Model-Driven Development, Analysis, and Optimization in Cadena

*SAnToS Laboratory, Kansas State University, USA*

http://cadena.projects.cis.ksu.edu

**Support**

US Army Research Office (ARO)
US National Science Foundation (NSF)
**US Department of Defense**
Advanced Research Projects Agency (DARPA-PCES)

Rockwell-Collins ATC
**Lockheed-Martin [Eagan] - (PCES sub-contract)**
Honeywell Technology Center and NASA Langley
IBM, Intel, Sun Microsystems
Distributed Components

Java

C++

C

Network

C

Java

C++
Distributed Components

Middleware (e.g. CORBA)
- Event Service
- Transaction Service
- Naming Service
- Synchronization Service
Goals of the Cadena Project

An Integrated Development Environment for Analysis, Synthesis, and Verification of Component-based Systems

I. Platform for real-world experimentation with technologies for building high-assurance distributed systems using CORBA Component Model

... robust tool environment suitable for industrial experimentation

... model-based development, middleware configuration, and code synthesis

... pluggable light-weight specification, analysis, and verification techniques

... customizable to different domains/product lines

II. Avenue for collaborating with industrial research teams and middleware experts to guide next-generation component/middleware technology

... interacting with groups at Lockheed Martin, Boeing, and Rockwell-Collins, to develop techniques that match fit into development process

... collaborating with middleware experts (e.g., ACE/TAO RT-middleware) to make frameworks more amenable to model-based configuration and analysis
Cadena Development

Cadena I

...tailored to CCM

June 2002 – May 2005

Cadena II

...full meta-modeling (CCM, EJB, PRiSM, etc.) for product-line architectures

Jan 2005 – ...

Ask me later!
Avionics Mission Control Systems

- PCES/MoBIES Experimental Platform
- Mission-control software for Boeing military aircraft
- Boeing’s Bold Stroke Avionics Middleware
  - …built on top of ACE/TAO RT CORBA

- Provided with an Open Experimental Platform (OEP) from Boeing
  - a sanitized version of the real system
  - 100,000+ lines of C++ code (including RT CORBA middleware)
  - 50+ page document that outline development process and describe challenge problems

- Must provide…
  - tool-based solutions that can be applied by Boeing research team to realistic systems
  - solutions that fit within current development process, code base, etc.
  - metrics for that allow Boeing research team to evaluate tool performance and ease of use
Control-Push Data-Pull

Typical situation

Component A computes some data that is to be read by one or more components $B_i$

Run-time Actions

- A publishes a `dataAvailable` event
- $B_i$ call the `getData()` method of A to fetch the data

Depending on current state, component may not fetch data
1. Logical GPS component receives a periodic event indicating that it should read the physical GPS device.

2. Logical GPS publishes `DATA_AVAILABLE` event.

3. Airframe component fetches GPS data by calling GPS GetData method.

4. Airframe updates its position data and publishes `DATA_AVAILABLE` event.

5. NavDisplay component fetches AirFrame data by calling AirFrame GetData method.

6. NavDisplay updates the physical display.
Larger Configuration

...moving up to 1000+ components
Lack of Model Analysis

Boeing OEP Challenge Problems

1. Forward & backward data and event dependencies
2. Dependency intersections
3. Components with high data coupling
4. All components from a particular rate group
5. Cycle checks

...15-20 others related to dependencies
Lack of Model Analysis

If component 1 is in mode A when component 2 produces event E, then component 3 will consume event F (Section 4.1.5.3.6)

A temporal property well-suited for model-checking!
System Design Aspects

Inputs triggered @ different rates

60Hz
Sensor1

1Hz
Mode Switch

20Hz
Sensor1

5Hz
Sensor1

Intermediate components correlate incoming data and produce higher-level info

Declare rates/priorities for intermediate event handlers

Outputs required @ different rates

20Hz
Display

1Hz
Control

5Hz
Display

Map components to onboard network nodes

Implement mode semantics for changing subsystem behavior

Inputs triggered @ different rates

Sensor1

Off

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Sensor1

Segmentation error
No Unifying Mechanism

C++ Component Code

Deployment XML Configurator Info

Model-Driven Design, Analysis, Configuration

Integrated Development Environment

UML Design Artifacts

Analysis and QoS Aspect Synthesis
Example System
Example System

Basic components

Interface (Data) Ports
- facet (interface provided)
- receptacle (interface used)

Event Ports
- source (event published)
- sink (event consumed)

Event Correlation
- and-correlation
Example System

**Navigation Steering Subsystem**

- **Navigator**
- **NavSteeringPoints**
- **NavSteering**
  - Disabled
  - Enabled

**Complete formalism and auto code generation in Cadena**

- **BM PushDataSource**
- **BM Passive**
- **BM LazyActive**
- **BM Device GPS**
- **BM AirFrame**
- **BM Display**
- **BM ModeSource PilotControl**

**Event Ports**
- Facet (interface provided)
- Receptor (interface used)
- Source (event published)
- Sink (event consumed)

**Event Correlation**
- oaf correlation
Example System

Tactical Steering Subsystem
Cadena development environment enables model-based development of applications using frameworks such as CORBA Component Model (CCM).
Component IDL

```idl
component BMLazyActive {
    provides ReadData outData;
    uses ReadData inData;
    publishes DataAvailable outDataAvailable;
    consumes DataAvailable inDataAvailable;
    attribute LazyActiveMode dataStatus;
};
```
Component IDL

```
component BMLazyActive {
    provides ReadData outData;
    uses ReadData inData;
    publishes DataAvailable outDataAvailable;
    consumes DataAvailable inDataAvailable;
    attribute LazyActiveMode dataStatus;
};
```

output data port (facet)

CORBA 3
CCM IDL
ModalSP Components
Component IDL

```idl
component BMLazyActive {
  provides ReadData outData;
  uses ReadData inData;
  publishes DataAvailable outDataAvailable;
  consumes DataAvailable inDataAvailable;
  attribute LazyActiveMode dataStatus;
};
```

input data port (receptacle)

CORBA 3
CCM IDL
ModalSP Components
Component IDL

```idl
component BMLazyActive {
    provides ReadData outData;
    uses ReadData inData;
    publishes DataAvailable outDataAvailable;
    consumes DataAvailable inDataAvailable;
    attribute LazyActiveMode dataStatus;
};
```

output event port (event source)

CORBA 3
CCM IDL
ModalSP Components
Component IDL

code snippet:

```idl
component BMLazyActive {
    provides ReadData outData;
    uses ReadData inData;
    publishes DataAvailable outDataAvailable;
    consumes DataAvailable inDataAvailable;
    attribute LazyActiveMode dataStatus;
};
```

CORBA 3
CCM IDL
ModalSP Components
Component IDL

```idl
component BMLazyActive {
    provides ReadData outData;
    uses ReadData inData;
    publishes DataAvailable outDataAvailable;
    consumes DataAvailable inDataAvailable;
    attribute LazyActiveMode dataStatus;
};
```

CORBA 3
CCM IDL
ModalSP Components
Component Development

- Development of component interfaces using CCM Interface Definition Language
- Automatic generation of component infrastructure code using CCM IDL compilers
- Development of core functional code (business logic) using Eclipse Java facilities
Outline

2. Component Connections
Three Synchronized Views

Scenario Description

Graphical View

Spreadsheet View

Textual View

Single Internal Representation
Instance AirFrame implements BMLazyActive on Board2 {
    connect this.inDataAvailable to GPS.outDataAvailable;
    connect this.dataIn to GPS.dataOut;
}

connect this.timeOut to EventChannel.timeOut20 atRate 20;

Instance AirFrame implements BMLazyActive on Board2 {
    connect this.inDataAvailable to GPS.outDataAvailable;
    connect this.dataIn to GPS.dataOut;
}

Instance DisplayCorrelator implements Correlator on Board3 {
    connect inDataAvailable to DisplayCorrelator.outDataAvailable;
    connect inDataAvailable to NavSteering.outDataAvailable;
    connect inDataAvailable to TacticalSteering.outDataAvailable;
}

Instance Display implements BMDisplay on Board3 {
    connect this.inDataAvailable to DisplayCorrelator.outDataAvailable;
    connect this.dataIn to AirFrame.dataOut;
    connect this.dataIn2 to TacticalSteering.dataOut;
    connect this.dataIn3 to NavSteering.dataOut;
}

Instance PilotControl implements BMModeSource on Board4 {
    connect this.timeOut to EventChannel.timeOut1 atRate 1;
    connect this.dataOut to bmBmp1.dataOut;
    connect this.dataOut to bmBmp2.dataOut;
    connect this.dataOut to bmBmp3.dataOut;
}

Instance NavSteering implements BMModal1 on Board5 {
    connect this.inDataAvailable to AirFrame.outDataAvailable;
}
..allocate AirFrame component instance

```
Instance AirFrame implements BMLazyActive on Board2 {
    connect this.inDataAvailable to GPS.outDataAvailable;
    connect this.dataIn to GPS.dataOut;
}
```
...connect event ports and facet/receptacles
Graphical View
Graphical View

...design-level analyses mode-
base views
### Spreadsheet View

#### Port Types

<table>
<thead>
<tr>
<th>Port Type</th>
<th>Location</th>
<th>Rate (Hz)</th>
<th>Connected to</th>
<th>Contained in</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>timeOut</code></td>
<td></td>
<td>20</td>
<td><code>&lt; EventChannel</code></td>
<td></td>
</tr>
<tr>
<td><code>outDataAvailable</code></td>
<td></td>
<td>20</td>
<td><code>&gt; inDataAvailable</code></td>
<td><code>AirFrame</code></td>
</tr>
<tr>
<td><code>dataOut</code></td>
<td></td>
<td>0</td>
<td><code>&gt; dataIn</code></td>
<td><code>AirFrame</code></td>
</tr>
</tbody>
</table>

#### Port Connections

- **Board 1**
  - `timeOut` connected to `EventChannel`
  - `outDataAvailable` connected to `inDataAvailable` in `AirFrame`
  - `dataOut` connected to `dataIn` in `AirFrame`

- **Board 2**
  - `timeOut` connected to `EventChannel`
  - `outDataAvailable` connected to `inDataAvailable` in `Display`
  - `dataOut` connected to `dataIn` in `Display`

- **Board 3**
  - `timeOut` connected to `EventChannel`
  - `outDataAvailable` connected to `inDataAvailable` in `TacticalSteering`
  - `dataOut` connected to `dataIn` in `TacticalSteering`

- **Board 4**
  - `timeOut` connected to `EventChannel`
  - `outDataAvailable` connected to `inDataAvailable` in `Control`
  - `dataOut` connected to `dataIn` in `Control`

- **Board 5**
  - `timeOut` connected to `EventChannel`
  - `outDataAvailable` connected to `inDataAvailable` in `TacticalSteering`
  - `dataOut` connected to `dataIn` in `TacticalSteering`

- **Board 6**
  - `timeOut` connected to `EventChannel`
  - `outDataAvailable` connected to `inDataAvailable` in `Display`
  - `dataOut` connected to `dataIn` in `Display`

- **Board 7**
  - `timeOut` connected to `EventChannel`
  - `outDataAvailable` connected to `inDataAvailable` in `Navigator`
  - `dataOut` connected to `dataIn` in `Navigator`
Spreadsheet View

Pull-down menus give type-correct connection possibilities

Value Added: Incremental, iterative scenario construction with multiple forms of visualization, analyses, and automated “design advice”. 
Packaging & Deployment

CCM XML-based Configuration and Deployment information

CCM Deployment Infrastructure (e.g., CIAO, OpenCCM)
Cadena supports…

Integration with underlying component middleware frameworks (e.g., CIAO, OpenCCM) through well-defined interfaces (plug in your favorite CCM framework)

Component Interfaces and implementations

Component Assemblies

…but the real power of model-driven development comes from…

- decorating component interfaces and assemblies with a variety of forms of behavioral contracts and meta-data that can be leveraged by model-level analysis and optimizations
- these analyses/optimization may be general purpose, or tailored to specific product-lines or products
Questions

- What types of light-weight semantic specifications have we found useful?
- What forms of automated design advice can be provided?
- What sorts of automatic optimization/customization can be performed?
- How does the product-line designer define organize & reuse these specs and information?
Leverage CORBA IDL

```
component BMLazyActive {
  provides ReadData outData;
  uses ReadData inData;
  publishes DataAvailable outDataAvailable;
  consumes DataAvailable inDataAvailable;
  attribute LazyActiveMode dataStatus;
};
```

Dependency Annotations

Transition System Semantics

IDL Compiler

Component Implementation
Stubs & Skeletons

Model Builder

Dependency Analysis
and Model-checking Engine
Incremental Specification

- port action dependencies
- state-based dependencies
- component transition semantics

Lockheed-Martin collaboration: extract/connect this info from MSCs

...only in mode Y

...state machines give abstract behavior
Light-weight Dependency Specs

dependency default == none;

dependencies {
    dataWriteOut.set_data() -> outDataAvailable;
}

behavior {
    call on set_data();
    triggers
    outDataAvailable port action
}
Light-weight Dependency Specs

```
dependency default == all;

dependencies {
  modeChange() ->;
  case modeChange.modeVar of {
    enabled: inDataAvailable -> dataIn.get_data(),
    outDataAvailable;
    disabled: inDataAvailable ->;
  }
}

behavior { ... }
```
Light-weight Dependency Specs


dependency default == all;

dependencies {
  modeChange() - >;
  case modeChange. modeVar of {
    enabled: inDataAvailable - > dataIn.get_data(), outDataAvailable;
    disabled: inDataAvailable - >;
  } }

behavior { ... }
Light-weight Dependency Specs

dependencydefault = all;

dependencies {
    modeChange() ->;
    case modeChange.modeVar of {
        enabled: inDataAvailable -> dataIn.get_data(),
                      outDataAvailable;
        disabled: inDataAvailable ->;
    }
}

behavior { ... }
Design-Level Flow/Dependence Graphs

- From system configuration information
- From user-specified intra-component dependences
- State predicates giving conditional dependences
Connecting Models & Code

Model-level Specification/Abstraction

...use Indus to automatically check that code dependences satisfy specification

SAnToS Indus Dependence Calculation (Java Program Slicing)
Connecting Models & Code

Model-level Specification/Abstraction

...use Indus to automatically synthesize (reverse engineer) model-level dependences from code

Java Implementation

SAnToS Indus Dependence Calculation (Java Program Slicing)
Goal: Simplify visualization of system by hiding component/connections in various modes.

Scenario Diagram w/ Complete Connectivity

Projections of Systems with Enabled Connectivity for Different Modes
Mode-based Projections

Value Added: Multiple *mode-based views* are automatically created and synchronized through the design process.
Cadena profiles enable flexible definition of attributes for CCM model entities and APIs for plug-in tools to access and manipulate attribute values.
Spreadsheet View

- Product-line/Application-specific attributes
- Ports for component type
- Port types
- RT Attributes
- Port connections
- Distribution sites
- Rate group
Automated Design Advice

Dependency-driven priority/rate assignment to event handlers

- **Bold Stroke product-line architecture** contains all threads in RT CORBA event channel
  - rate monotonic scheduling, so a group of threads where each thread has a certain priority/rate
- Each event handler has a model-level attribute to indicate the priority/rate of the thread that should run the handler

*developer must select attribute value*...
Automated Design Advice

Dependency-driven priority/rate assignment to event handlers

Heuristics

...time out handlers run at rate associated with time out event
...when a handler is dependent on the actions of handler X, it should run at same rate as X

...plug-in leverages dependency information and automates the heuristics to seed rates for handlers
Results of automatic rate group synthesis are fed back into spreadsheet.
Automated Design Advice

Selecting event transport facility

- Many systems may have multiple mechanisms/services or protocols for propagating events
- For example…
  - through full real-time event channel
  - through ORB
  - ...optimizations of the above
- Direct dispatch (bypass ORB and event channel) optimization conditions
  - publisher and subscriber
    - run at same rate
    - co-located
  - no correlation involved
Automated Design Advice

Automatic selection of event transport mechanism

Asynchronous message delivery to synchronous method calls (must be co-located and run at same rate)
Overall Theme

Analyze & Leverage Annotated Models

Automated design advice for choosing middle and service parameters

Verification of system invariants, crucial event sequencing constraints, etc.

Annotated Models

Collect & Expose Attributes

...collect crucial system info at a manageable level of abstraction

Collect & Expose Attributes

Code level static analysis

Middleware and service configuration parameters

Data-collected by run-time monitoring

Event Service

CORBA Middleware

Naming Service

Synchronization Service

CCM Component

Code level static analysis

Data-collected by run-time monitoring
Boeing Tech Transition

White Sands Missile Range Live Fire Demo (April 2005)

- Two Scan Eagle UAVs
- HIMARS Missile System
- Dynamically Balanced QoS Streaming Video

- Cadena used to develop RT-Java flight control software for Scan Eagles
- Scan Eagles sent streaming video to integrated command/control for launching HIMARS rocket
Concluding Themes

Lifting programming to a higher level of abstraction

- Huge amount of code auto-generated from IDL and model developed structures
- Component designers can be more effectively “compartmentalized” and isolated from complexities of entire system
- *Component integrators* play a key role in correct system construction and need to be supported by a variety of visualizations, structural and semantics analyses
Concluding Themes

“Aspectize” a variety of programming system elements and configure those at the modeling level

- Examples in Cadena…
  - locking strategies, choice of communication layers, configuration/placement of event channel and other service infrastructure, deployment information

- Provide flexible mechanisms to allow component integrators to select from pre-identified values/strategies for these aspects

- Use automated analysis on models of both the system and target platforms (hardware, OS) to automatically synthesize values of these attributes, or at least suggest values as design advice
Concluding Themes

Structural analyses with lightweight semantic information can go a long way

- Many examples in Cadena of model/analysis-driven configuration are based on the simple declaration of intra-component dependences with mode information
- Easy for developers to specify
- Fairly easy to reverse engineer from existing code (using program slicing techniques)
- Easy to check that code conforms to model-based specifications
- Provides a gentle transition path to richer specifications with clear notions of refinement
For More Information...

SAnToS Laboratory,
Kansas State University
http://www.cis.ksu.edu/santos

Cadena Project
http://cadena.projects.cis.ksu.edu

…web-site provides distribution that is integrated with CIAO and OpenCCM, examples, user-manual, tutorial, etc.