night - day etc.

- **Required Gate protocol types**
  - unknown deployment context
  - context services reliability
  - availability of independent resources

- **Component Abstract Machine translation types**
  - unknown realisation details
  - probabilistic state transitions
  - probabilities hiding resource information details

---

- Provided
- Required

---

Basic or composite (subsystem) component

- P_1
- P_2
- P_3
- P_n

- R_1
- R_2
- R_m

Configuration

Deployment

External components

Infrastructure

User/client

- Provided
- Required
Towards Handling Cyclic Dependencies

Connector DFAs (extended rational trace language)
Component DCFTs (extended context-free translations)

R DFA sublanguage P DFA

Component DCFT

DCFT computed for composites
Simple Reduced Dependency Networks

- Required signatures $R_1, R_2, R_3, R_4, \ldots$
- Provided regular $P$
- Abstract CF component $C$
- Projected CF required protocol languages
- Projected regular provided protocol languages
- Binding problem decidable (!) and efficiently so (practical?)

\[
\text{proj}_i(L_C) \cap \text{proj}_j(L_{P'}) = \text{proj}_i(L_C)
\]
Related Work

- WCET analysis
  - most work focuses on monolithic/whole-of-system analyses
- Model checking / automata research
  - efficient model “folding”
  - hybrid/extended automata (Presburger arithmetic)
- Context modeling
- Architectural modeling
  - UML2, Koala (Phillips), SaveComp (MRTC et al), Robocop
Conclusion

- Foundations, key ideas and approach for reliable/ rich architecture definition language (RADL)
  - Protocol dependency models important - not just connector conformance
  - Compositional extra-functional property models
    - WCET, usage probabilities, reliability
  - Context-dependent analysis

Implementation:
- Proof-of-concept prototypes and industrial tools
- Ongoing generalisation and extension