

Model-Driven Development of Real-Time Embedded Systems in Boeing's BoldStroke Product-Line

SAnToS Laboratory, Kansas State University, USA

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

John Hatcliff

William Deng

Venkatesh Ranganath

Matt Dwyer

Georg Jung

Robby

Jesse Greenwald

Gurdip Singh

Adam Childs

Prashant Shanti Kumar

Matt Hoosier

Support

US Army Research Office (ARO)

US National Science Foundation (NSF)

US Department of Defense

Advanced Research Projects Agency (DARPA-PCES)

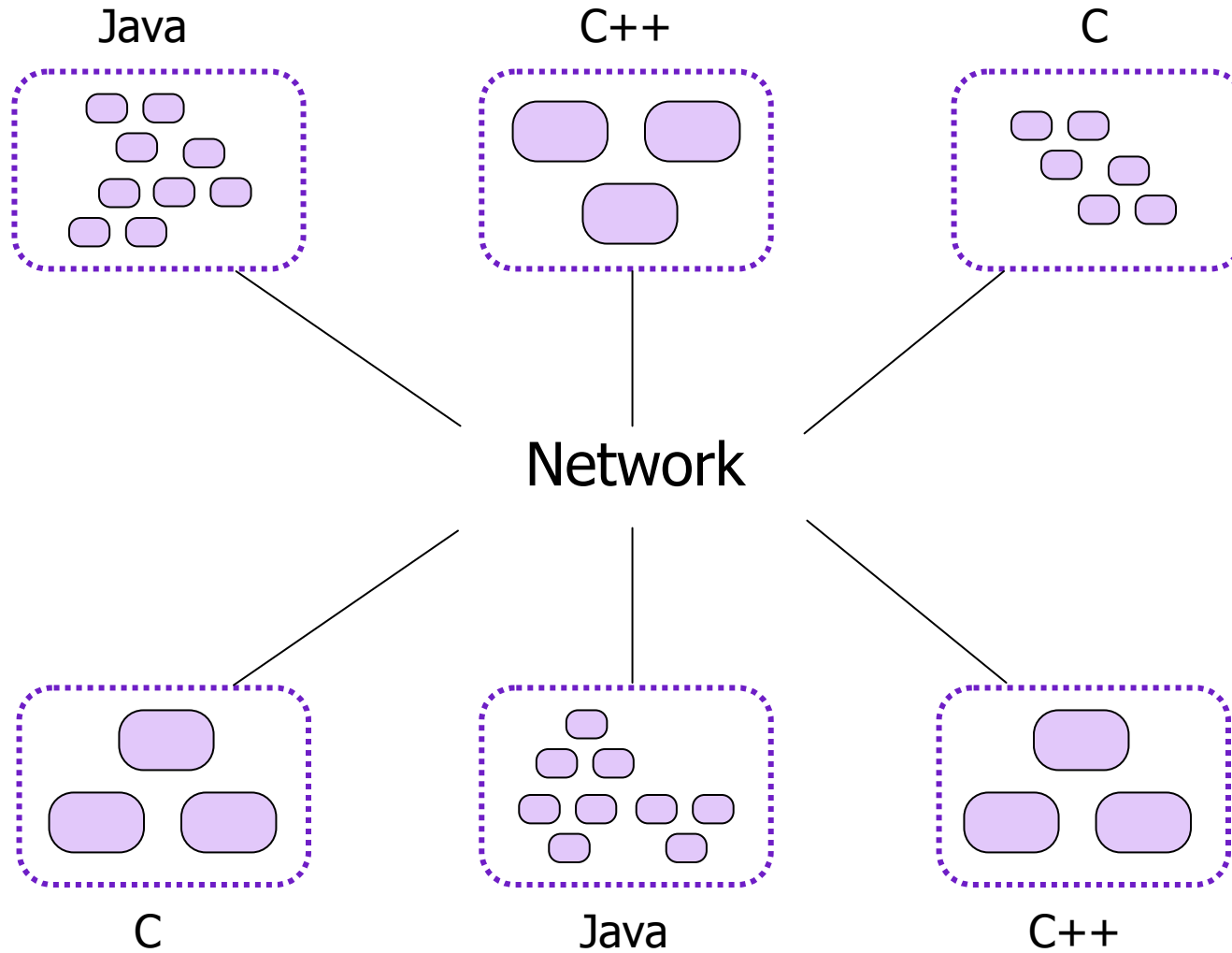
Rockwell-Collins ATC

Lockheed-Martin [Eagan] – (PCES sub-contract)

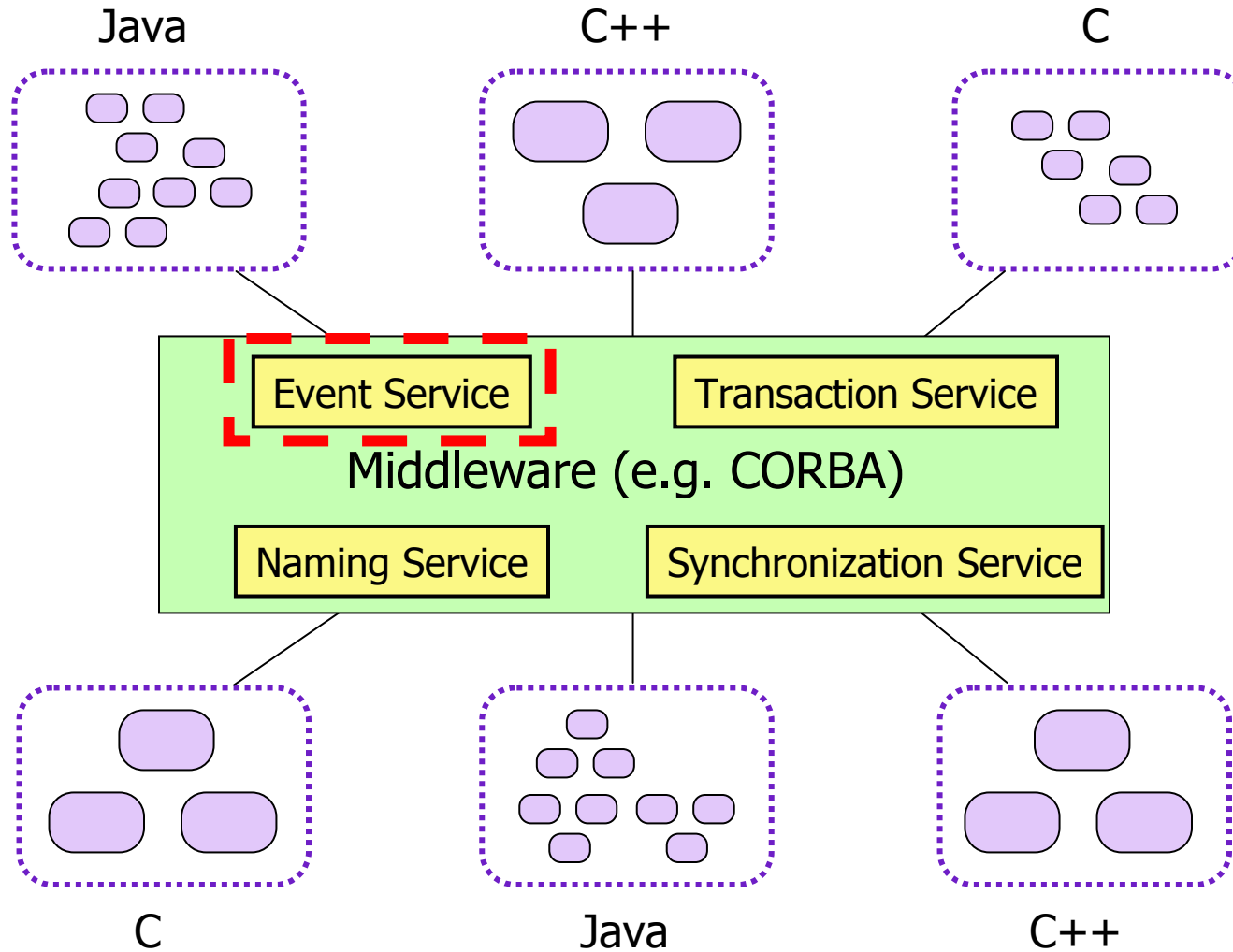
Honeywell Technology Center and NASA Langley

IBM, Intel, Sun Microsystems

Distributed Components



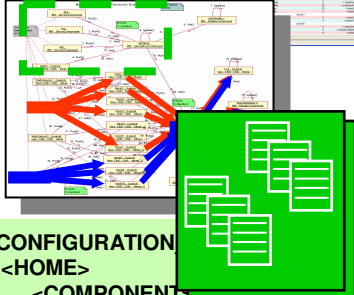
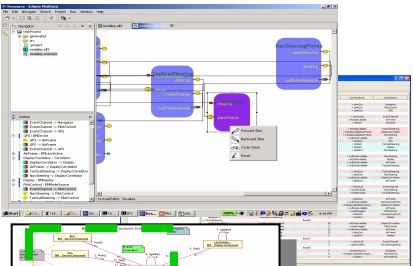
Distributed Components




Cadena Development

Cadena I

...tailored to CCM



```
<CONFIGURATION  
<HOME>  
<COMPONENT  
<ID> <...></ID>  
<EVENT_SUPPLIER>  
<...events this component  
supplies...>  
</EVENT_SUPPLIER>  
</COMPONENT>  
</HOME>  
</CONFIGURATION_PASS>
```

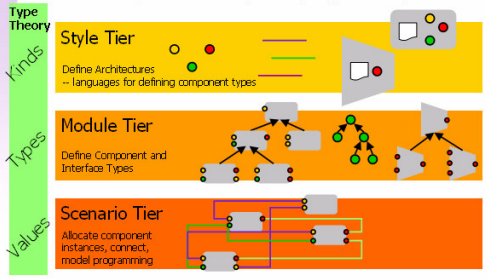
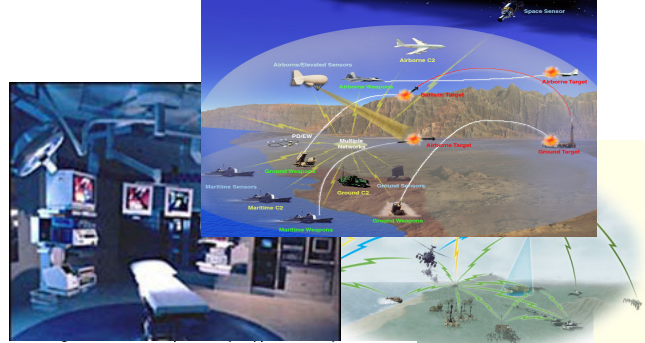


June 2002 – May 2005

The focus of this talk

Cadena II

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



Type Theory

Yields

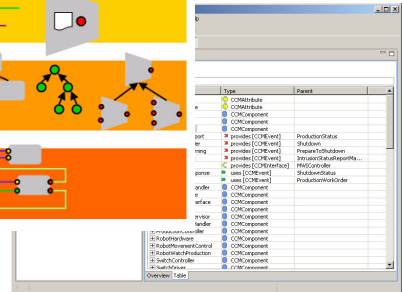
Types

Values

Style Tier
Define Architectures
– languages for defining component types

Module Tier
Define Component and
Interface Types

Scenario Tier
Allocate component
instances, connect,
model programming



Jan 2005 – ...

Avionics Mission Control Systems

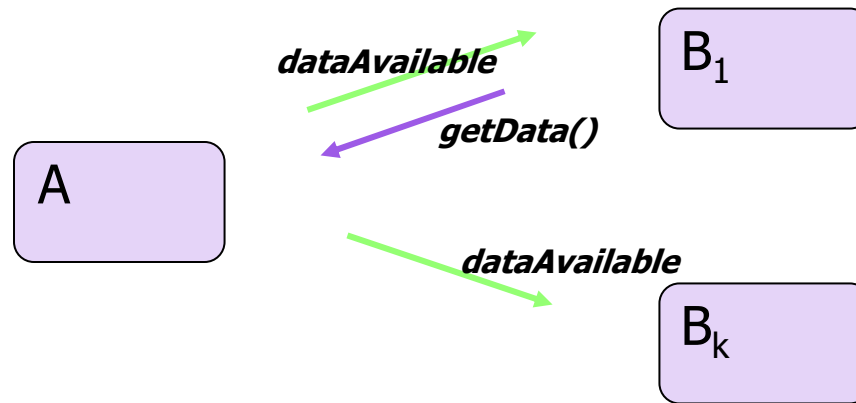


- PCES/MoBIES Experimental Platform
 - Mission-control software for Boeing military aircraft
 - Boeing's Bold Stroke Avionics Middleware
 - ...built on top of ACE/TAO RT CORBA
-
- Provided with an Open Experimental Platform (OEP) from Boeing
 - a sanitized version of the real system
 - 100,000+ lines 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



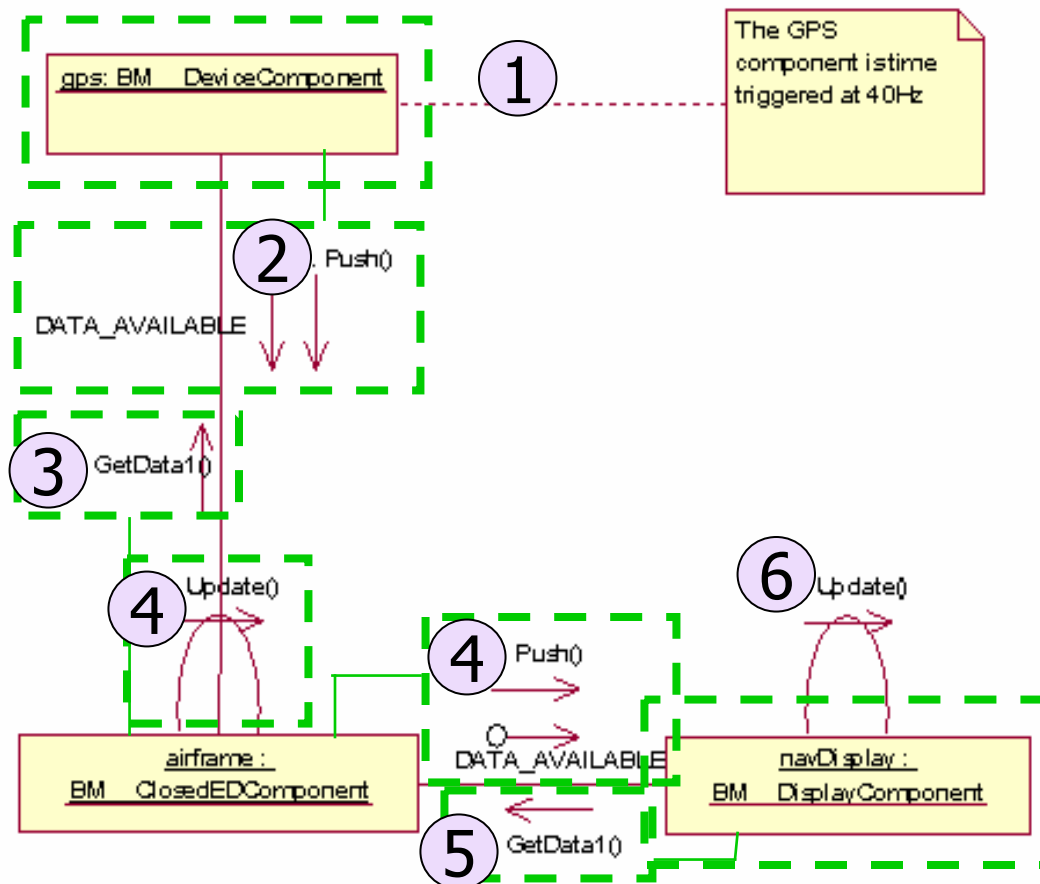
Run-time Actions

- 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

Control-Push Data-Pull Structure

Input



Output

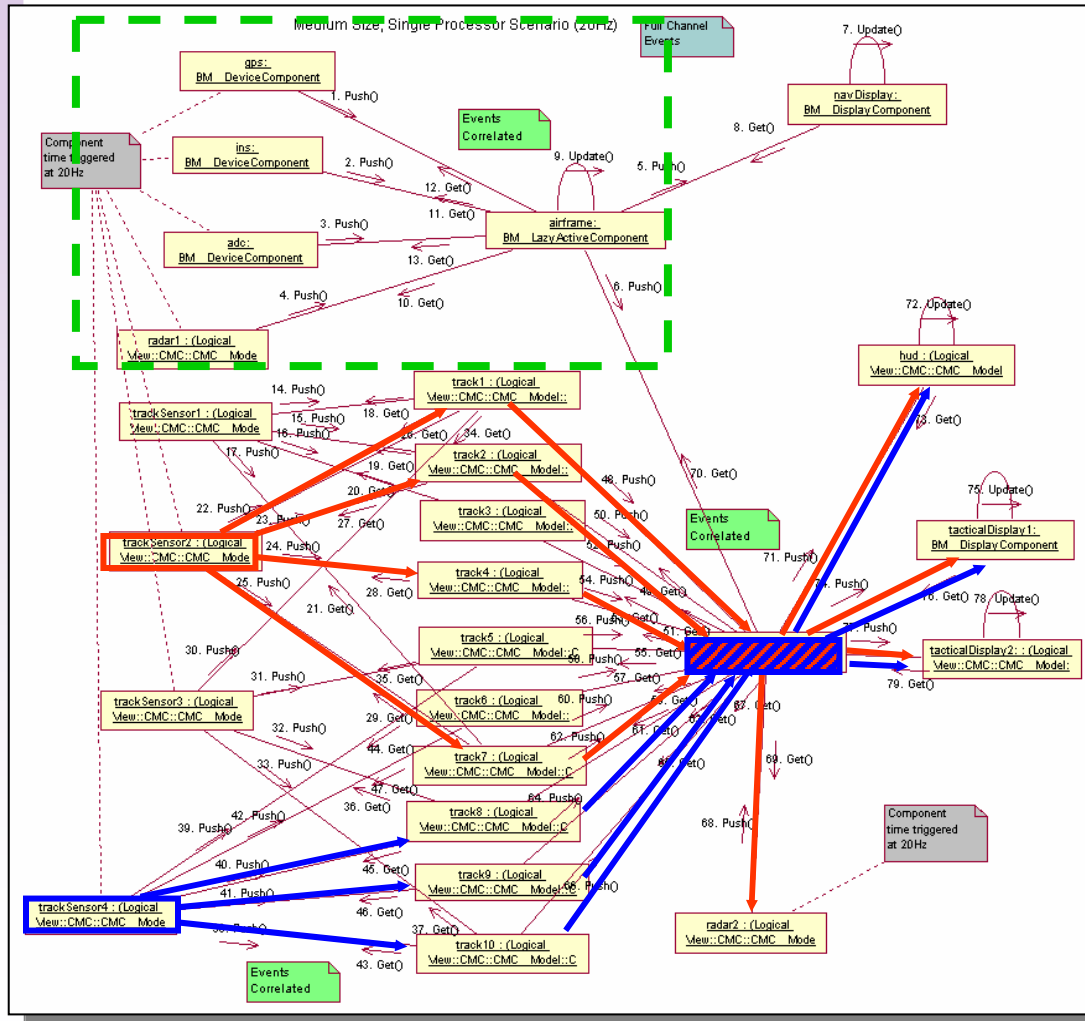
1. Logical GPS component *receives a periodic event* indicating that it should read the physical GPS device.
2. Logical GPS *publishes DATA_AVAILABLE event*
3. Airframe component *fetches GPS data* by calling GPS `GetData` method
4. Airframe *updates its position data* and publishes `DATA_AVAILABLE` event
5. NavDisplay component *fetches AirFrame data* by calling AirFrame `GetData` method
6. NavDisplay *updates the physical display*

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



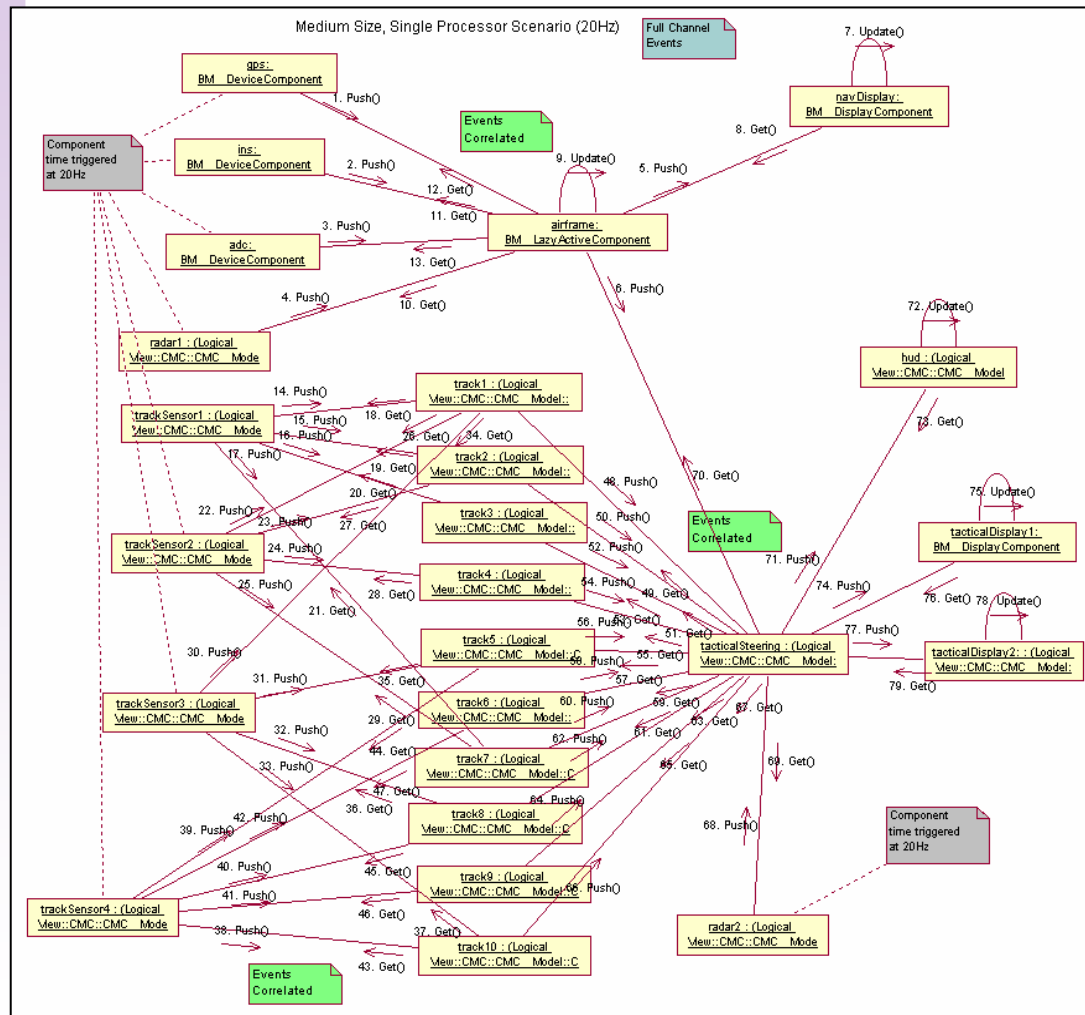
Lack of Model Analysis

Boeing OEP Challenge Problems

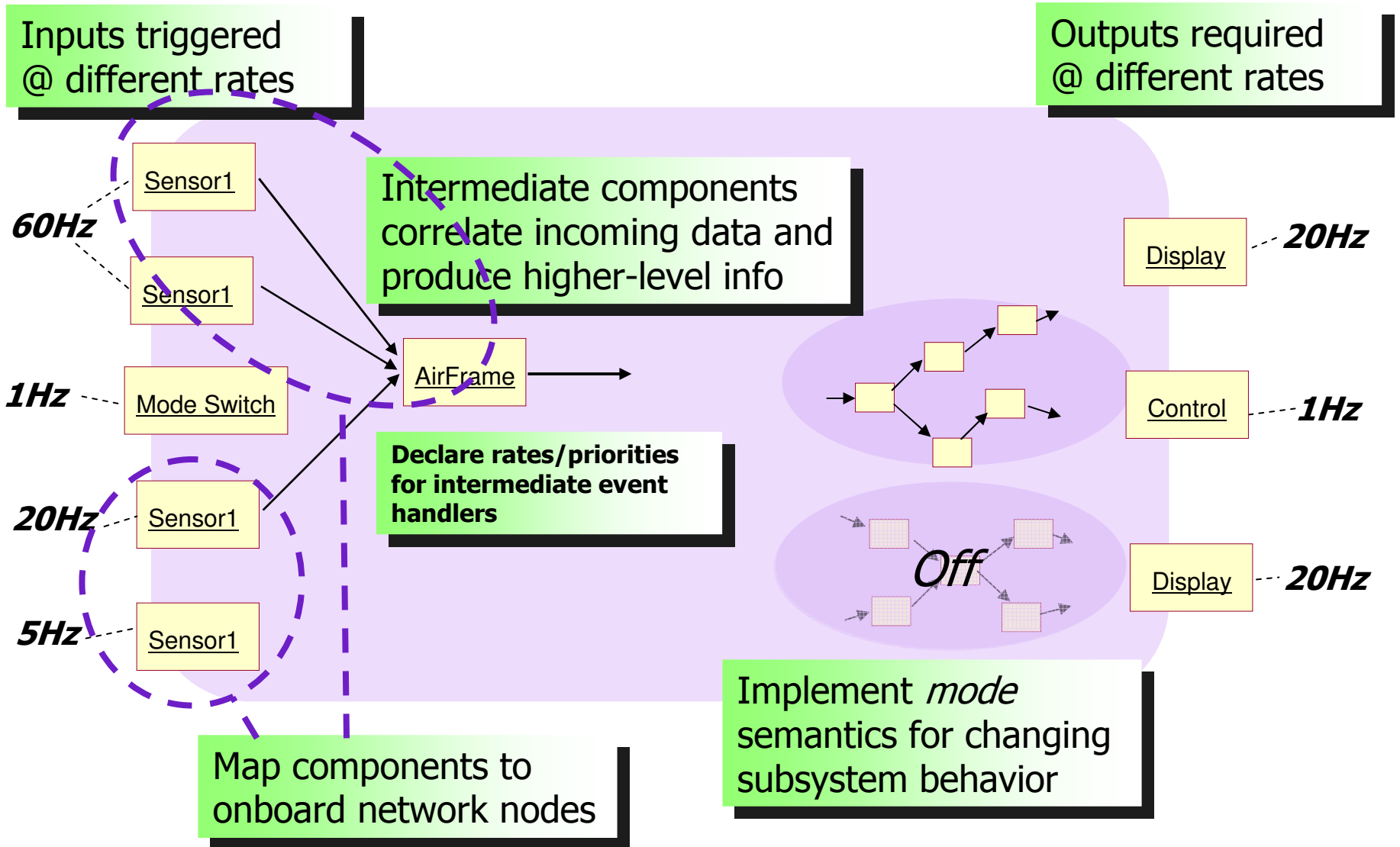
If component 1 is in mode A when component 2 produces event E, then component 3 will consume event F

(Section 4.1.5.3.6)

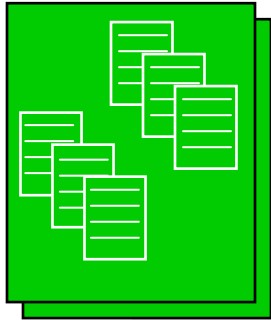
A temporal property well-suited for model-checking!



System Design Aspects



No Unifying Mechanism



C++
Component Code

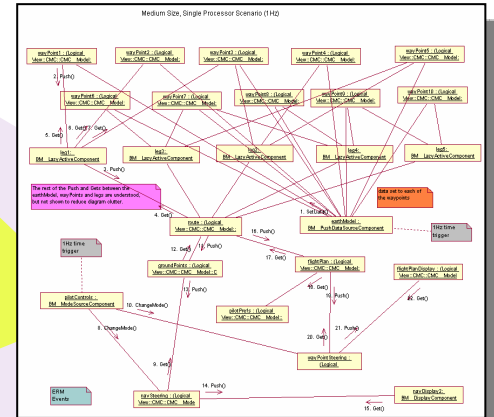
```
<CONFIGURATION_PASS>  
<HOME> <...>  
<COMPONENT>  
<ID> <...></ID>  
<EVENT_SUPPLIER>  
<...events this component supplies...>  
</EVENT_SUPPLIER>  
</COMPONENT>  
</HOME>  
</CONFIGURATION_PASS>
```

Deployment XML
Configurator Info

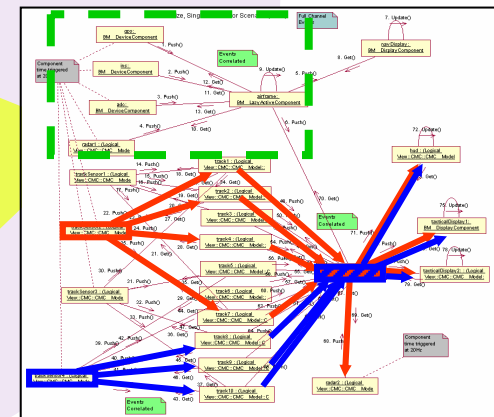


Model-Driven
Design, Analysis,
Configuration

Integrated Development
Environment

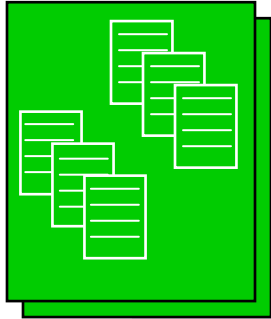


UML Design Artifacts



Analysis and QoS
Aspect Synthesis

Cadena



Java/C++
Component Code

```
<CONFIGURATION_PASS>  
<HOME> <...>  
<COMPONENT>  
<ID> <...></ID>  
<EVENT_SUPPLIER>  
<...events this component supplies...>  
</EVENT_SUPPLIER>  
</COMPONENT>  
</HOME>  
</CONFIGURATION_PASS>
```

Bold Stroke XML
Configurator Info

Cadena

CCM Interface
Definition
Language

System
Assembly Support

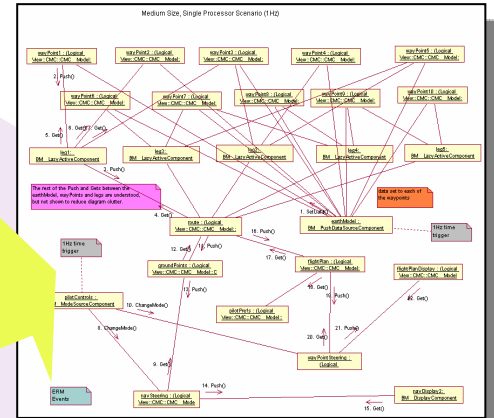
Light-weight
Semantic Specs

Analyses

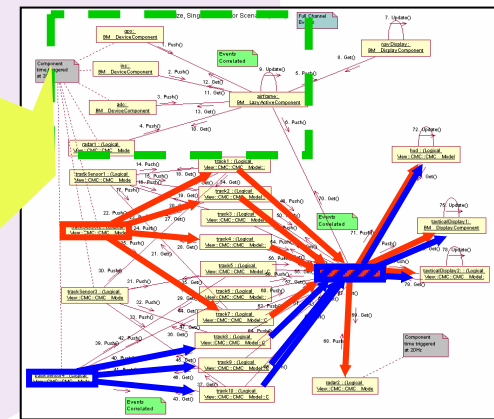
CIAO/TAO

Eclipse Plug-In

Integrated Development
Environment

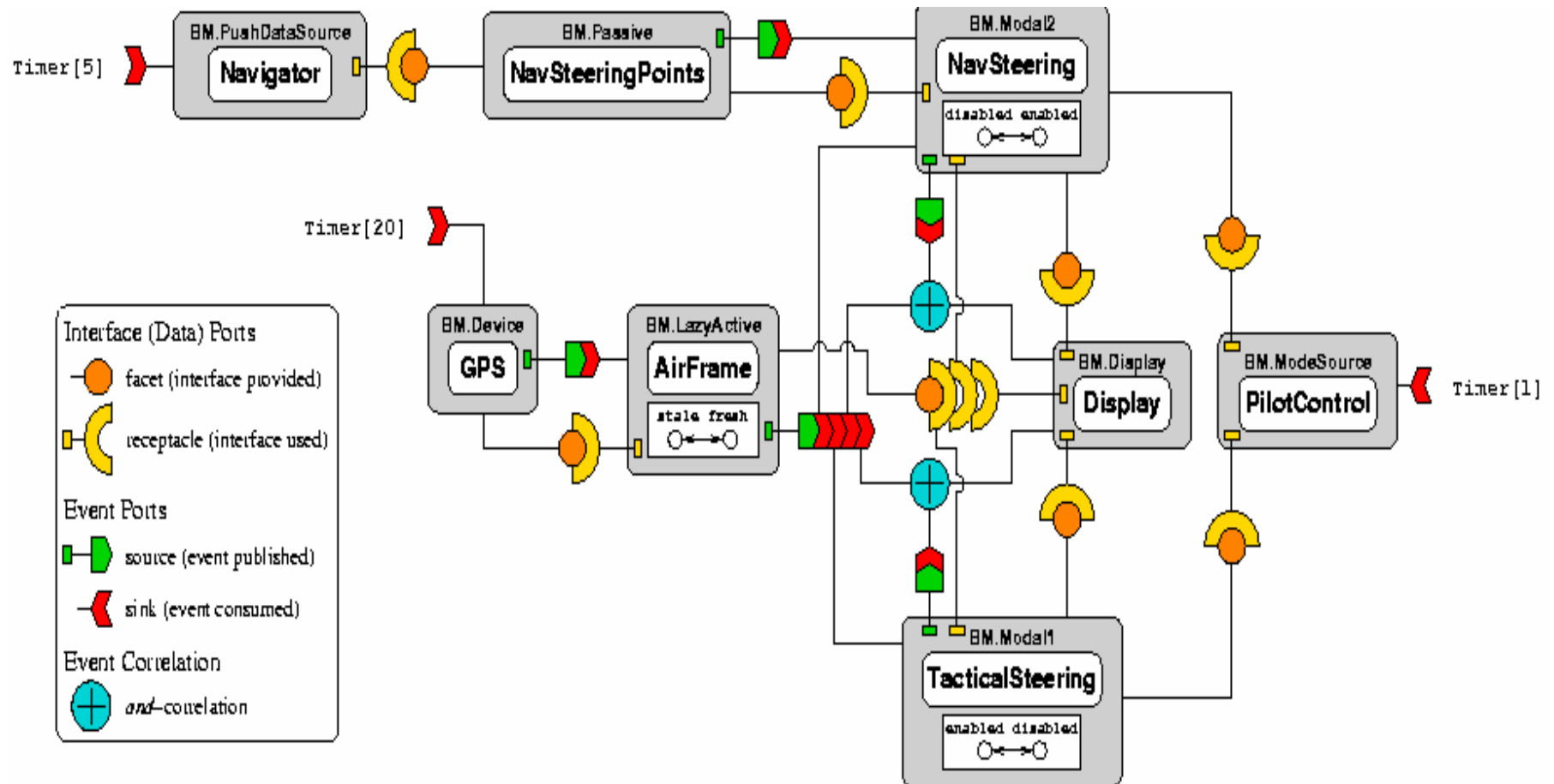


UML Design Artifacts

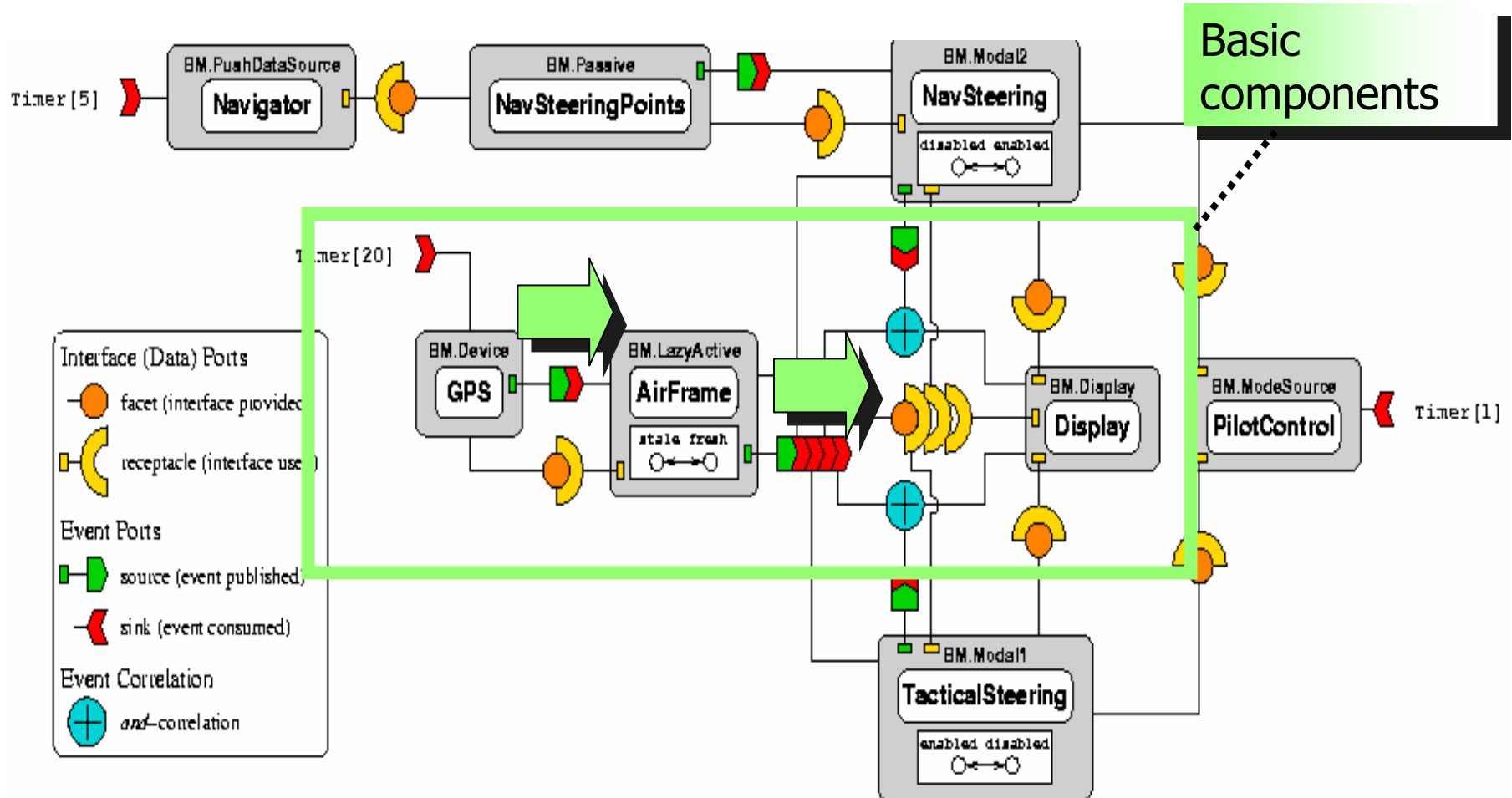


Analysis and QoS
Aspect Synthesis

Example System

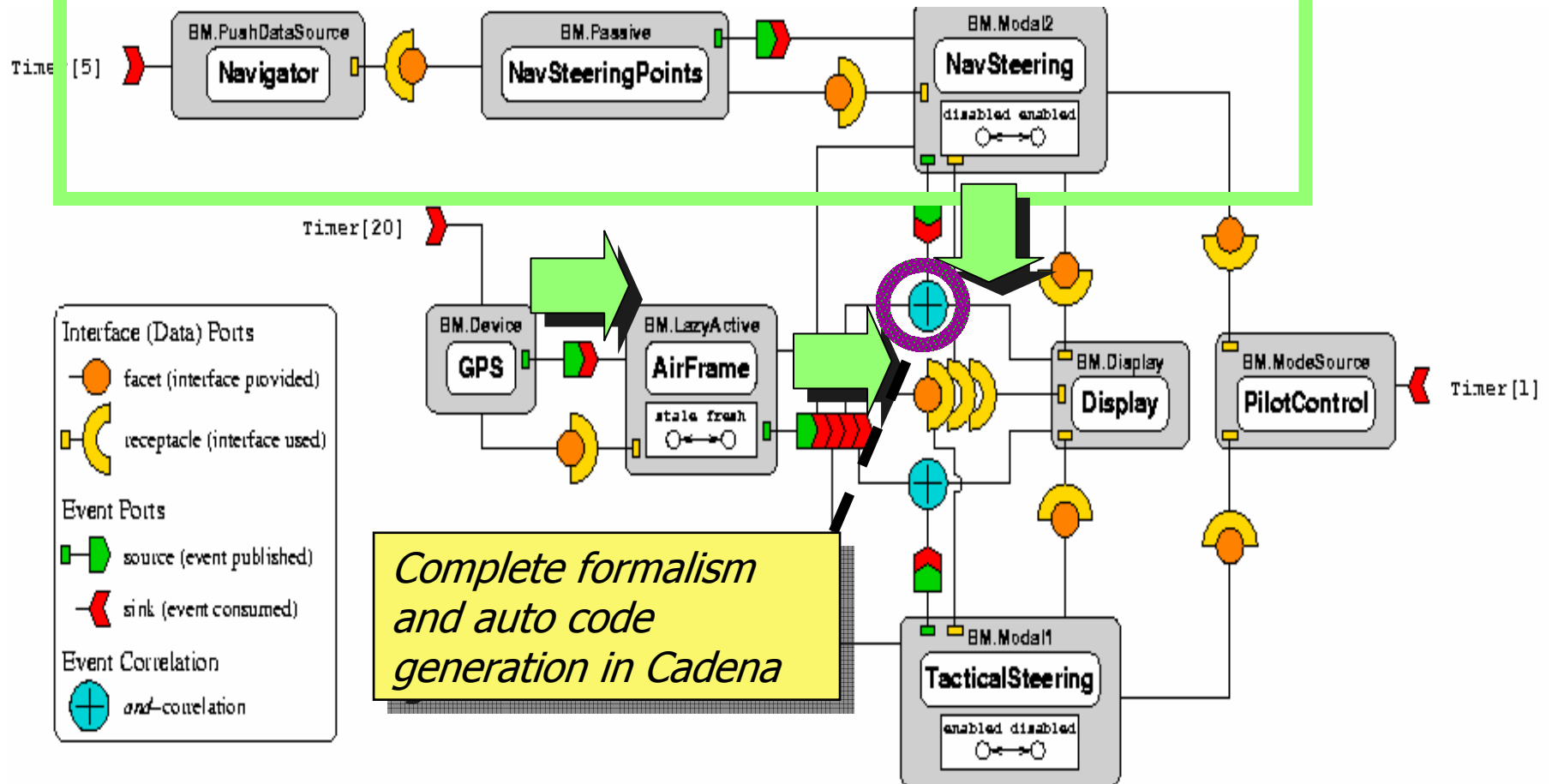


Example System



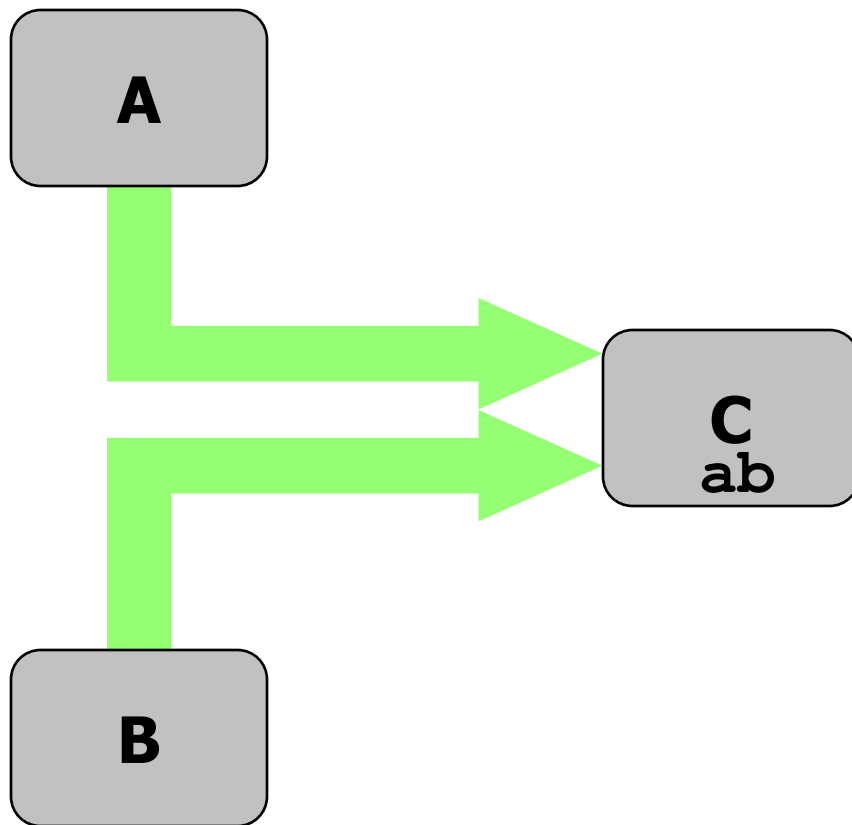
Example System

Navigation Steering Subsystem



Challenges of Event Communication

Consider the following situation:

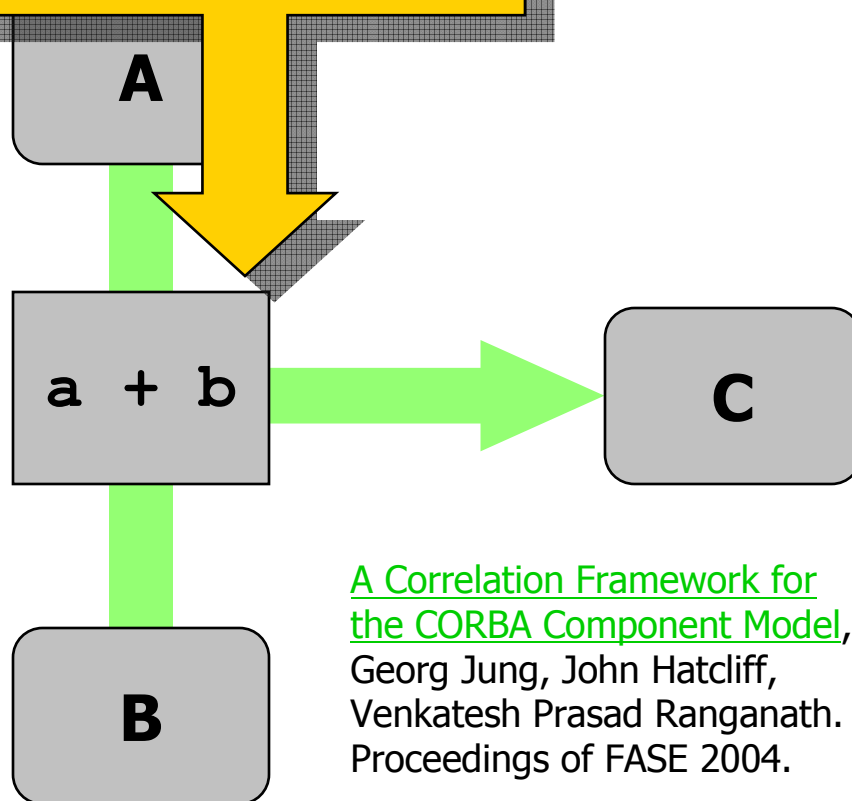


- Component **C** is receiving from two components **A** and **B**
- **A** and **B** send at different rates
- **C** needs both inputs to become active

Challenges of Event Communication

If we add **correlations** to the infrastructure

the following situation:

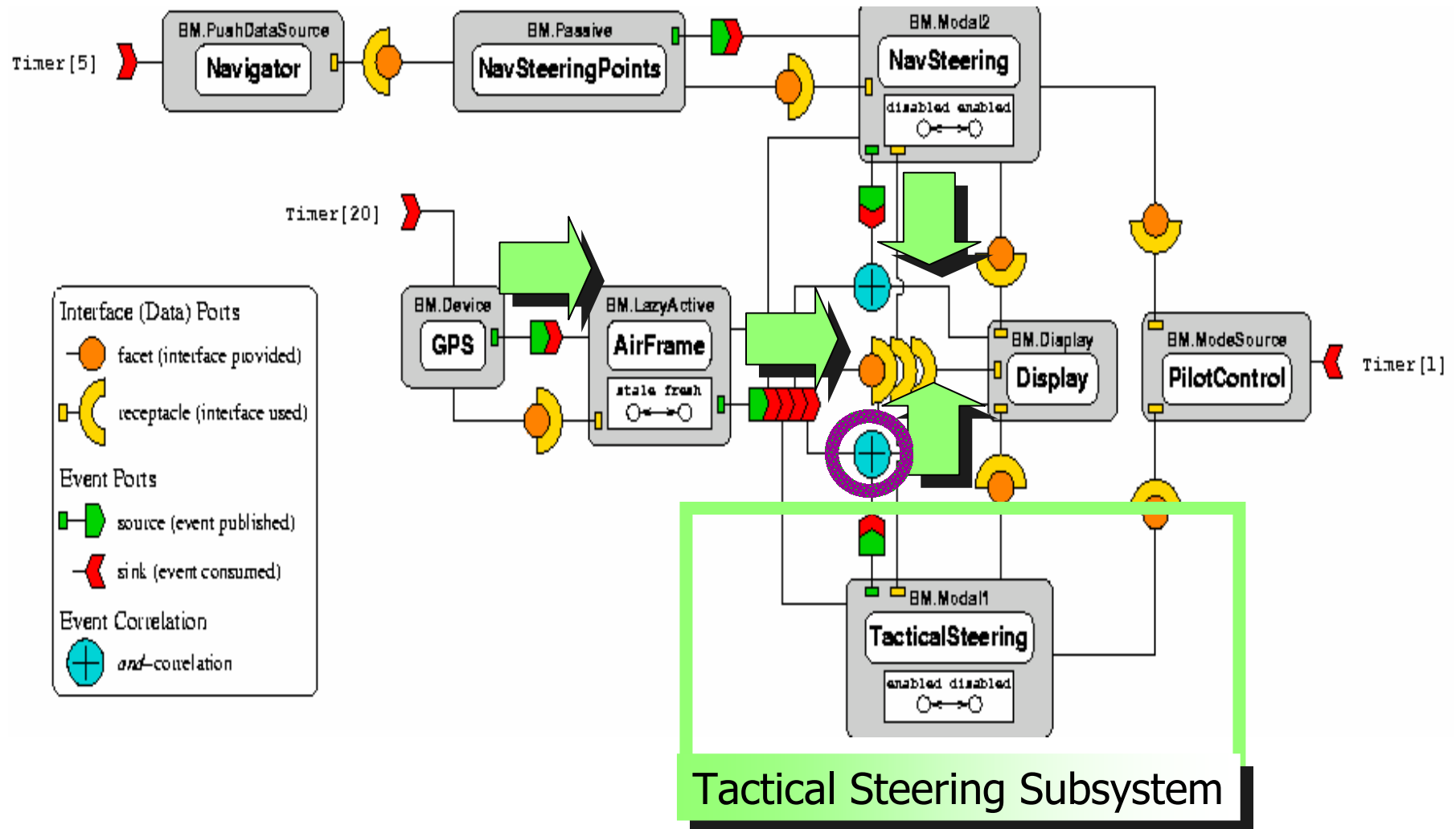


[A Correlation Framework for the CORBA Component Model](#),
Georg Jung, John Hatcliff,
Venkatesh Prasad Ranganath.
Proceedings of FASE 2004.

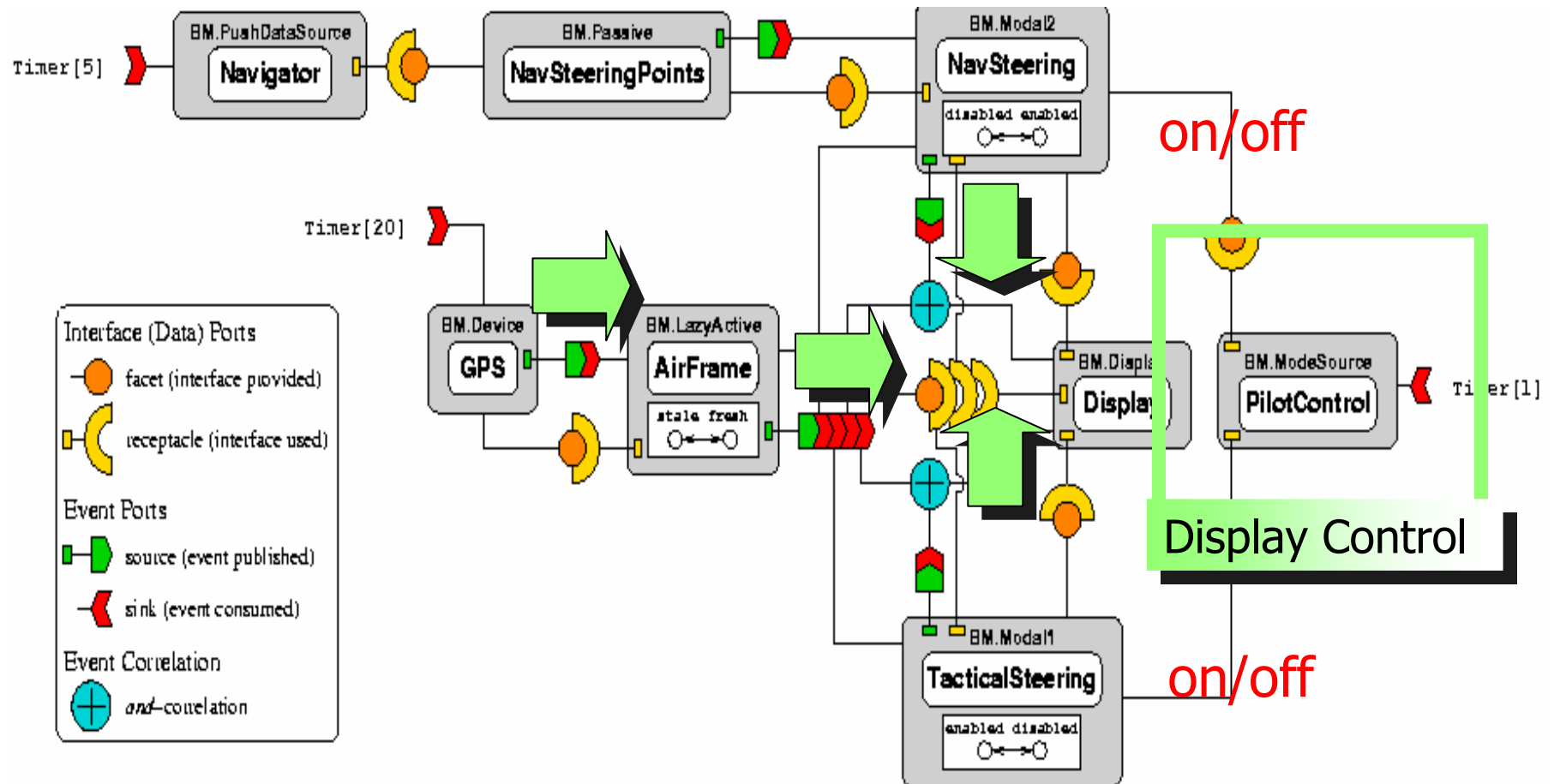
We can:

- Reduce network traffic
- Simplify computation inside the component
- Clarify the design
- Define the components in a more general way

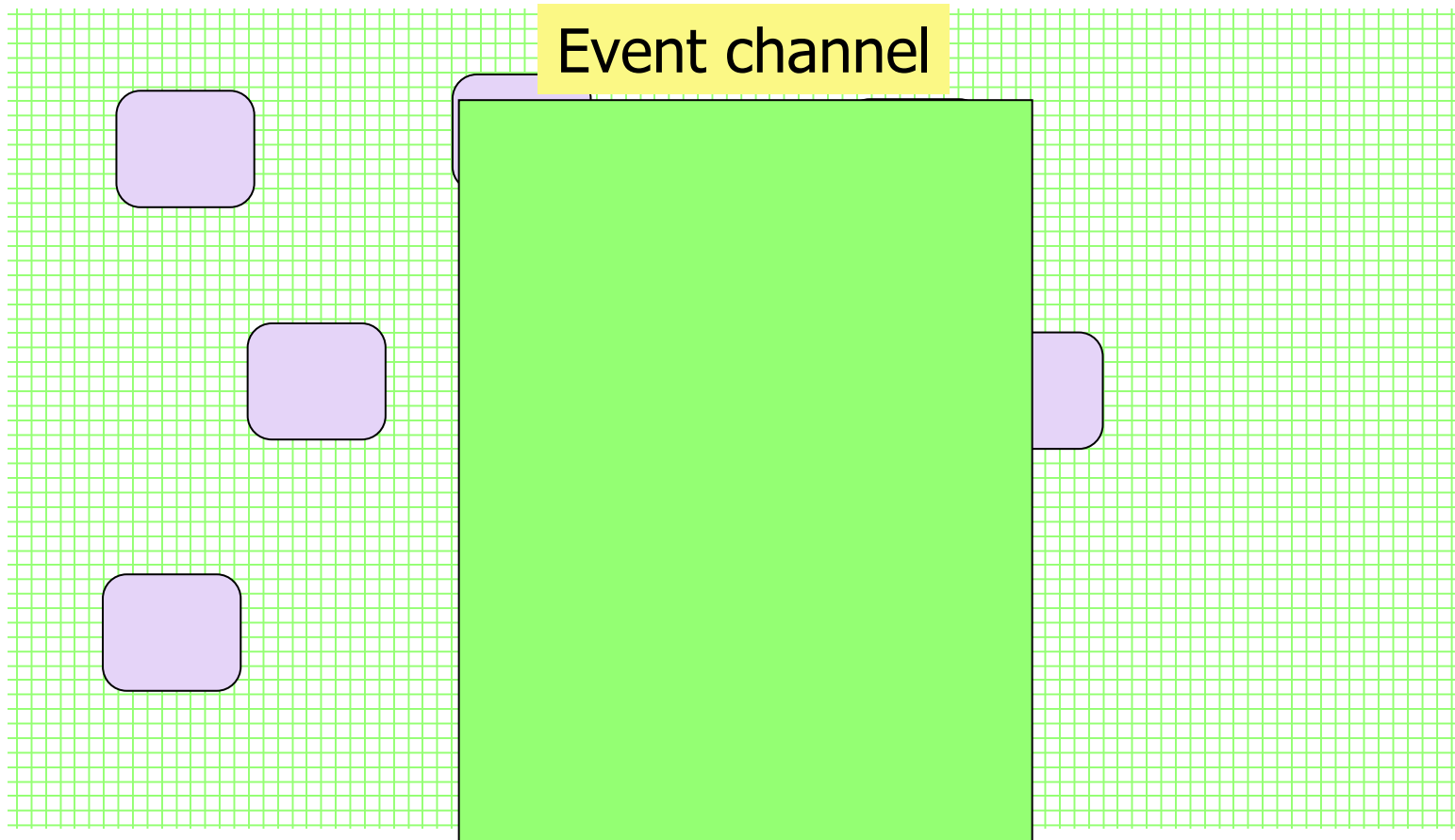
Example System



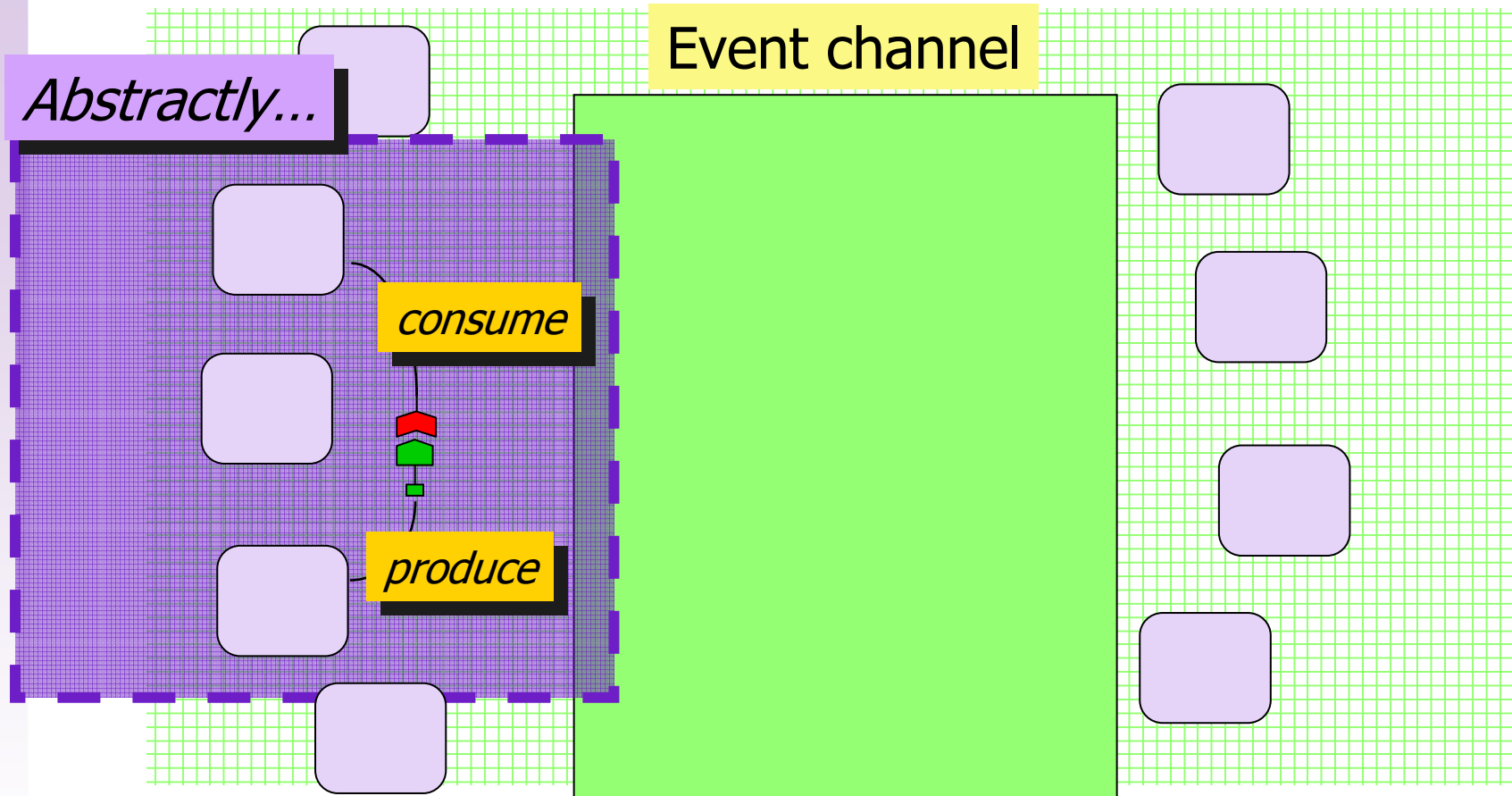
Example System



RT Event Channel Infrastructure



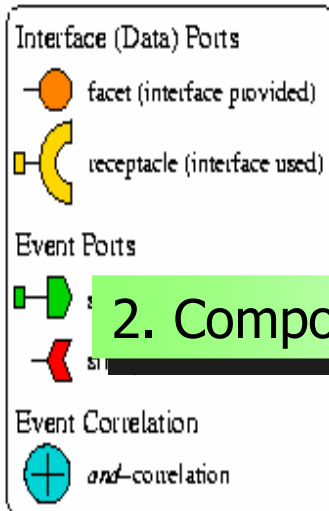
RT Event Channel Infrastructure



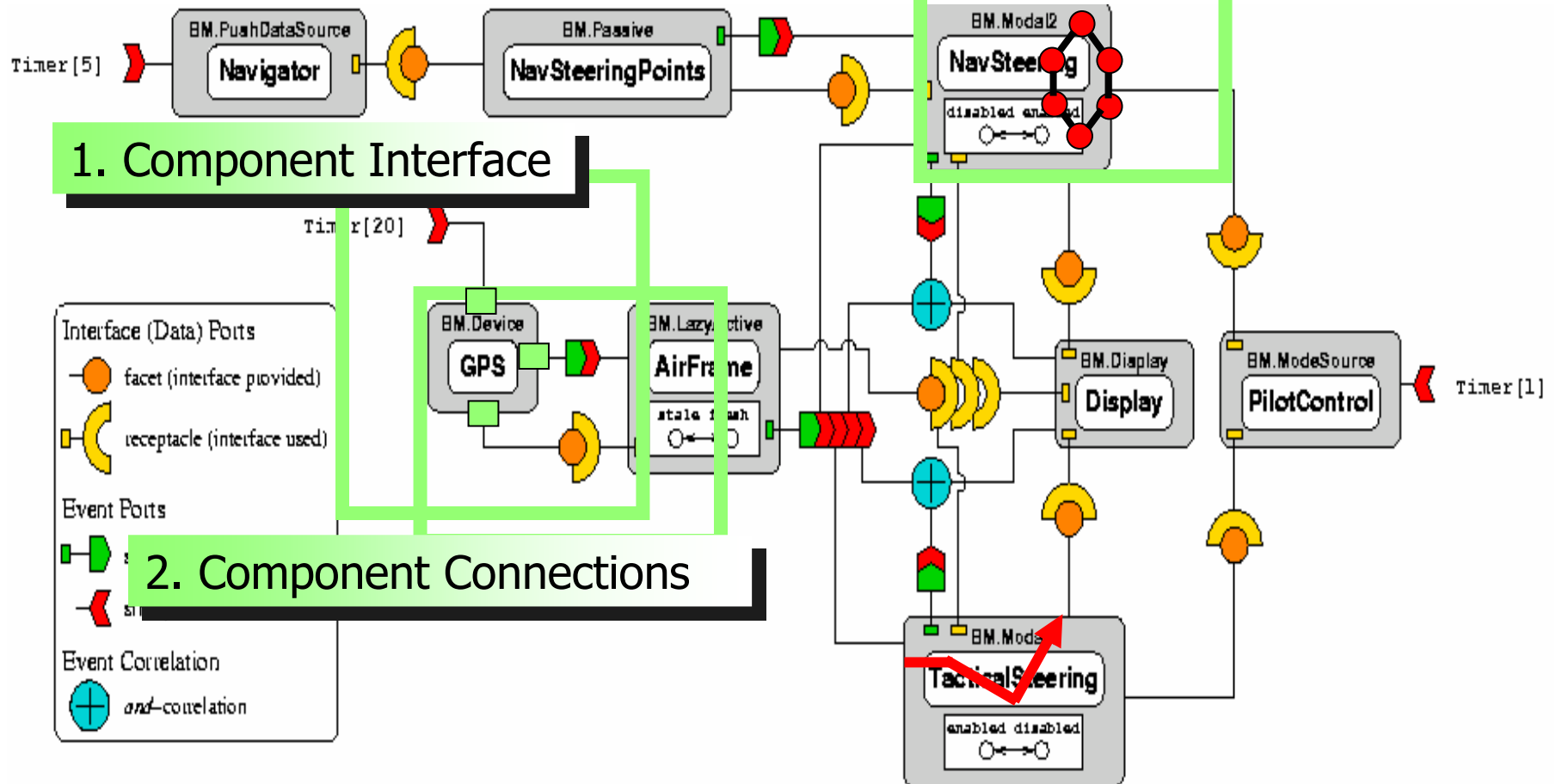
Outline

4. Transition Semantics

1. Component Interface

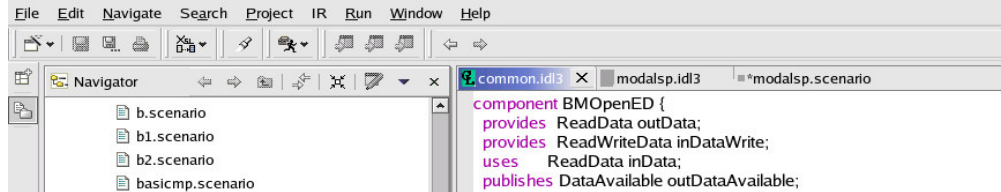


2. Component Connections

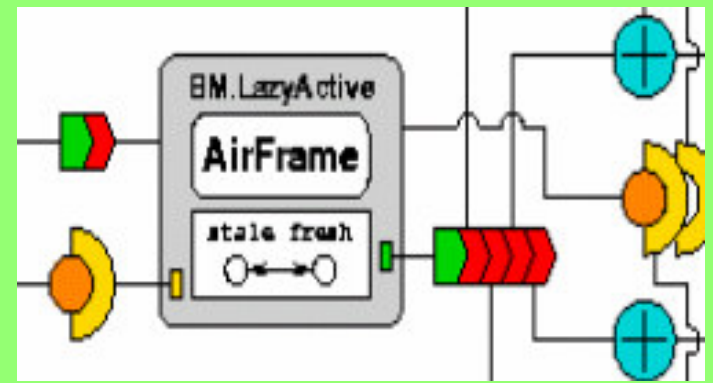


3. Dependence Information

Component IDL

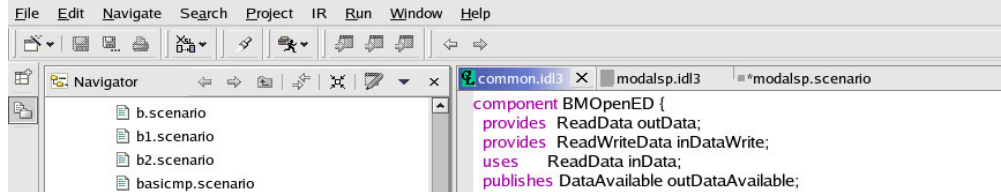


```
component BMLazyActive {  
  provides ReadData outData;  
  uses ReadData inData;  
  publishes DataAvailable outDataAvailable;  
  consumes DataAvailable inDataAvailable;  
  attribute LazyActiveMode dataStatus;  
};
```



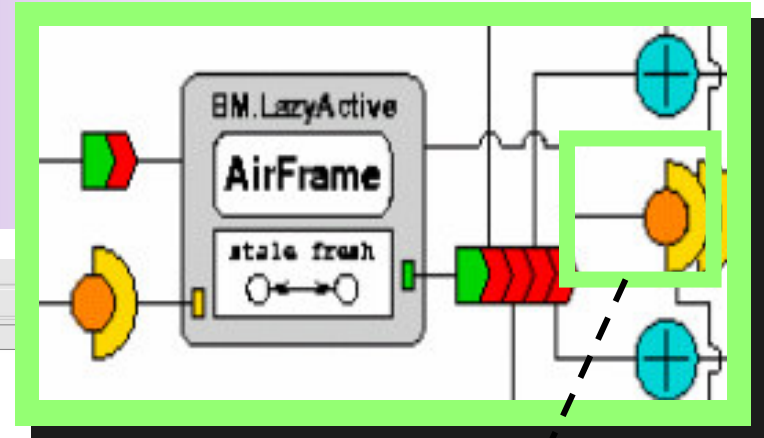
CORBA 3
CCM IDL
ModalSP Components

Component IDL



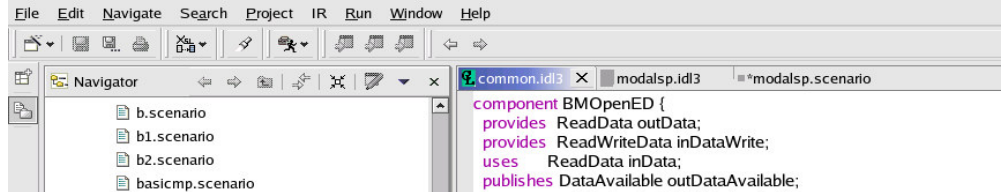
```
component BMLazyActive {  
  provides ReadData outData;  
  uses ReadData inData;  
  publishes DataAvailable outDataAvailable;  
  consumes DataAvailable inDataAvailable;  
  attribute LazyActiveMode dataStatus;  
};
```

*output data port
(facet)*

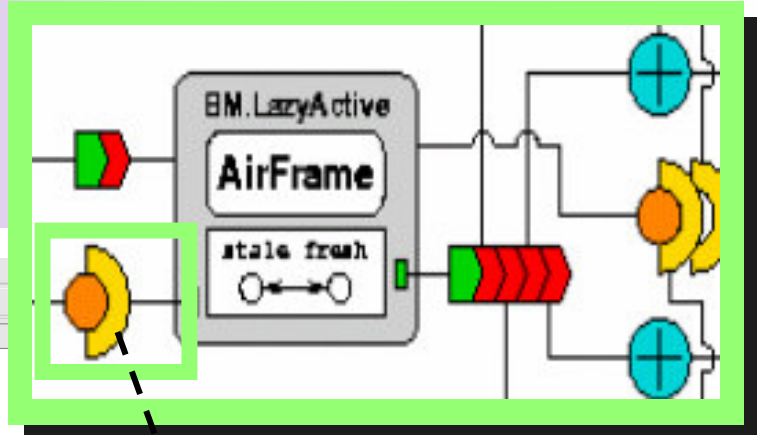


CORBA 3
CCM IDL
ModalSP Components

Component IDL



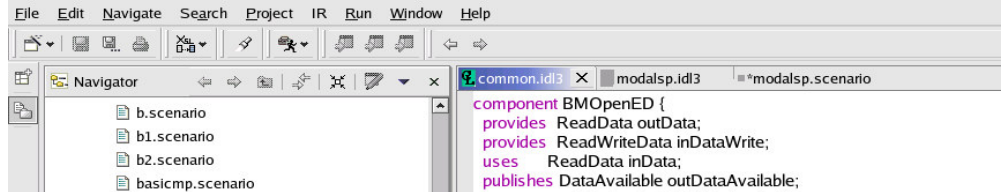
```
component BMLazyActive {  
  provides ReadData outData;  
  uses ReadData inData;  
  publishes DataAvailable outDataAvailable;  
  consumes DataAvailable inDataAvailable;  
  attribute LazyActiveMode dataStatus;  
};
```



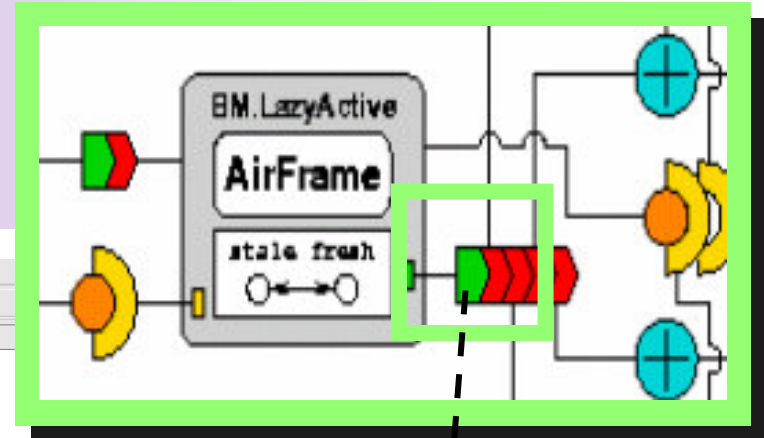
*input data port
(receptacle)*

CORBA 3
CCM IDL
ModalSP Components

Component IDL



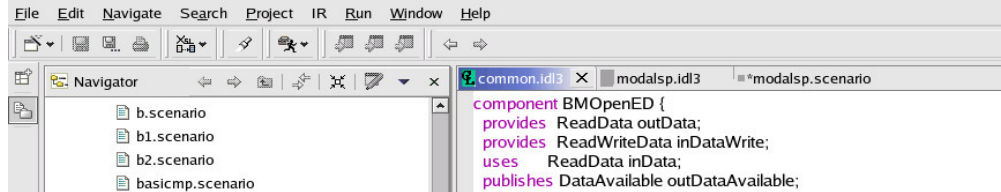
```
component BMLazyActive {  
  provides ReadData outData;  
  uses ReadData inData;  
  publishes DataAvailable outDataAvailable;  
  consumes DataAvailable inDataAvailable;  
  attribute LazyActiveMode dataStatus;  
};
```



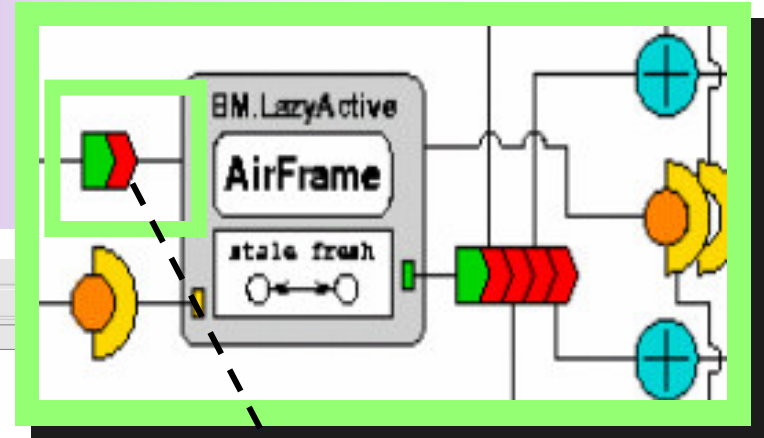
*output event port
(event source)*

CORBA 3
CCM IDL
ModalSP Components

Component IDL



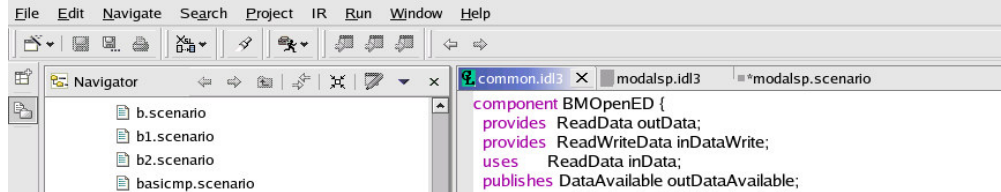
```
component BMLazyActive {  
  provides ReadData outData;  
  uses ReadData inData;  
  publishes DataAvailable outDataAvailable;  
  consumes DataAvailable inDataAvailable;  
  attribute LazyActiveMode dataStatus;  
};
```



*input event port
(event sink)*

CORBA 3
CCM IDL
ModalSP Components

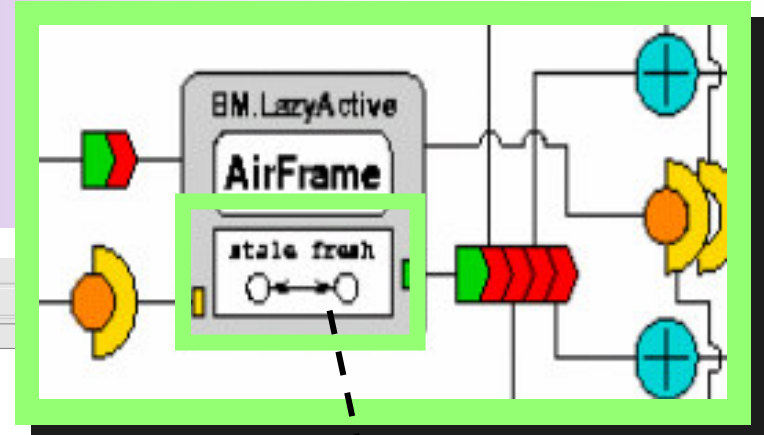
Component IDL



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

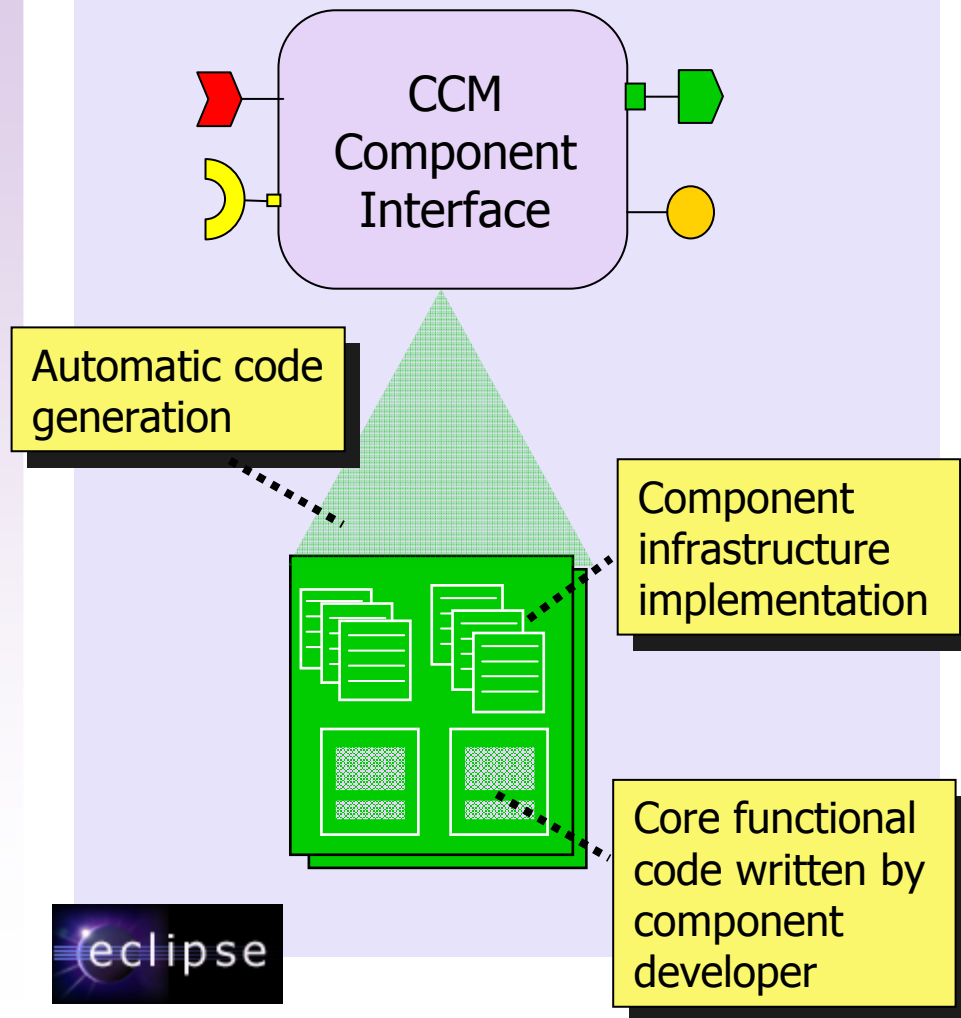
mode attribute

CORBA 3
CCM IDL
ModalSP Components



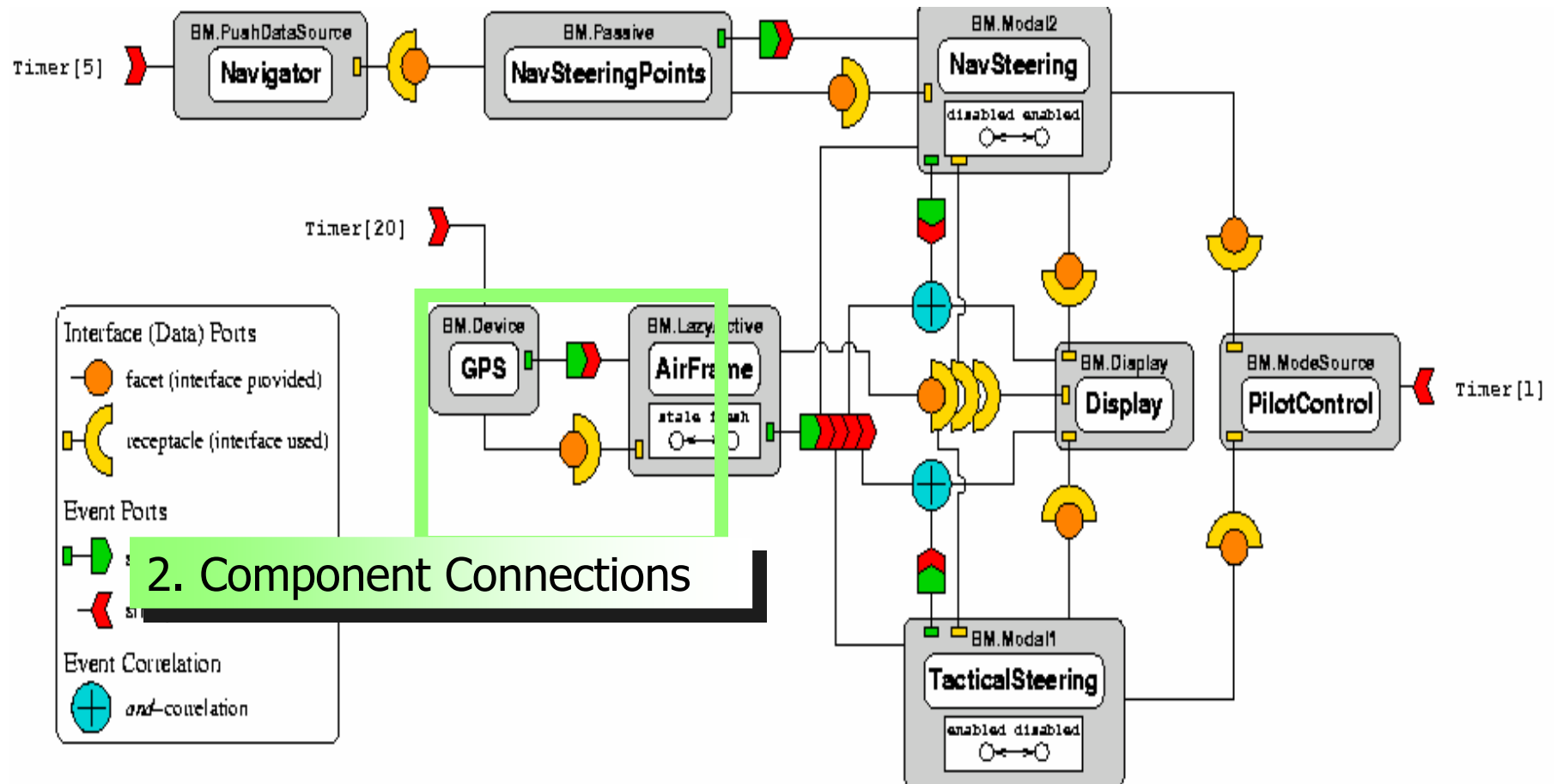
Component Development

CADENA GO



- 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



Three Synchronized Views

Scenario Description

Graphical View

Spreadsheet View

Textual View

```
system ModelSPScenario {
  import modslp;

  locations Board1, Board2, Board3, Board4, Board5, Board6, Board7, Board8;
  // convention is that component instances that are
  // clients of data/event declare for each client port the corresponding
  // server port.
  // connect ... client port of current instance) to (server port)...

  instance EventChannel implements EventChannel on Board1 {
    // connect ... client port of current instance) to (server port)...
  }

  instance GPS implements BMDevice on Board1 {
    connect this.timeout to EventChannel.timeout atRate 20;
  }

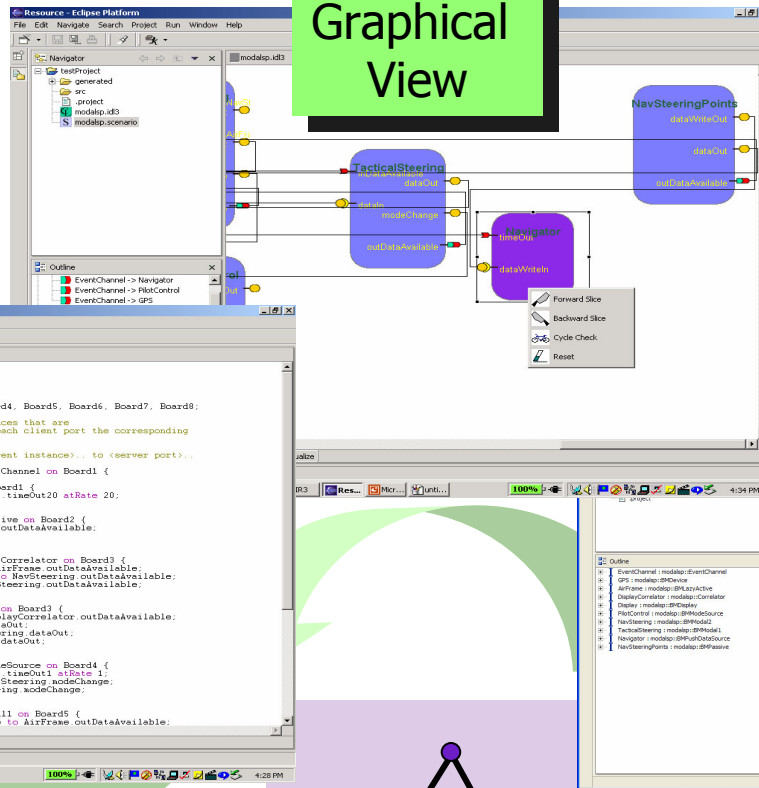
  instance AirFrame implements BMLazyActive on Board2 {
    connect this.inDataAvailable to GPS.outDataAvailable;
    connect this.dataIn to GPS.dataOut;
  }

  instance DisplayCorrelator implements Correlator on Board3 {
    connect this.inDataAvailable to AirFrame.outDataAvailable;
    connect this.inDataAvailableNavSteering to NavSteering.outDataAvailable;
    connect this.inDataAvailable to TacticalSteering.outDataAvailable;
  }

  instance Display implements BMDisplay on Board3 {
    connect this.inDataAvailable to DisplayCorrelator.outDataAvailable;
    connect this.dataIn to AirFrame.dataOut;
    connect this.dataIn to TacticalSteering.dataOut;
  }

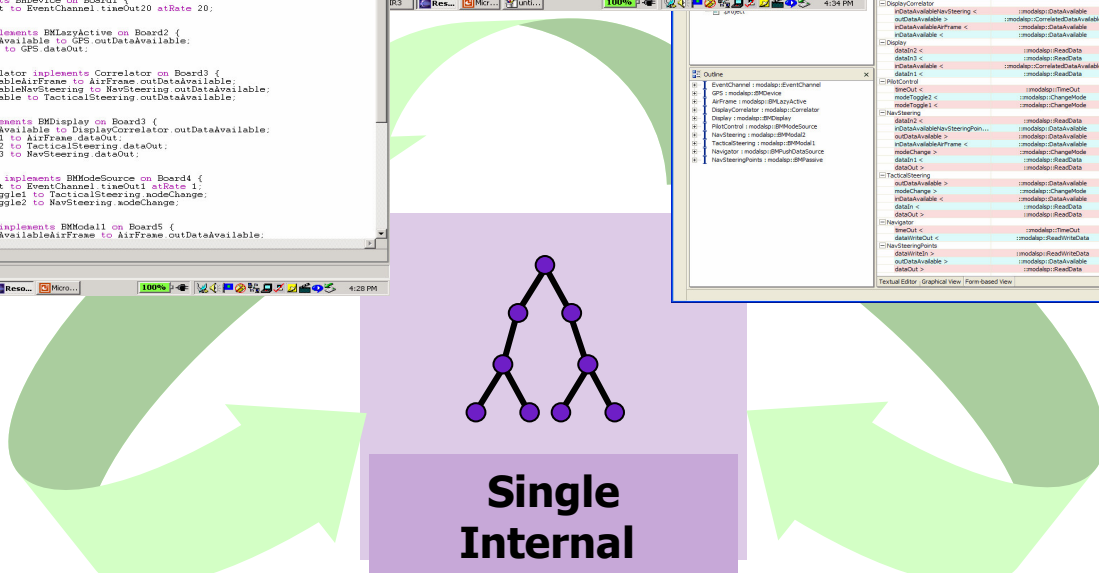
  instance PilotControl implements BMModeSource on Board4 {
    connect this.inDataAvailable to AirFrame.outDataAvailable;
  }

  instance NavSteering implements BMModell on Board5 {
    connect this.inDataAvailable to AirFrame.outDataAvailable;
  }
}
```



ComponentPart	Type	Board	Port	Direction	Target
EventChannel	impl:port:Timeout	Board1	1	> timeOut	Navigator
GPS	impl:port:Timeout	Board1	20	> timeOut	GPS
AirFrame	impl:port:DataAvailable	Board2	20	< dataIn	EventChannel
DisplayCorrelator	impl:port:DataAvailable	Board3	20	< dataIn	AirFrame
Display	impl:port:DataAvailable	Board3	20	< dataIn	TacticalSteering
PilotControl	impl:port:DataAvailable	Board4	1	< dataIn	NavSteering
NavSteering	impl:port:DataAvailable	Board5	20	< dataIn	DisplayCorrelator
NavSteeringPoints	impl:port:DataAvailable	Board8	5	< dataIn	NavSteering

Single Internal Representation



Textual View

The screenshot shows the Eclipse IDE interface. The main editor displays the contents of a file named 'modalsp.scenario'. The code is written in a textual style, using keywords like 'system', 'import', and 'instance' to define a scenario. A green box highlights a specific instance definition for 'AirFrame' on 'Board2'. A purple lightning bolt icon points to the 'instance' keyword in the code below the highlighted box. The Navigator and Outline views are visible on the left side of the IDE.

```
system ModalSPScenario {  
  import modalsp;  
  
  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 inDataAvailableAirFrame to AirFrame.outDataAvailable;  
  connect inDataAvailableNavSteering to NavSteering.outDataAvailable;  
  connect inDataAvailable to TacticalSteering.outDataAvailable;  
}  
  
Instance Display implements BMDisplay on Board3 {  
  connect this.inDataAvailable to DisplayCorrelator.outDataAvailable;  
  connect this.dataIn1 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.modeToggle1 to TacticalSteering.modeChange;  
  connect this.modeToggle2 to NavSteering.modeChange;  
}  
  
Instance NavSteering implements BModall on Board5 {  
  connect this.inDataAvailableAirFrame to AirFrame.outDataAvailable;  
}
```

Textual View

The screenshot shows the Eclipse IDE interface. The main editor displays the textual view of a scenario file named `modalsp.scenario`. The code defines a system `ModalSPScenario` and includes several component instances with their connections. A yellow callout box points to the `Instance AirFrame` line, with the text "...allocate AirFrame component instance". A green dashed box highlights the connection lines for the `Instance AirFrame` block. A purple lightning bolt icon points to the `Instance AirFrame` line. The Outline view on the left shows the project structure, including the `modalsp.idl3` project and the `modalsp.scenario` file. The taskbar at the bottom shows the Windows taskbar with various open applications and the system clock at 4:28 PM.

```
system ModalSPScenario {
import modalsp;

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 inDataAvailableAirFrame to AirFrame.outDataAvailable;
connect inDataAvailableNavSteering to NavSteering.outDataAvailable;
connect inDataAvailable to TacticalSteering.outDataAvailable;
}

Instance Display implements BMDisplay on Board3 {
connect this.inDataAvailable to DisplayCorrelator.outDataAvailable;
connect this.dataIn1 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.modeToggle1 to TacticalSteering.modeChange;
connect this.modeToggle2 to NavSteering.modeChange;
}

Instance NavSteering implements BModall on Board5 {
connect this.inDataAvailableAirFrame to AirFrame.outDataAvailable;
}
```

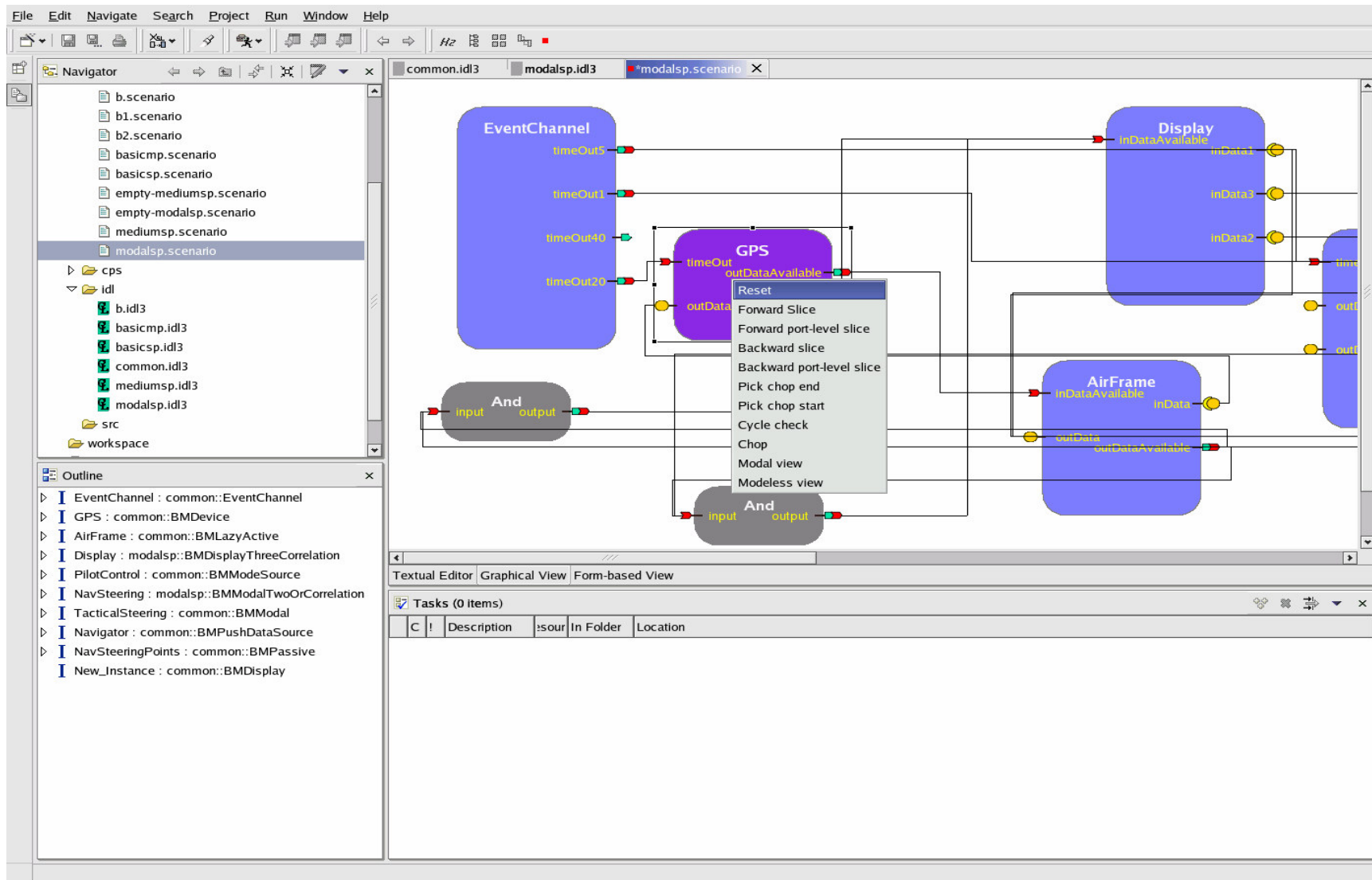
Textual View

The screenshot shows the Eclipse IDE interface with the following components:

- Navigator:** Shows a project structure with folders like 'generated', 'src', and files like 'modalsp.idl3' and 'modalsp.scenario'.
- Outline:** Shows a hierarchical view of the scenario's components, including 'EventChannel', 'GPS', 'AirFrame', 'DisplayCorrelator', 'Display', 'PilotControl', and 'NavSteering'.
- Textual Editor:** Displays the scenario code in a textual view. A specific instance is highlighted with a green box and a red dashed border:

```
Instance AirFrame implements BMLazyActive on Board2 {  
  connect this.inDataAvailable to GPS.outDataAvailable;  
  connect this.dataIn to GPS.dataOut;  
}
```
- Annotation:** A yellow box with a black border contains the text: *...connect event ports and facet/receptacles*. A red dashed arrow points from this box to the highlighted code block.
- Taskbar:** Shows the Windows taskbar with various open applications and the system clock at 4:28 PM.

Graphical View



Graphical View

The screenshot displays a software development environment with a graphical view of a system. The interface includes a menu bar (File, Edit, Navigate, Search, Project, Run, Window, Help), a Navigator pane on the left showing a project tree, and an Outline pane at the bottom left listing components like EventChannel, GPS, AirFrame, Display, PilotControl, NavSteering, TacticalSteering, Navigator, NavSteeringPoints, and New_Instance. The main workspace shows a graphical diagram with components such as EventChannel, And, Display, and AirFrame. A modal view menu is overlaid on the diagram, listing various analysis options. A yellow callout box points to the diagram with the text "...design-level analyses mode-base views".

Modal View Menu:

- Reset
- Forward Slice
- Forward port-level slice
- Backward slice
- Backward port-level slice
- Pick chop end
- Pick chop start
- Cycle check
- Chop
- Modal view
- Modeless view

Diagram Components:

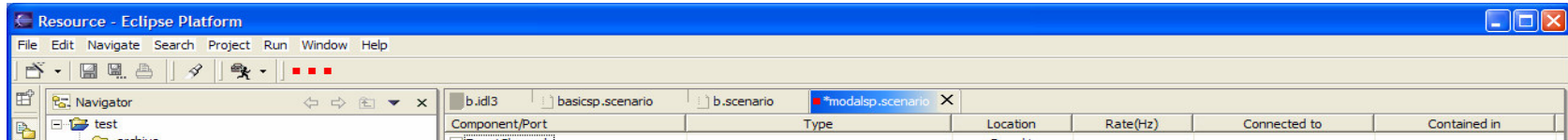
- EventChannel
- And (input/output)
- Display (inDataAvailable, inData1, inData2, out)
- AirFrame (inDataAvailable, inData, outData, outDataAvailable)

Callout Text: ...design-level analyses mode-base views

Tasks Table:

C	Description	Source	In Folder	Location

Spreadsheet View



Component/Port	Type	Location	Rate(Hz)	Connected to	Contained in
timeOut <	::modalsp::TimeOut	Board1	20	< timeOut20	EventChannel
outDataAvailable >	::modalsp::DataAvailable		20	> inDataAvailable	AirFrame
dataOut >	::modalsp::ReadData		0	> dataIn	AirFrame

...ports for component type

...port types

...port connections

- EventChannel : modalsp::EventChannel
- GPS : modalsp::BMDevice
- AirFrame : modalsp::BMLazyActive
- DisplayCorrelator : modalsp::Correlator
- Display : modalsp::BMDisplay
- PilotControl : modalsp::BMMModeSource
- NavSteering : modalsp::BMModal2
- TacticalSteering : modalsp::BMModal1
- Navigator : modalsp::BMPushDataSource
- NavSteeringPoints : modalsp::BMPassive

Component/Port	Type	Location	Rate(Hz)	Connected to	Contained in
dataOut >	::modalsp::ReadData		0	> dataIn	TacticalSteering
timeOut <	::modalsp::TimeOut	Board3	20	< timeOut1	EventChannel
modeToggle2 <	::modalsp::ChangeMode		1	< modeChange	NavSteering
modeToggle1 <	::modalsp::ChangeMode		1	< modeChange	TacticalSteering
inDataAvailableAirFrame <	::modalsp::DataAvailable		20	< outDataAvailable	AirFrame
inDataAvailable <	::modalsp::DataAvailable		20	< outDataAvailable	TacticalSteering
dataIn2 <	::modalsp::ReadData	Board3	20	< dataOut	TacticalSteering
dataIn3 <	::modalsp::ReadData		20	< dataOut	NavSteering
inDataAvailable <	::modalsp::CorrelatedDataAvailable		20	< outDataAvailable	DisplayCorrelator
dataIn1 <	::modalsp::ReadData		20	< dataOut	AirFrame
timeOut <	::modalsp::TimeOut	Board4	1	< timeOut1	EventChannel
modeToggle2 <	::modalsp::ChangeMode		1	< modeChange	NavSteering
modeToggle1 <	::modalsp::ChangeMode		1	< modeChange	TacticalSteering
dataIn2 <	::modalsp::ReadData	Board5	20	< dataOut	AirFrame
inDataAvailableNavSteeringPoi...	::modalsp::DataAvailable		0	< outDataAvailable	NavSteeringPoints
outDataAvailable >	::modalsp::DataAvailable		20	> inDataAvailableNavStee...	DisplayCorrelator
inDataAvailableAirFrame <	::modalsp::DataAvailable		20	< outDataAvailable	AirFrame
modeChange >	::modalsp::ChangeMode		0	> modeToggle2	PilotControl
dataIn1 <	::modalsp::ReadData		20	< dataOut	NavSteeringPoints
dataOut >	::modalsp::ReadData		0	> dataIn3	Display
outDataAvailable >	::modalsp::DataAvailable	Board6	20	> inDataAvailable	DisplayCorrelator
modeChange >	::modalsp::ChangeMode		0	> modeToggle1	PilotControl
inDataAvailable <	::modalsp::DataAvailable		20	< outDataAvailable	AirFrame
dataIn <	::modalsp::ReadData		20	< dataOut	AirFrame
dataOut >	::modalsp::ReadData		0	> dataIn2	Display
timeOut <	::modalsp::TimeOut	Board7	5	< timeOut5	EventChannel
dataWriteOut <	::modalsp::ReadWriteData		5	< dataWriteIn	NavSteeringPoints
dataWriteIn >	::modalsp::ReadWriteData	Board8	0	> dataWriteOut	Navigator
outDataAvailable >	::modalsp::DataAvailable		0	> inDataAvailableNavStee...	NavSteering
dataOut >	::modalsp::ReadData		0	> dataIn1	NavSteering

Spreadsheet View

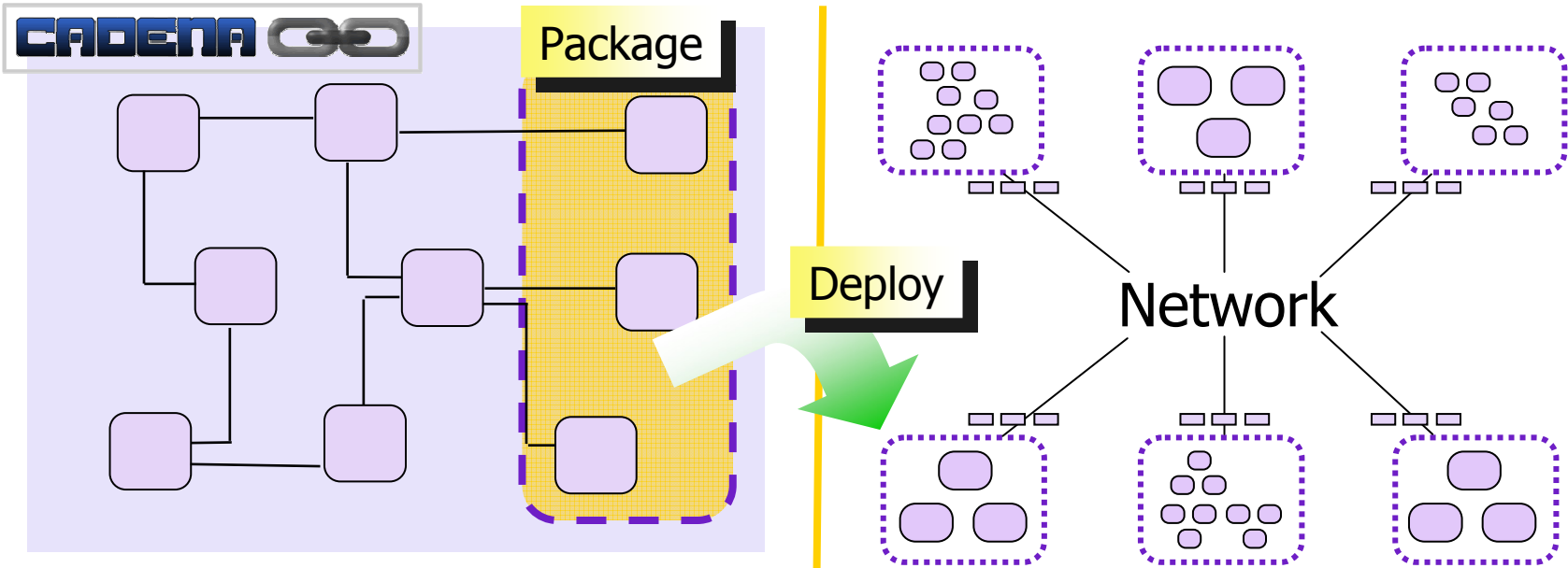
The screenshot shows the Eclipse IDE interface with the Spreadsheet View open. The main window displays a table with columns: Component/Port, Type, Location, Rate(Hz), and Connected to. The table lists various components like EventChannel, GPS, AirFrame, Display, PilotControl, NavSteering, TacticalSteering, Navigator, NavSteeringPoints, New_Instance, CorrelatedEventInstance_\$1, and CorrelatedEventInstance_\$2. A yellow callout box with a red dashed arrow points to the 'Connected to' column, highlighting a pull-down menu with options like GPS.outData, TacticalSteering.outData, AirFrame.outData, NavSteeringPoints.outData, PilotControl.outDataWrite, NavSteeringPoints.inDataWrite, NavSteering.outDataWrite, PilotControl.outData, TacticalSteering.modeChange, NavSteering.outData, and NavSteering.modeChange.

Component/Port	Type	Location	Rate(Hz)	Connected to
EventChannel	common::EventChannel	Board1		
GPS	common::BMDevice	Board1		
timeOut <=	::common::TimeOut		20	<= EventChannel.timeOut20
outDataAvailable =>	::common::DataAvailable		0	=> AirFrame.inDataAvailable
outData <=	::common::ReadData		0	<= AirFrame.inData
AirFrame	common::BMLazyActive	Board2		
Display	modalsp::BMDisplayThreeCorrelator	Board3		
PilotControl	common::BMModeSource	Board4		
timeOut <=	::common::TimeOut		1	<= EventChannel.timeOut1
modeToggle2 =>	::common::ChangeMode		0	=> NavSteering.modeChange
modeToggle1 =>	::common::ChangeMode		0	=> TacticalSteering.modeChange
outDataWrite <=	::common::ReadData		0	<= New_Instance.inData
NavSteering	modalsp::BMModalTwoOrCorrelation	Board5		
TacticalSteering	common::BMModal	Board6		
Navigator	common::BMPushDataSource	Board7		
NavSteeringPoints	common::BMPassive	Board8		
New_Instance	common::BMDisplay	unDefinedLoc		
inData =>	::common::ReadData		0	GPS.outData
CorrelatedEventInstance_\$1	And			TacticalSteering.outData
CorrelatedEventInstance_\$2	And			AirFrame.outData

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



automatic
generation

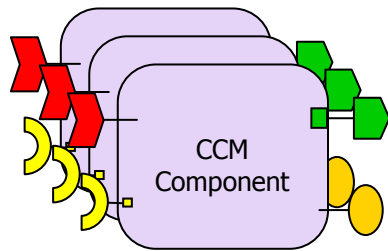
```
<CONFIGURATION_PASS>  
<HOME> <...>  
<COMPONENT>  
<ID> <...></ID>  
<EVENT_SUPPLIER>  
<...events this component supplies...>  
</EVENT_SUPPLIER>  
</COMPONENT>  
</HOME>  
</CONFIGURATION_PASS>
```

CCM XML-based
Configuration and
Deployment information

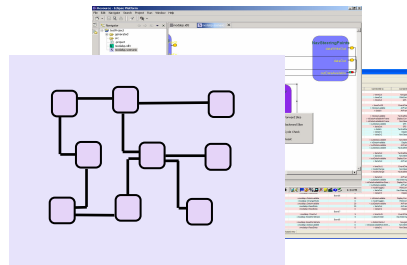
CCM Deployment
Infrastructure
(e.g., CIAO, OpenCCM)

Assessment

Cadena supports...



Component Interfaces and implementations



Component Assemblies

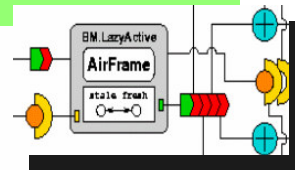
Integration with underlying component middleware frameworks (e.g., CIAO, OpenCCM) through well-defined interfaces (plug in your favorite CCM framework)

...but the *real* power of model-driven development comes from...

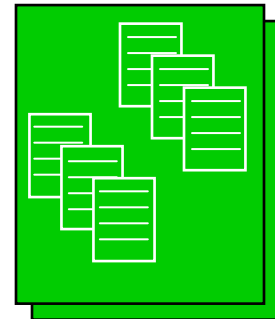
- decorating component interfaces and assemblies with a variety of forms of behavioral contracts and meta-data that can be leveraged by model-level analysis and optimizations
- these analyses/optimization may be general purpose, or tailored to specific product-lines or products

Leverage CORBA IDL

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



IDL Compiler



Component
Implementation
Stubs & Skeletons

+

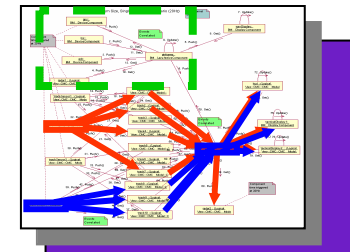
```
dependencydefault  
  == none;  
dependencies {  
  inDataAvailable  
  ->  
  outDataAvailable;  
}
```

Dependency
Annotations

```
behavior {  
  if (mode==enabled) {  
    push outDataAvailable;  
  }  
  else  
  ...  
}
```

Transition System
Semantics

Model Builder



Dependency Analysis
and
Model-checking Engine

Incremental Specification

Specifications

Component Structure

Increasing Effort & Strength of Verification

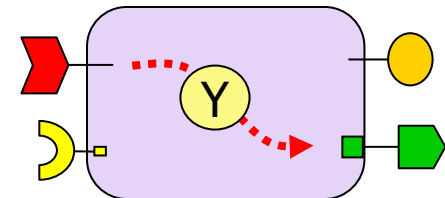
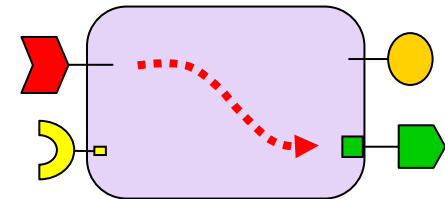
port action dependencies

refinement

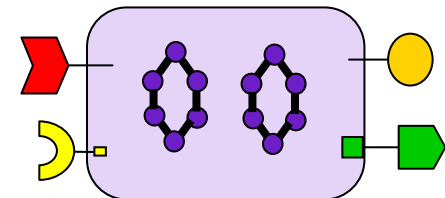
state-based dependencies

refinement

component transition semantics



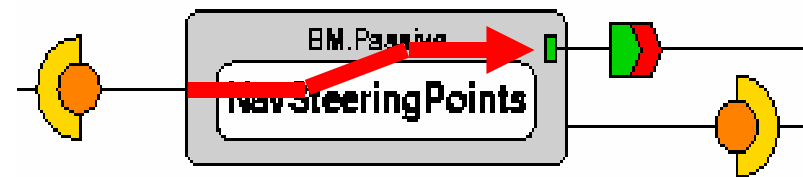
...only in mode Y



...state machines give abstract behavior

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

Light-weight Dependency Specs



```
dependencydefault == none;
```

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

```
behavior { ... }
```

call on set_data()

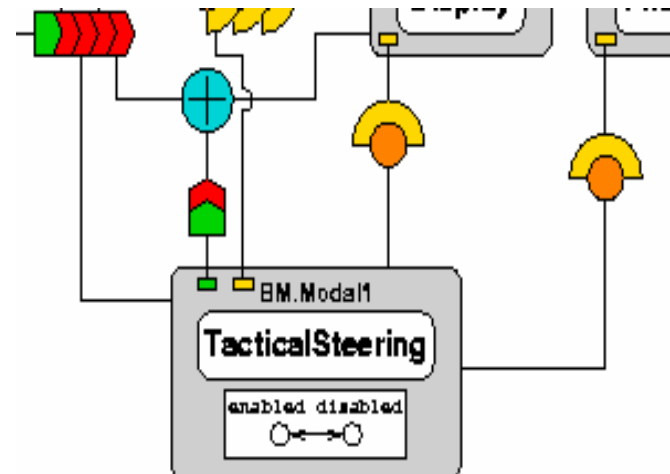
triggers

*outDataAvailable
port action*

Light-weight Dependency Specs

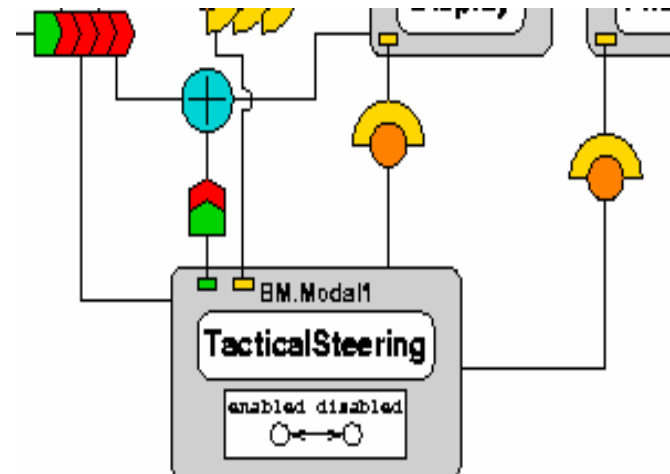
triggers no other actions

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



Light-weight Dependency Specs

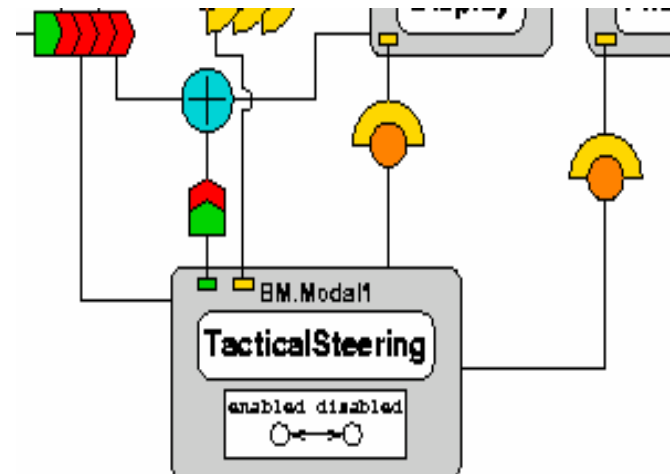
```
dependencydefault == all;  
  
dependencies {  
  modeChange() ->;  
  case modeChange.modeVar of {  
    enabled: inDataAvailable  
             -> dataIn.get_data(),  
              outDataAvailable;  
    disabled: inDataAvailable ->;  
  }  
}  
  
behavior { ... }
```



*in enabled mode,
shows actions
triggered by
receipt of event
on
inDataAvailable
port*

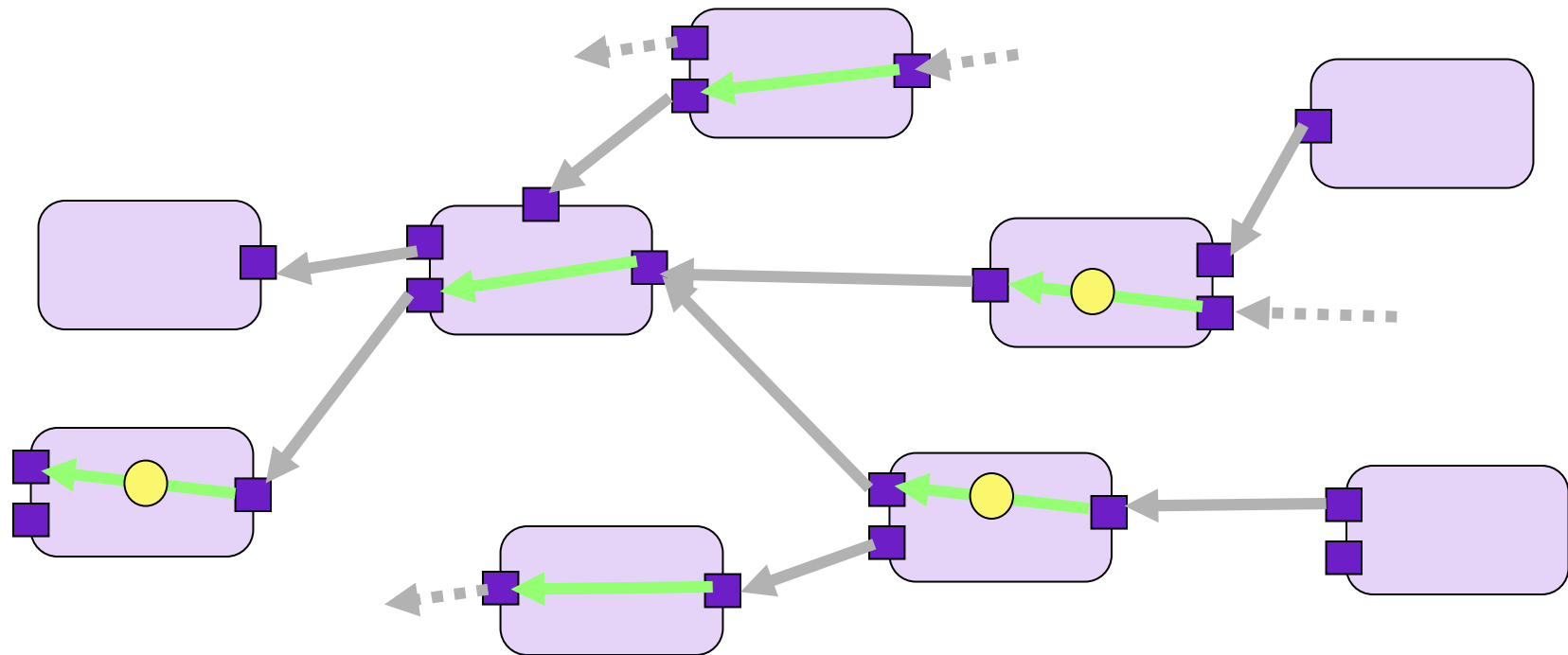
Light-weight Dependency Specs

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



in disabled mode, inDataAvailable triggers no other port actions

Design-Level Flow/Dependence Graphs

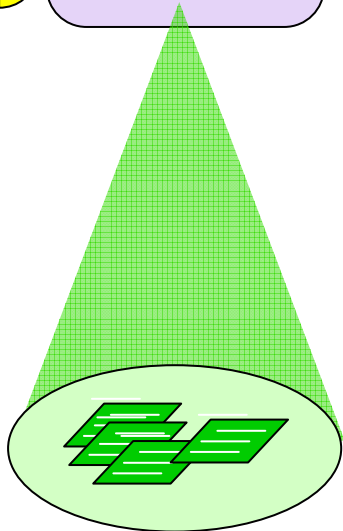
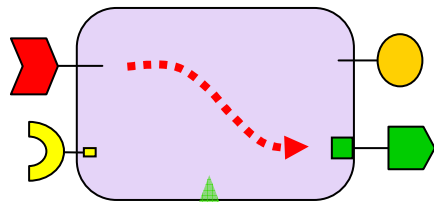


- From system configuration information
- From user-specified intra-component dependences
- State predicates giving conditional dependences

Theme

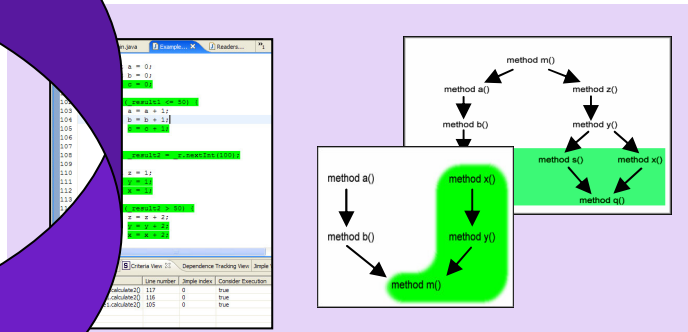
Connecting Models & Code

Model-level Specification/Abstraction



Java Implementation

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

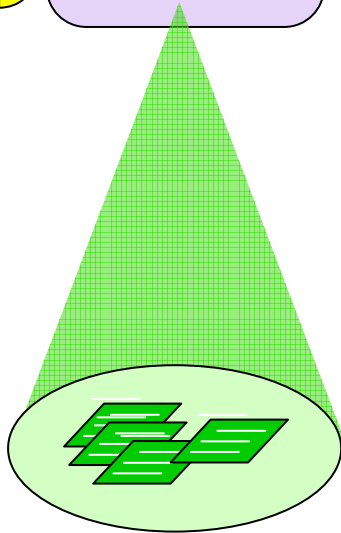
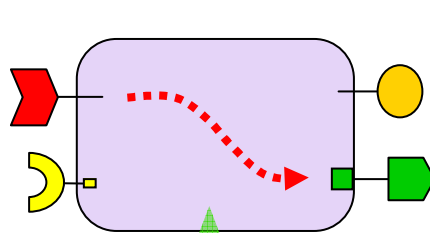


SAnToS Indus
Dependence Calculation
(Java Program Slicing)

Theme

Connecting Models & Code

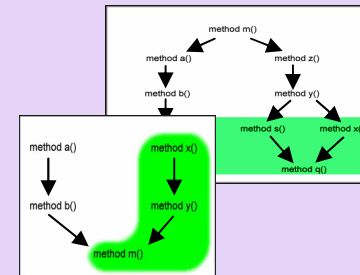
Model-level Specification/Abstraction



Java Implementation

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

```
97  
98 int a = 0;  
99 int b = 1;  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120
```

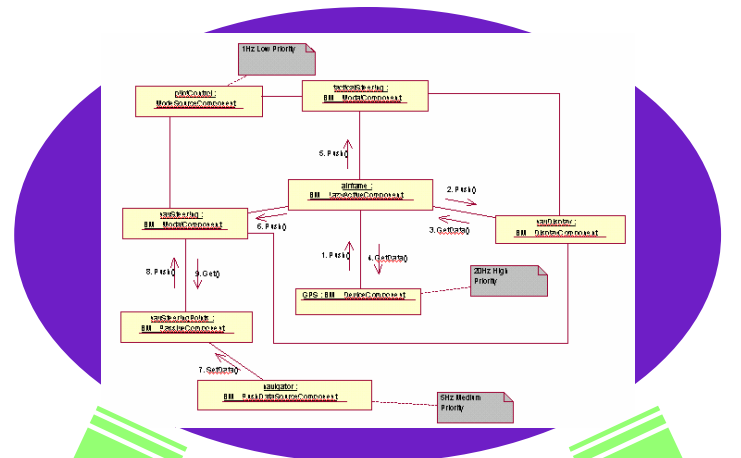


SAnToS Indus
Dependence Calculation
(Java Program Slicing)

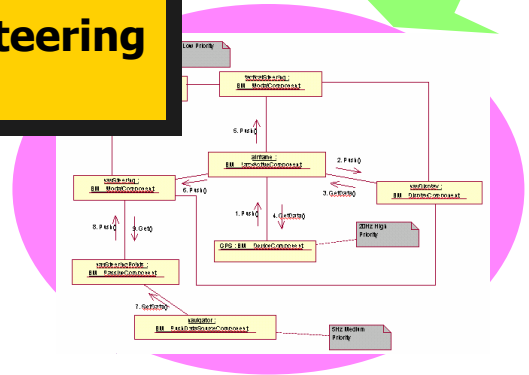
Simplifying Assembly Visualization

Goal: Simplify visualization of system by hiding component/connections in various modes.

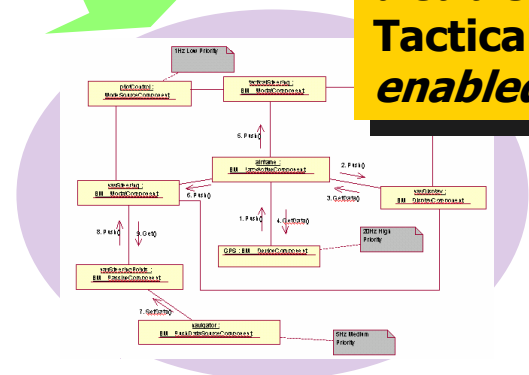
Scenario Diagram w/
Complete Connectivity



**NavSteering
enabled
TacticalSteering
disabled**



**NavSteering
disabled
TacticalSteering
enabled**



Projections of Systems with Enabled
Connectivity for Different Modes

Automatic!

Mode-based Projections

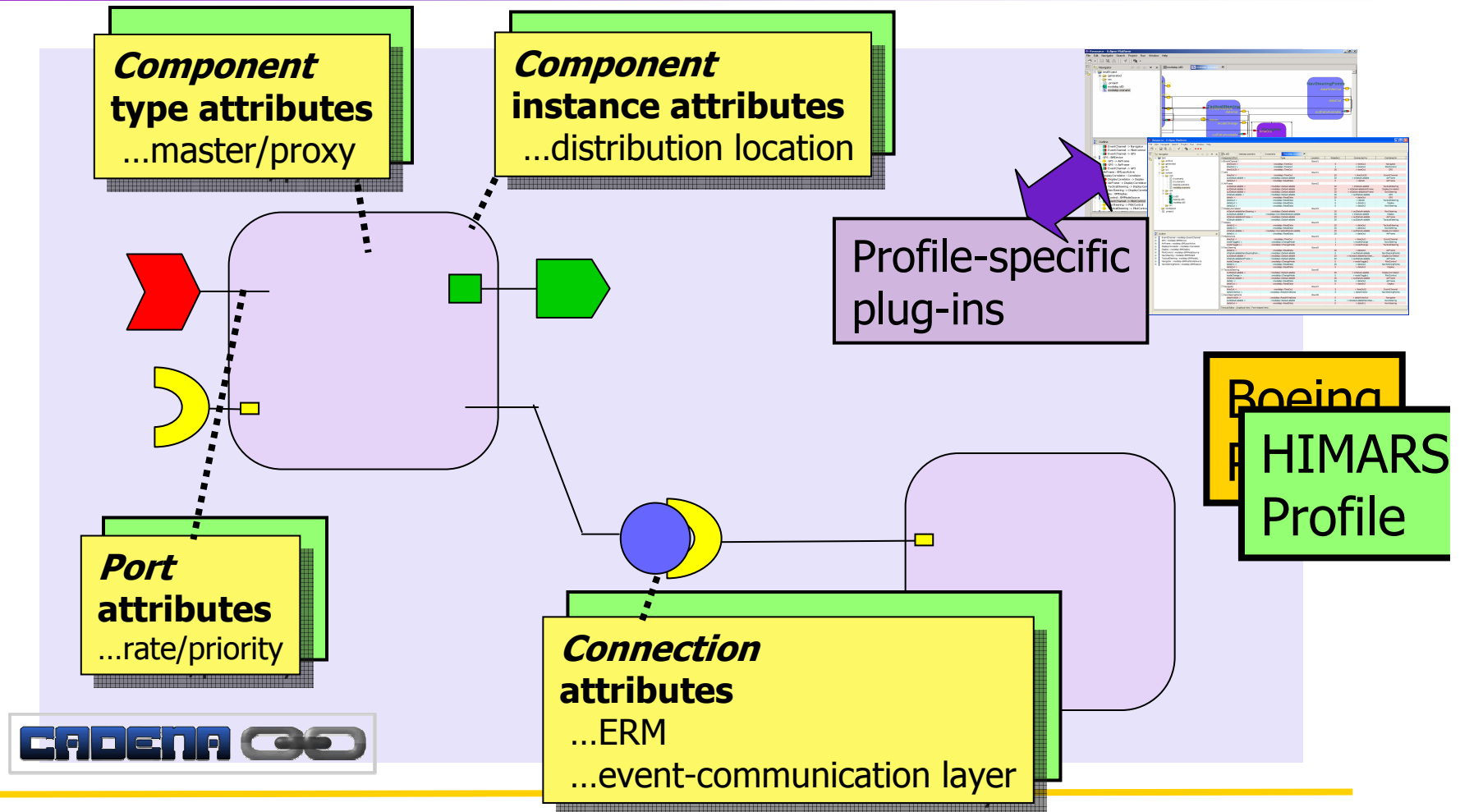
The screenshot displays a software development environment with a mode-based projection diagram. The diagram shows an EventChannel component on the left and a Display component on the right, connected by lines. A dialog box is overlaid on the diagram, showing a table of mode variables and their values. The dialog box has a green border and contains the following table:

Mode Variable	Values
AirFrame#LazyActiveMode	stale
TacticalSteering#OnOffMode	enabled
NavSteering#OnOffMode	enabled
	enabled
	disabled

A yellow callout box with a black border points to the 'enabled' value in the fourth row of the table, containing the text: *...possible values for mode variables*. The dialog box also features 'OK' and 'Cancel' buttons at the bottom.

Value Added: Multiple *mode-based views* are automatically created and synchronized through the design process.

Configurable Product Line Profiles



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

Component/Port	Type	Location	Rate(Hz)	Connected to	Contained in
GPS		Board1			
timeOut <	::modalsp::TimeOut	20	<	timeOut20	EventChannel
outDataAvailable >	::modalsp::DataAvailable	20	>	inDataAvailable	AirFrame
dataOut >	::modalsp::ReadData	0	>	dataIn	AirFrame

...ports for component type

...port types

RT Attributes

...distribution sites

...rate group

...port connections

Component Model Hierarchy:

- EventChannel : modalsp::EventChannel
- GPS : modalsp::BMDevice
- AirFrame : modalsp::BMLazyActive
- DisplayCorrelator : modalsp::Correlator
- Display : modalsp::BMDisplay
- PilotControl : modalsp::BMMModeSource
- NavSteering : modalsp::BMModal2
- TacticalSteering : modalsp::BMModal1
- Navigator : modalsp::BMPushDataSource
- NavSteeringPoints : modalsp::BMPassive

Table Content (continued):

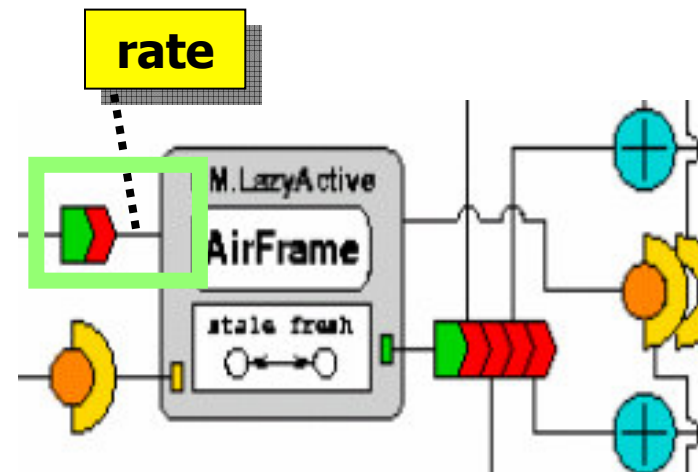
inDataAvailableAirFrame <			20	<	outDataAvailable	AirFrame
inDataAvailable <			20	<	outDataAvailable	TacticalSteering
dataIn2 <			20	<	dataOut	TacticalSteering
dataIn3 <			20	<	dataOut	NavSteering
inDataAvailable <			20	<	outDataAvailable	DisplayCorrelator
dataIn1 <			20	<	dataOut	AirFrame
timeOut <	::modalsp::TimeOut	Board4	1	<	timeOut1	EventChannel
modeToggle2 <	::modalsp::ChangeMode		1	<	modeChange	NavSteering
modeToggle1 <	::modalsp::ChangeMode	Board5	1	<	modeChange	TacticalSteering
dataIn2 <	::modalsp::ReadData		20	<	dataOut	AirFrame
inDataAvailableNavSteeringPoi...	::modalsp::DataAvailable		0	<	outDataAvailable	NavSteeringPoints
outDataAvailable >	::modalsp::DataAvailable		20	>	inDataAvailableNavStee...	DisplayCorrelator
inDataAvailableAirFrame <	::modalsp::DataAvailable		20	<	outDataAvailable	AirFrame
modeChange >	::modalsp::ChangeMode		0	>	modeToggle2	PilotControl
dataIn1 <	::modalsp::ReadData		20	<	dataOut	NavSteeringPoints
dataOut >	::modalsp::ReadData	Board6	0	>	dataIn3	Display
outDataAvailable >	::modalsp::DataAvailable		20	>	inDataAvailable	DisplayCorrelator
modeChange >	::modalsp::ChangeMode		0	>	modeToggle1	PilotControl
inDataAvailable <	::modalsp::DataAvailable		20	<	outDataAvailable	AirFrame
dataIn <	::modalsp::ReadData		20	<	dataOut	AirFrame
dataOut >	::modalsp::ReadData		0	>	dataIn2	Display
timeOut <	::modalsp::TimeOut	Board7	5	<	timeOut5	EventChannel
dataWriteOut <	::modalsp::ReadWriteData		5	<	dataWriteIn	NavSteeringPoints
dataWriteIn >	::modalsp::ReadWriteData	Board8	0	>	dataWriteOut	Navigator
outDataAvailable >	::modalsp::DataAvailable		0	>	inDataAvailableNavStee...	NavSteering
dataOut >	::modalsp::ReadData		0	>	dataIn1	NavSteering

Automated Design Advice

Dependency-driven priority/rate assignment to event handlers

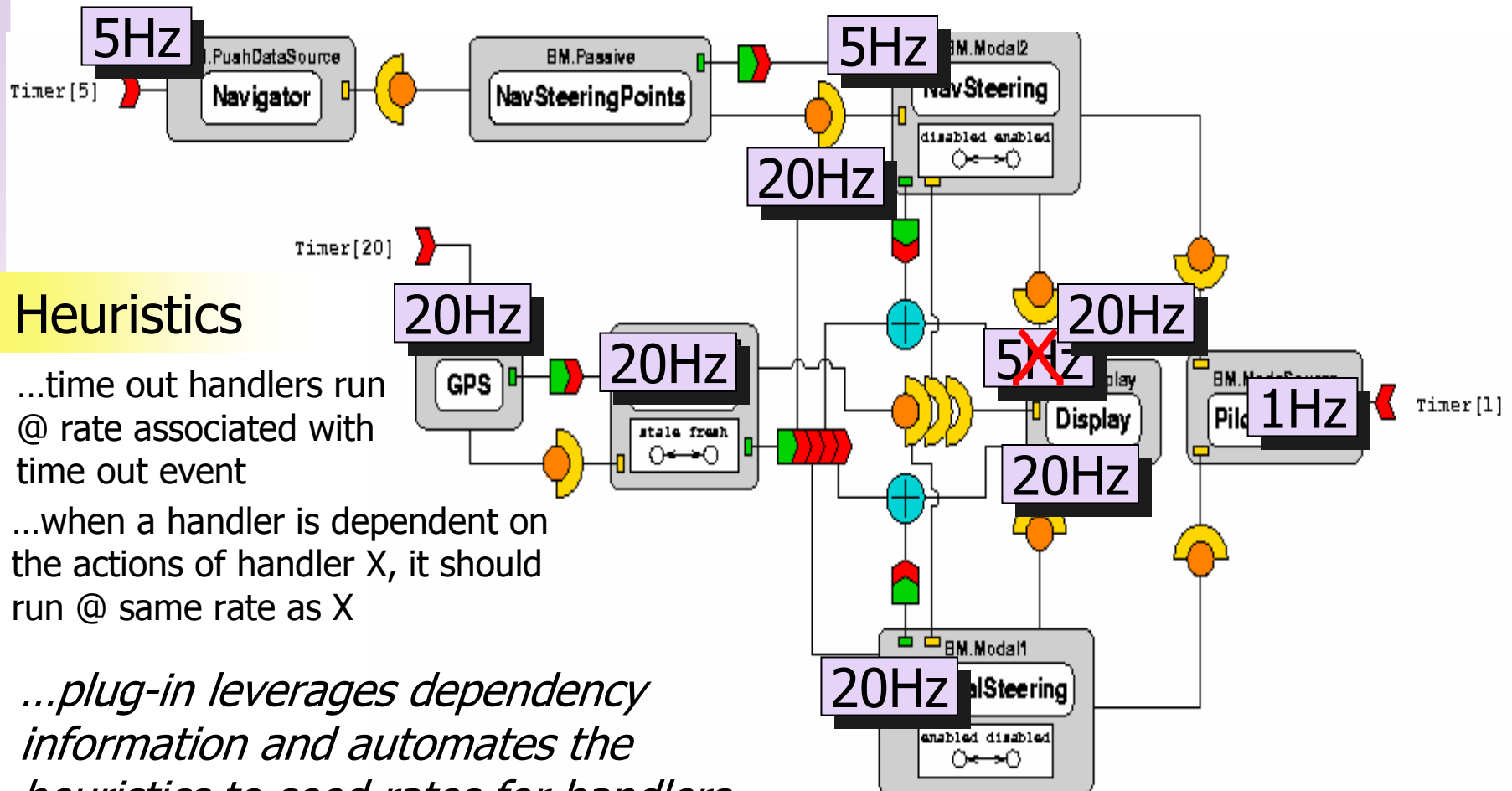
- Bold Stroke product-line architecture contains all threads in RT CORBA event channel
 - rate monotonic scheduling, so a group of thread where each thread has a certain priority/rate
- Each event handler has a model-level attribute to indicate the priority/rate of the thread that should run the handler

developer must select attribute value...



Automated Design Advice

Dependency-driven priority/rate assignment to event handlers



Heuristics

...time out handlers run
@ rate associated with
time out event

...when a handler is dependent on
the actions of handler X, it should
run @ same rate as X

*...plug-in leverages dependency
information and automates the
heuristics to seed rates for handlers*

Spreadsheet View

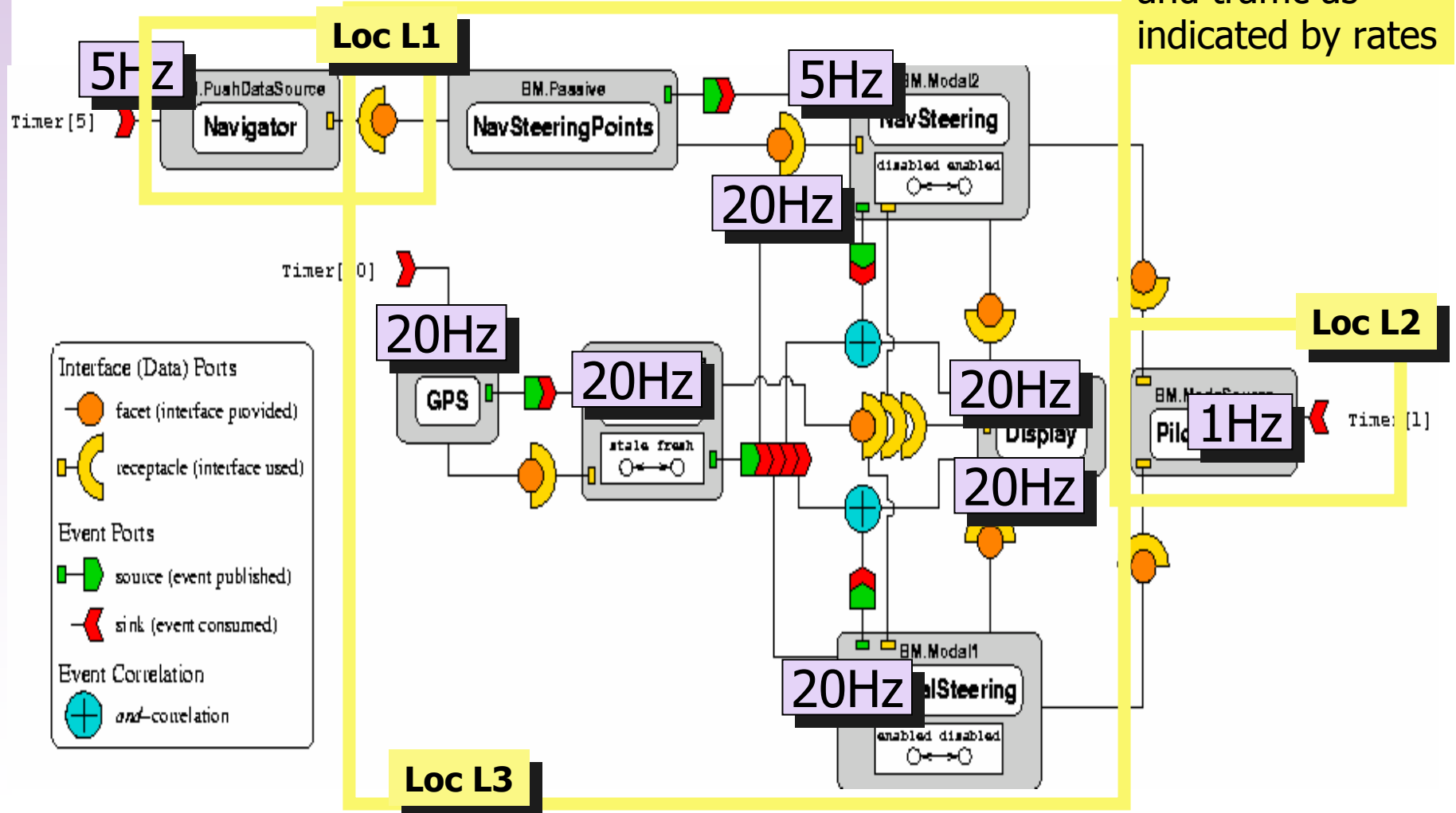
Results of automatic rate group synthesis are fed back into spreadsheet

Component	Rate(Hz)	Connected to	Contained in
Board1	5	> timeOut	Navigator
Board1	1	> timeOut	PilotControl
Board1	20	> timeOut	GPS
Board1	20	< timeOut20	EventChannel
Board1	20	> inDataAvailable	AirFrame
Board1	0	> dataIn	AirFrame
Board2	20	> inDataAvailable	TacticalSteering
Board2	20	> inDataAvailableAirFrame	DisplayCorrelator
Board2	20	> inDataAvailableAirFrame	NavSteering
Board2	20	< outDataAvailable	GPS
Board2	20	< dataOut	GPS
Board2	0	> dataIn	TacticalSteering
Board2	0	> dataIn1	Display
Board2	0	> dataIn2	NavSteering
Board3	20	< outDataAvailable	NavSteering
Board3	20	> inDataAvailable	Display
Board3	20	< outDataAvailable	AirFrame
Board3	20	< outDataAvailable	TacticalSteering
Board3	20	< dataOut	TacticalSteering
Board3	20	< dataOut	NavSteering
Board3	20	< outDataAvailable	DisplayCorrelator
Board3	20	< dataOut	AirFrame
Board4	1	< timeOut1	EventChannel
Board4	1	< modeChange	NavSteering
Board4	1	< modeChange	TacticalSteering
Board5	20	< dataOut	AirFrame
Board5	0	< outDataAvailable	NavSteeringPoints
Board5	20	> inDataAvailableNavStee...	DisplayCorrelator
Board5	20	< outDataAvailable	AirFrame
Board5	0	> modeToggle2	PilotControl
Board5	20	< dataOut	NavSteeringPoints
Board5	0	> dataIn3	Display
Board6	20	> inDataAvailable	DisplayCorrelator
Board6	0	> modeToggle1	PilotControl
Board6	20	< outDataAvailable	AirFrame
Board6	20	< dataOut	AirFrame
Board6	0	> dataIn2	Display
Board7	5	< timeOut5	EventChannel
Board7	5	< dataWriteIn	NavSteeringPoints
Board8	0	> dataWriteOut	Navigator
Board8	0	> inDataAvailableNavStee...	NavSteering
Board8	0	> dataIn1	NavSteering

Automated Design Advice

Synthesis of distribution information

Look at coupling and traffic as indicated by rates

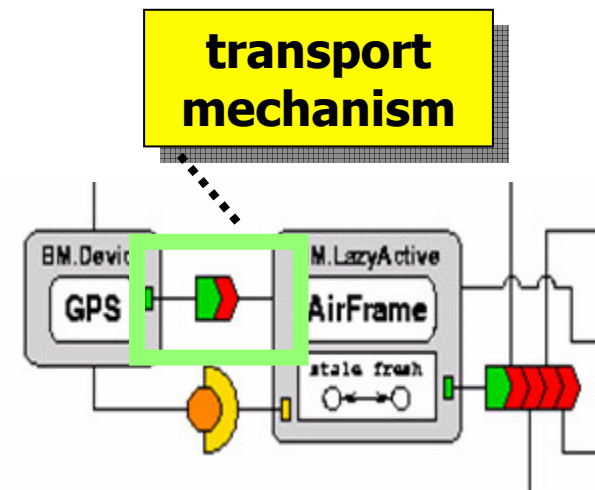


Automated Design Advice

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

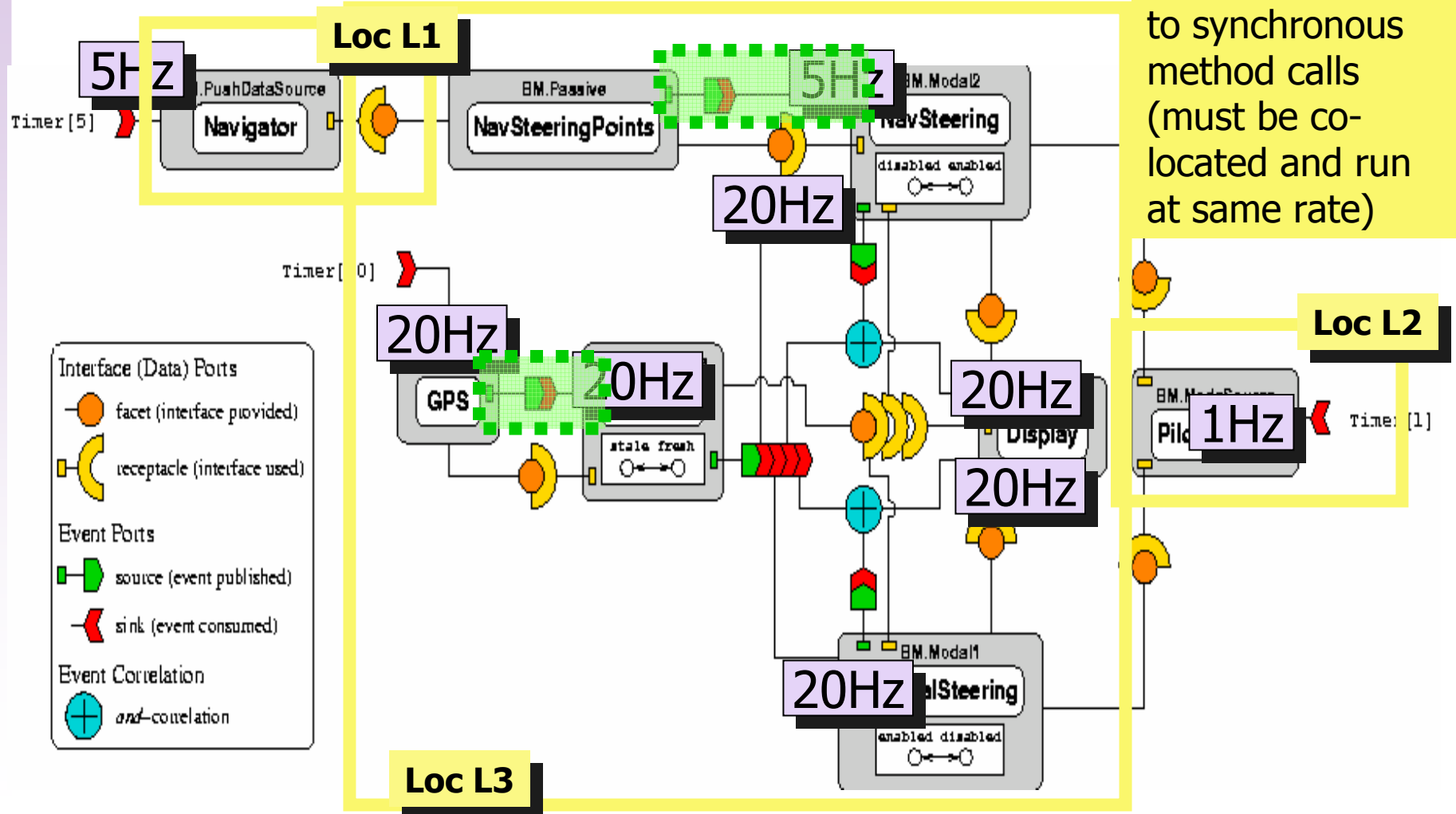
developer must select attribute value...



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

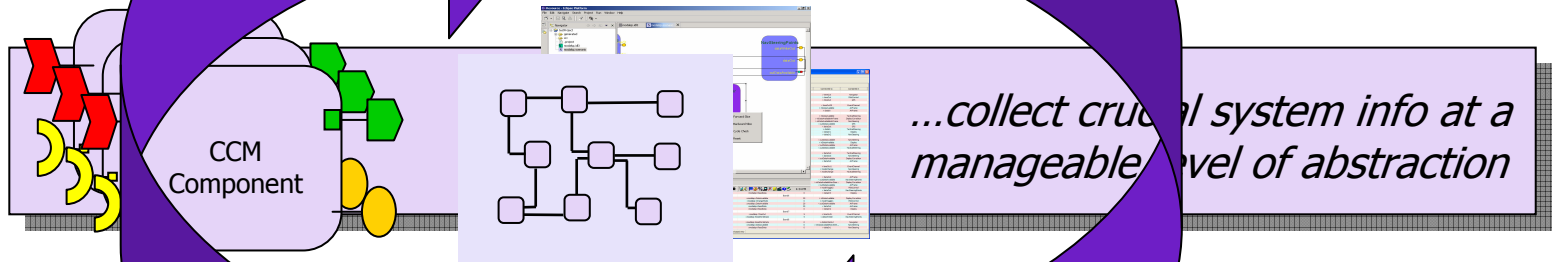
Analyze & Leverage Annotated Models

Automated design advice for middle-tier servers

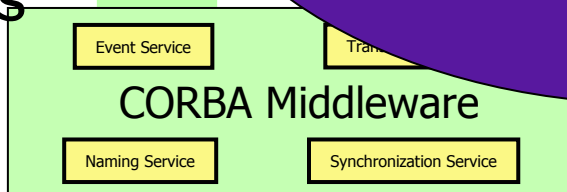
Smart placement of middleware components

Verification of system invariants, crucial event sequencing constraints,

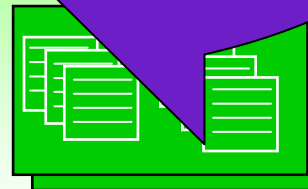
Annotated Models



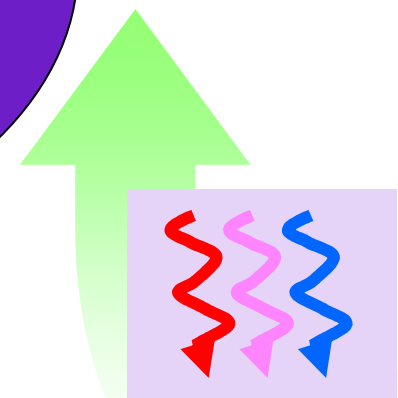
Collect & Expose Attributes



Middleware and service configuration parameters



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.