Verification of Redundant Architectures in AADL

UML & AADL Workshop 2009
June 2, 2009

Dio de Niz & Peter Feiler

Software Engineering Institute | Carnegie Mellon

Problem

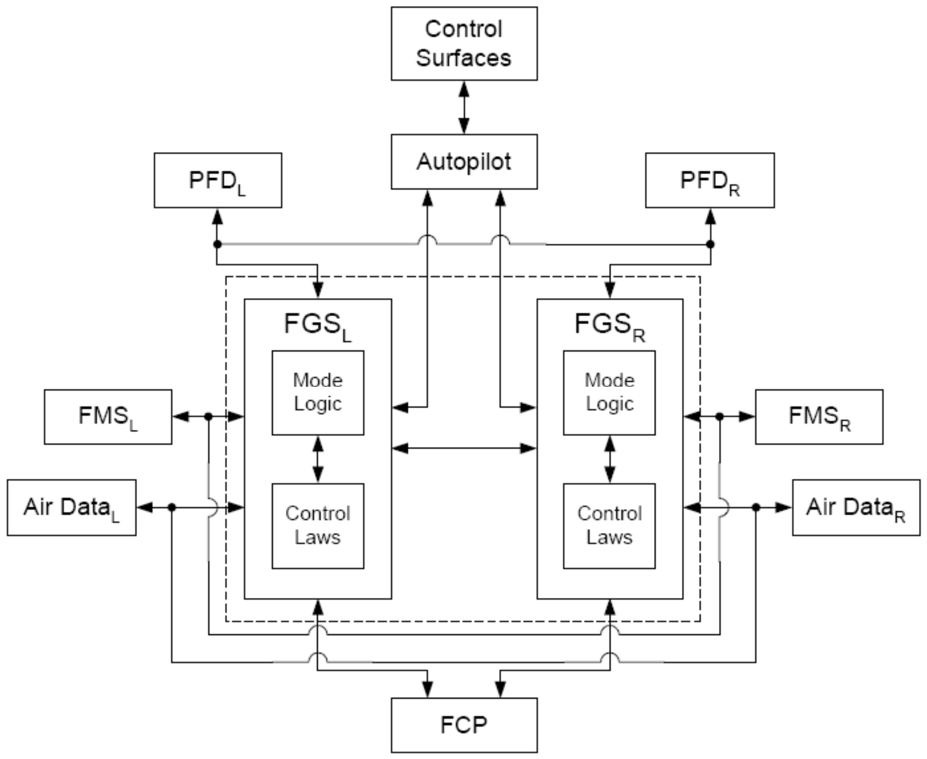
Redundant Architectures Used for Fault-Tolerance, but…

Assume replicas “reactions” to faults/commands are synchronous

- E.g. primary switches to backup when backup to primary

What effects do we want to tolerate

Dual Flight Guidance Systems



Property 1

- At least one FGS shall always be active

Property 2

- Exactly one side shall be the pilot flying side

Property 3

- If the system is in independent mode, both FGS shall be active

NASA/CR-2005-213912. S.P. Miller et al. “A Methodology for the Design and Verification of Globally Asynchronous / Locally Synchronous Architectures

NASA's Incremental Approach

Fully synchronous

Asynchronous

- Mode-switching signal loss: previous active does not switch to inactive
- Fix: model acknowledgement & relaxing properties
    - During mode transition : 2 pilots flying

Asynchronous with failures

- Failure modeled as a signal to the still-active component
- Properties needed to relax to account for “failure discovery” time
- Issues:
    – The introduction of a failure as a signal is unnatural to model

Suggestions on NASA Approach

Use “proven” asynchronous / fault-tolerant constructs

- E.g. ack in a fault-tolerant communication

Better asynchrony modeling to avoid unbounded clocks

- Known period and drift

Modeling of mode synchronization should be part of system architecture and not component

Not include logic of component failure

Use patterns of common architectural features

Architectural Abstraction Approach using AADL

Modality model at architectural level

Evaluate expected synchrony between distributed modes

Develop synchronous “expected” model

Create new distributed architecture

Evaluate all potential failures

From Chaos to Order

Non Functional Properties

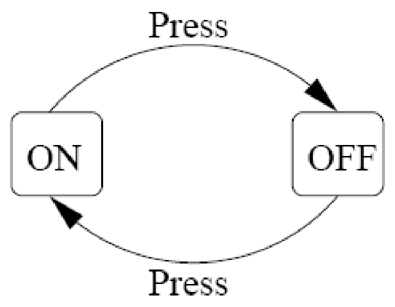
Systems modeled with two chaotic components

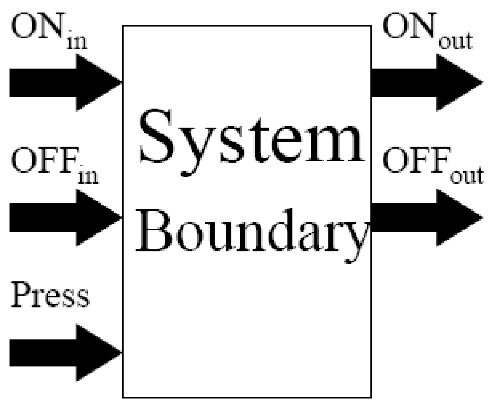
- Execution Platform (Executor)
- Software (Executable)

Model all possible faults

- Executor
  - Faulty execution
  - Execution Reorder
- Executable: fault intolerant

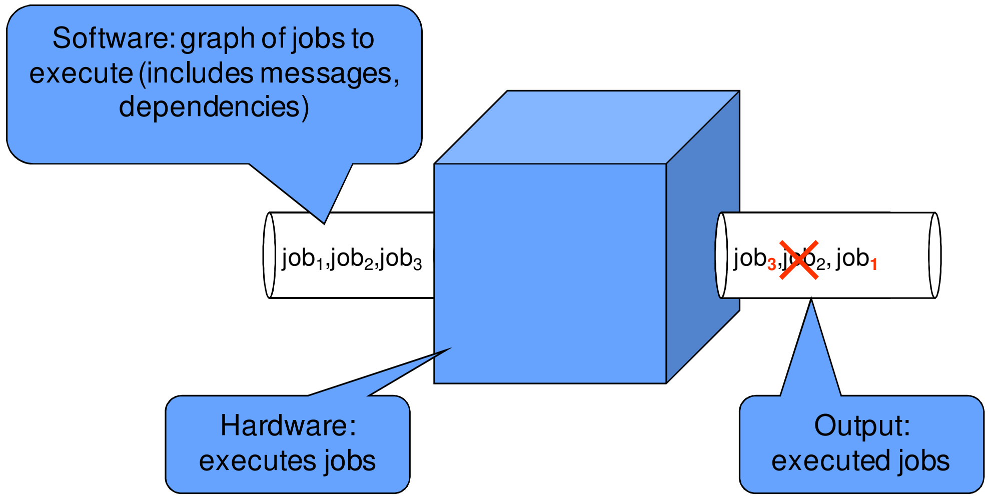
Chaotic Model (Ormeier et al.)





ON_{in}	OFF_{in}	Press	ON_{out}	OFF_{out}
				X
			X	
			X	X
…	…	…	…	…
X	X	X	X	
X	X	X	X	X

Non-Functional


Software: graph of jobs to execute (includes messages, dependencies)
job_1, job_2, job_3
$job_3, \cancel{job_2}, job_1$
Hardware: executes jobs
Output: executed jobs

INPUT			OUTPUT			
t1	t2	t3	t1	t2	t3	t4
job_1	job_2	job_3	job_3	$j\cancel{o}b_2$	job_1	
job_1	job_2	job_3	job_2	$j\cancel{o}b_3$	job_1	
job_1	job_2	job_3	job_1	job_2	job_3	
…	…	…	…	…	…	…

Reducing Behaviors

Hardware (Chaos)

- Rule out impossible behaviors
- Add hardware properties
  - A single processor cannot execute two jobs at a time
- Add software structure
  - E.g. Component does not execute until message delivered

Software (Intolerance)

- Order of execution just need to honor precedence
- Some jobs may be optional
- Some messages can be lost

AADL Model

Default model: all behaviors

Reduce Software Intolerance

```
flows:
  f1: end to end flow t1.p1->t2.p1 {tolerate=>loss;};
```

Reduce Hardware Chaos

- bus: Bus {ensures => no_loss;};

Analysis of Architectural Differences in Alloy

- Base model assumed to be correct
- Modified model that can introduce problems
  - New model with requirements from previous model
- Discover new not tolerated behaviors

Consequences to Modes

Redundant Architecture can mean synchronous modes

- E.g. when node1 in primary, node2 in backup

Loss of transition signal means modes out of sync

Delayed transition means out of sync for a some time

Out-of-sync modes

- Connections not active
- Duplicated connections

Synchronous Model (control flow)

```
thread implementation single.i
 calls
  s1: {
  cf: subprogram controller_function;
  pf: subprogram primary_function ;
  bf: subprogram backup_function;
  p_pbw: subprogram primary_to_backup_waiting;
  b_bpw: subprogram backup_to_primary_waiting;
  p_bpw: subprogram backup_to_primary_waiting;
  b_pbw: subprogram primary_to_backup_waiting;
  auto: subprogram auto_pilot;
  };
end single.i;
```

Asynchronous
Left (Primary) / Right (Backup)

```
thread implementation primary.i
 calls
  s1: {
   pf: subprogram primary_function;
   } in modes (primary_mode, backup_mode);
  s2: {
   p_pbw: subprogram primary_to_backup_waiting;
   } in modes (primary_to_backup_synchronizing);
  s3: {
   p_bpw: subprogram backup_to_primary_waiting;
   } in modes (backup_to_primary_synchronizing);
 connections
  c1: event port p_pbw.switched_mode -> out_switched_mode;
  c2: event port p_bpw.switched_mode -> out_switched_mode;
  c3: event port pf.outNav->outNav in modes (primary_mode);
 modes
  primary_mode initial mode ;
  primary_to_b    kup_synchronizing: mode ;
  backup_to      nary_synchronizing: mode ;
  backup_m       mode ;
  primary_       -[ transfer ]-> primary_to_backup_synchronizing;
  primary        ckup_synchronizing -[ in_switched_mode]-> backup_mode;
  backu          -[ transfer]-> backup_to_primary_synchronizing;
  back           mary_synchronizing -[ in_switched_mode ]-> primary_mode;
end p
```

```
thread implementation backup.i
 calls
  s1: {
   bf: subprogram backup_function;
   } in modes (backup_mode, primary_mode);
  s2: {
   b_bpw: subprogram backup_to_primary_waiting;
   } in modes (backup_to_primary_synchronizing);
  s3: {
   b_pbw: subprogram primary_to_backup_waiting;
   } in modes (primary_to_backup_synchronizing);
 connections
  c1: event port b_bpw.switched_mode -> out_switched_mode;
  c2: event port b_pbw.switched_mode -> out_switched_mode;
  c3: event port bf.outNav -> outNav in modes (backup_mode);
 modes
  primary_mode: mode ;
  primary_to_backup_synchronizing: mode ;
  backup_to_primary_synchronizing: mode ;
  backup_mode: initial mode ;
  backup_mode -[ transfer]-> backup_to_primary_synchronizing;
  backup_to_primary_synchronizing -[ in_switched_mode ]-> primary_mode;
  primary_mode -[ transfer]-> primary_to_backup_synchronizing;
  primary_to_backup_synchronizing -[in_switched_mode ]-> backup_mode;
end backup.i;
```

Modal connections

Mode transition triggers

AADL Annotations

Hardware

```
system implementation final.i
 subcomponents
  cpu1: processor cpu {chaotic::Lossless => true;};
  cpu2: processor cpu {chaotic::Lossless => true;};
  cpu3: processor cpu {chaotic::Lossless => true;};
  cpu4: processor cpu {chaotic::Lossless => true;};
  crnet1: bus net {chaotic::Lossless => true;};
  crnet2: bus net {chaotic::Lossless => true;};
  crnet3: bus net {chaotic::Lossless => true;};
  crnet4: bus net {chaotic::Lossless => true;};
  crnet5: bus net {chaotic::Lossless => true;};
```

Software

```
connections
  c1: event port control.transfer ->
                 primary_sw.transfer
                {chaotic::InOrder => true;};
 c2: event port control.transfer ->
                 backup_sw.transfer
                {chaotic::ReorderTolerant => true;
                 chaotic::LossTolerant => true;};
 c3: event port primary_sw.out_switched_mode->
                backup_sw.in_switched_mode
           {chaotic::ReorderTolerance => 10 ms;};
  c4: event port backup_sw.out_switched_mode ->
                 primary_sw.in_switched_mode
                 {chaotic::InOrder => true;};
  c5: event port backup_sw.outNav->
                 auto_sw.inNav
                  {chaotic::InOrder => true;};
  c6: event port primary_sw.outNav->
                 auto_sw.inNav
                 {chaotic::InOrder => true;};
```

Mode Transition Loss

Built-in acknowledge of mode transition

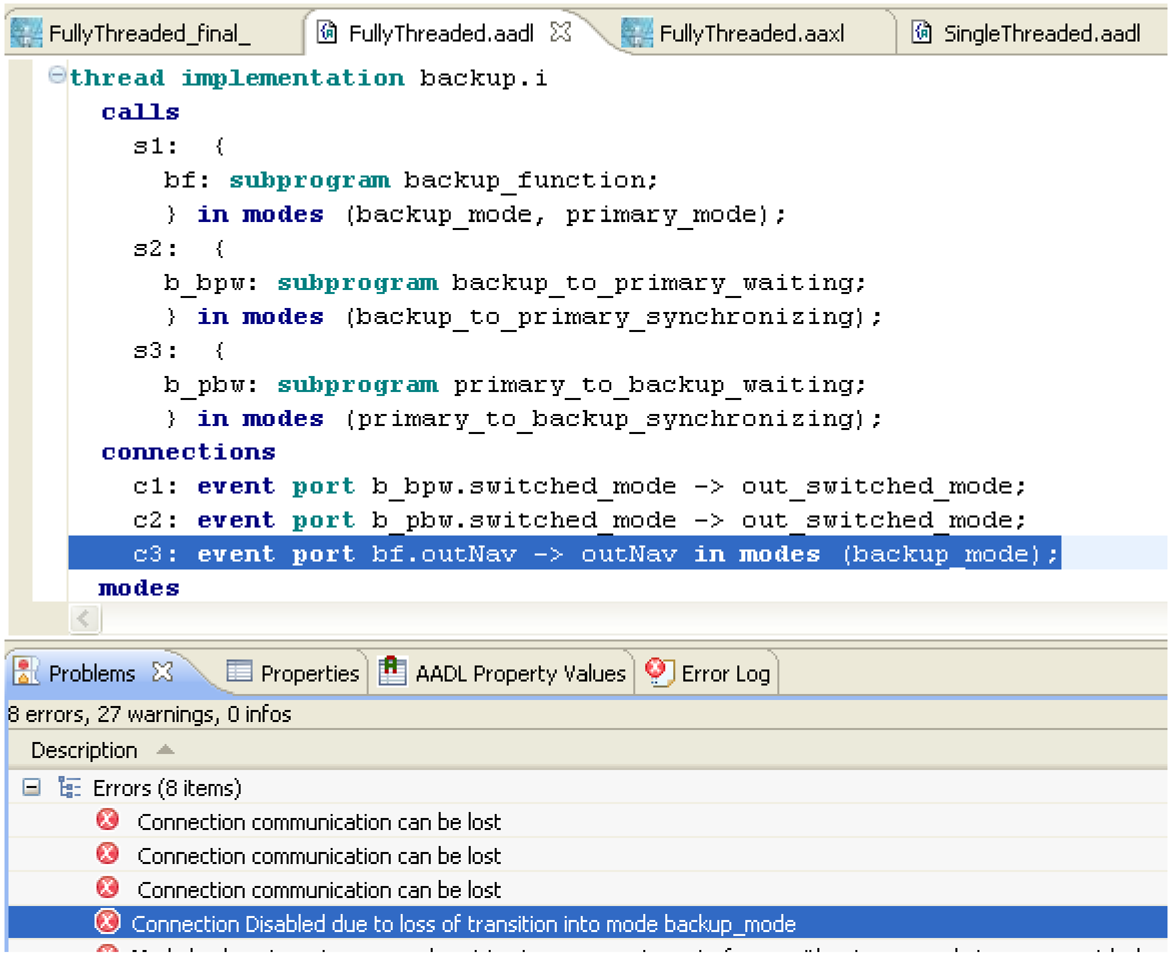
Mode loss due to network message loss

Automatically discovered “out-of-sync” modes

Automatically discovered connection loss due to “inactive” mode

- No output to autopilot

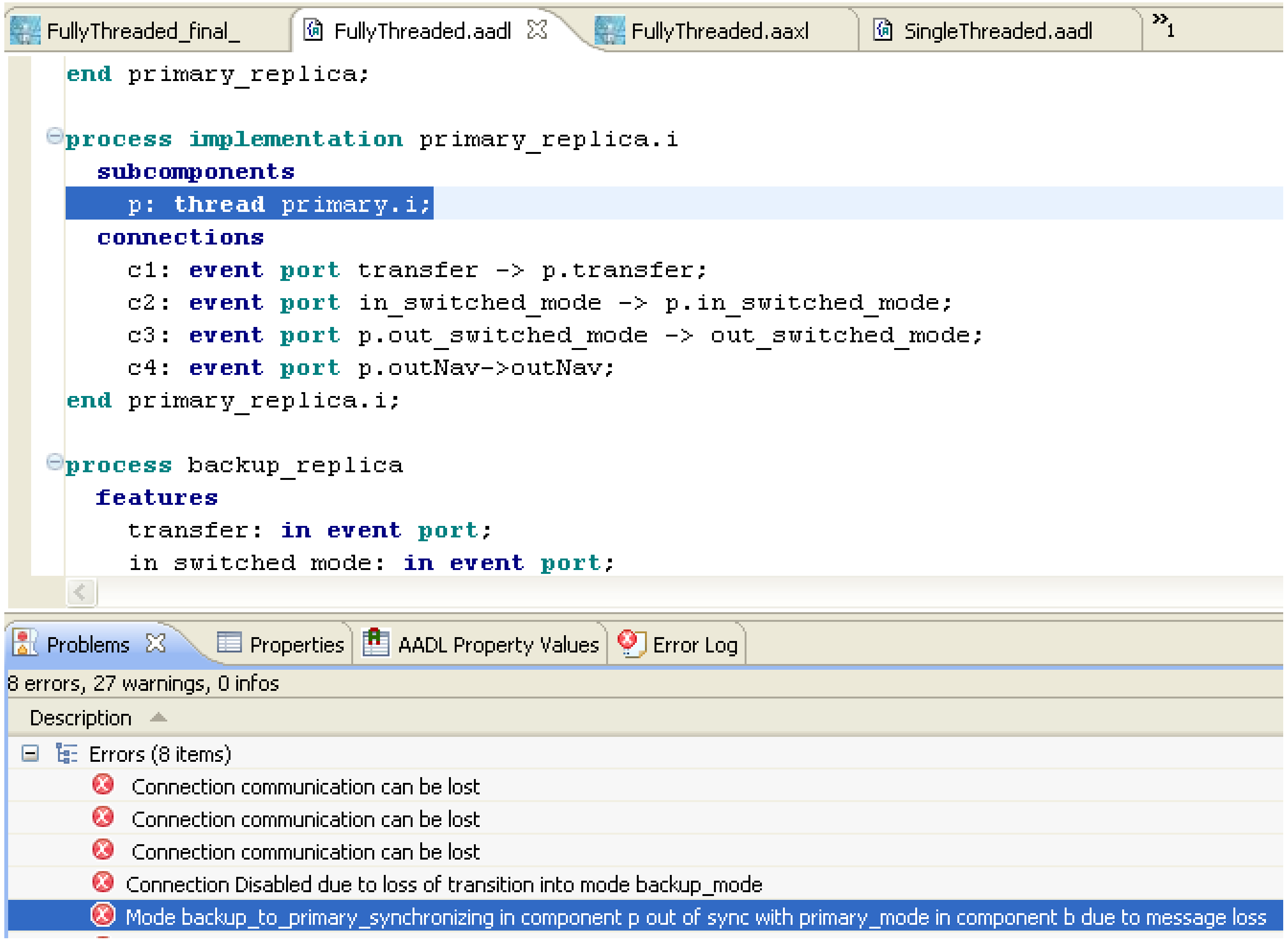
Loss of communication due to mode transition failure



```
thread implementation backup.i
  calls
    s1:  {
      bf: subprogram backup_function;
      } in modes (backup_mode, primary_mode);
    s2:  {
      b_bpw: subprogram backup_to_primary_waiting;
      } in modes (backup_to_primary_synchronizing);
    s3:  {
      b_pbw: subprogram primary_to_backup_waiting;
      } in modes (primary_to_backup_synchronizing);
  connections
    c1: event port b_bpw.switched_mode -> out_switched_mode;
    c2: event port b_pbw.switched_mode -> out_switched_mode;
    c3: event port bf.outNav -> outNav in modes (backup_mode);
  modes
```

```
8 errors, 27 warnings, 0 infos
Description
Errors (8 items)
    Connection communication can be lost
    Connection communication can be lost
    Connection communication can be lost
    Connection Disabled due to loss of transition into mode backup_mode
```

Out-of-sync modes



```
end primary_replica;

process implementation primary_replica.i
  subcomponents
    p: thread primary.i;
  connections
    c1: event port transfer -> p.transfer;
    c2: event port in_switched_mode -> p.in_switched_mode;
    c3: event port p.out_switched_mode -> out_switched_mode;
    c4: event port p.outNav->outNav;
end primary_replica.i;

process backup_replica
  features
    transfer: in event port;
    in switched mode: in event port;
```

```
8 errors, 27 warnings, 0 infos
Errors (8 items)
    Connection communication can be lost
    Connection communication can be lost
    Connection communication can be lost
    Connection Disabled due to loss of transition into mode backup_mode
    Mode backup_to_primary_synchronizing in component p out of sync with primary_mode in component b due to message loss
```

Quantifying out-of-sync errors

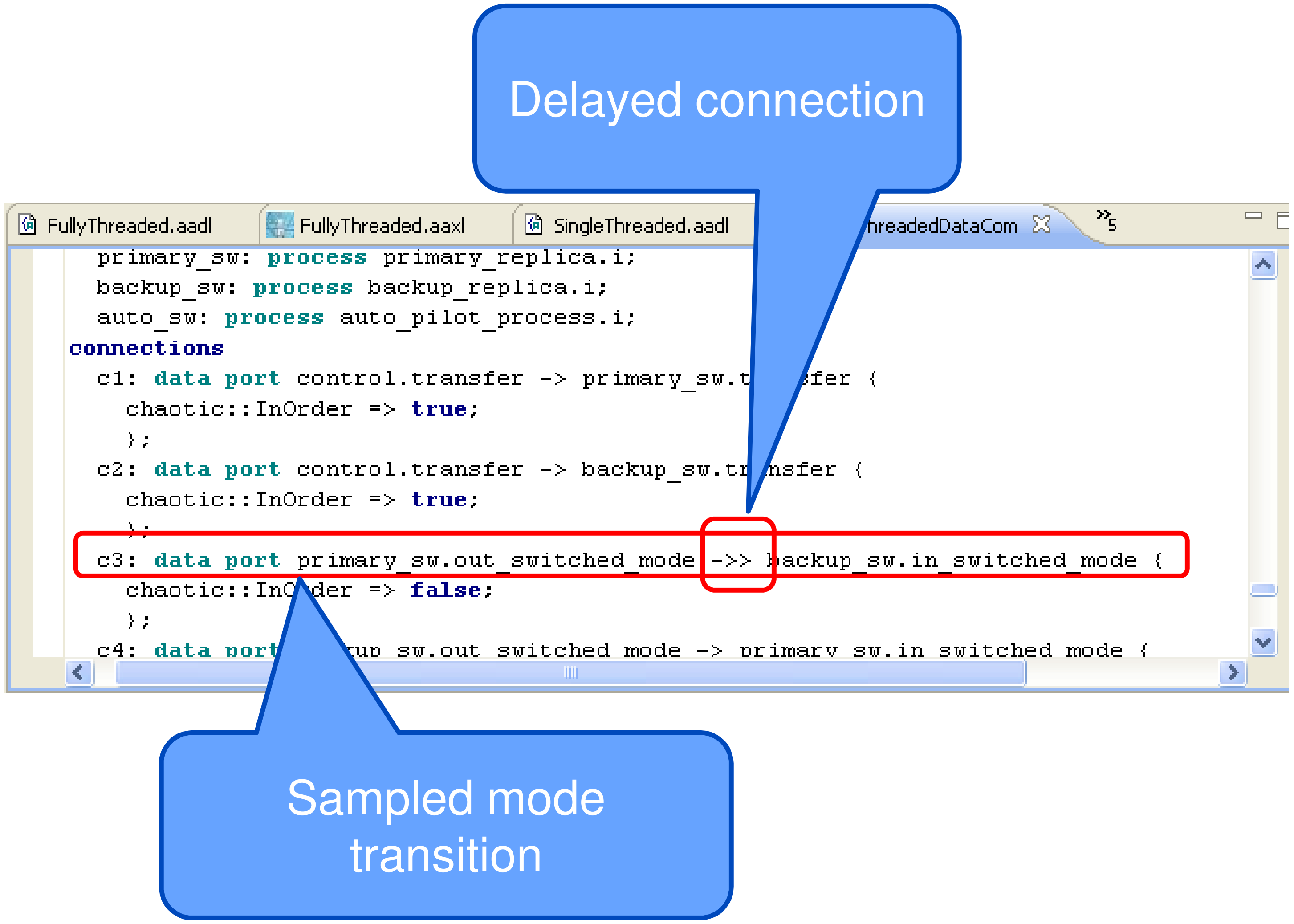
Separate loss from out of sync

Out of sync modes happens due out of sync communication/execution

- Sampled communication

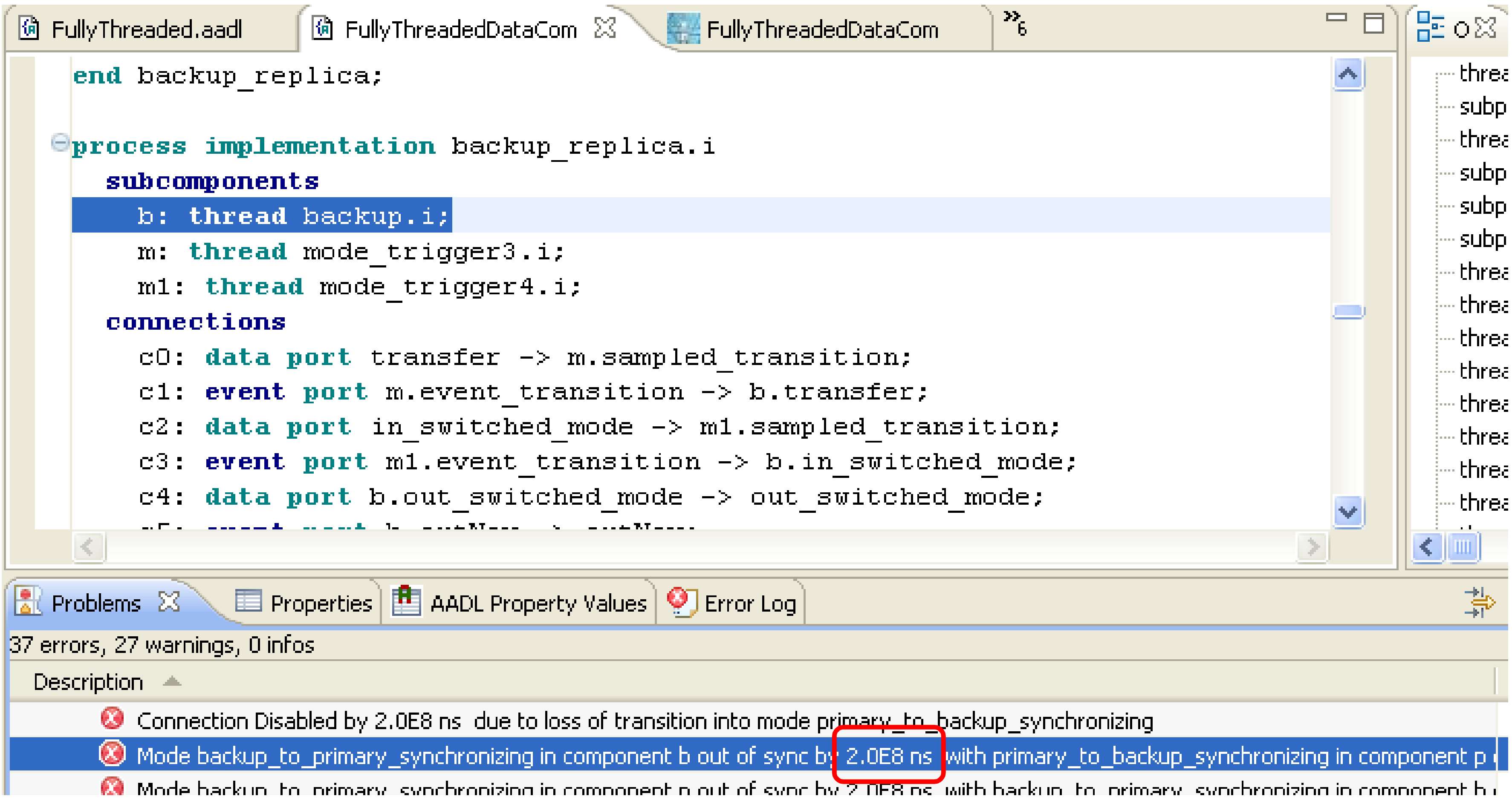
Modeled in AADL as sampled data communication

Sampled Data Communication



```
  primary_sw: process primary_replica.i;
  backup_sw: process backup_replica.i;
  auto_sw: process auto_pilot_process.i;
connections
  c1: data port control.transfer -> primary_sw.t    sfer {
    chaotic::InOrder => true;
    };
  c2: data port control.transfer -> backup_sw.tr nsfer {
    chaotic::InOrder => true;
    };
  c3: data port primary_sw.out_switched_mode ->> backup_sw.in_switched_mode {
    chaotic::InOrder => false;
    };
  c4: data port     kup_sw.out_switched_mode -> primary_sw.in_switched_mode {
```

Quantified Out-of-sync Modes



```
end backup_replica;

process implementation backup_replica.i
  subcomponents
    b: thread backup.i;
    m: thread mode_trigger3.i;
    m1: thread mode_trigger4.i;
  connections
    c0: data port transfer -> m.sampled_transition;
    c1: event port m.event_transition -> b.transfer;
    c2: data port in_switched_mode -> m1.sampled_transition;
    c3: event port m1.event_transition -> b.in_switched_mode;
    c4: data port b.out_switched_mode -> out_switched_mode;
```

```
37 errors, 27 warnings, 0 infos
Connection Disabled by 2.0E8 ns  due to loss of transition into mode primary_to_backup_synchronizing
Mode backup_to_primary_synchronizing in component b out of sync by 2.0E8 ns with primary_to_backup_synchronizing in component p
Mode backup_to_primary_synchronizing in component p out of sync by 2.0E8 ns with backup_to_primary_synchronizing in component b
```

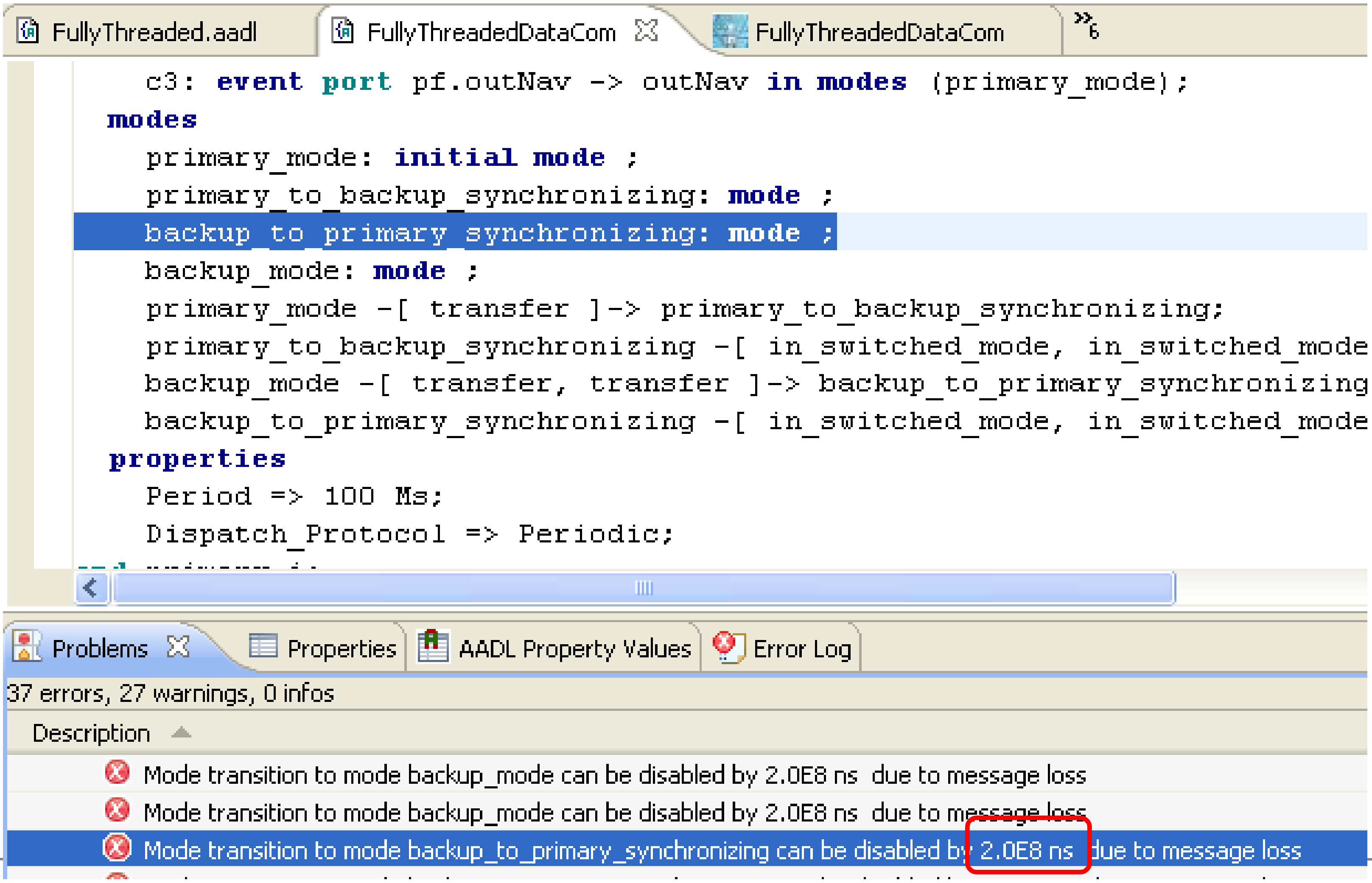
Mode Transition Disable Period

Quantified delay (instead of “communication step”)

Bounded due to periodicity of threads

Precise worst-case calculation of communication interruption / duplication

Mode Transition Delay



FullyThreaded.aadl | FullyThreadedDataCom | FullyThreadedDataCom

```
    c3: event port pf.outNav -> outNav in modes (primary_mode);
  modes
    primary_mode: initial mode ;
    primary_to_backup_synchronizing: mode ;
    backup_to_primary_synchronizing: mode ;
    backup_mode: mode ;
    primary_mode -[ transfer ]-> primary_to_backup_synchronizing;
    primary_to_backup_synchronizing -[ in_switched_mode, in_switched_mode
    backup_mode -[ transfer, transfer ]-> backup_to_primary_synchronizing
    backup_to_primary_synchronizing -[ in_switched_mode, in_switched_mode
  properties
    Period => 100 Ms;
    Dispatch_Protocol => Periodic;
```

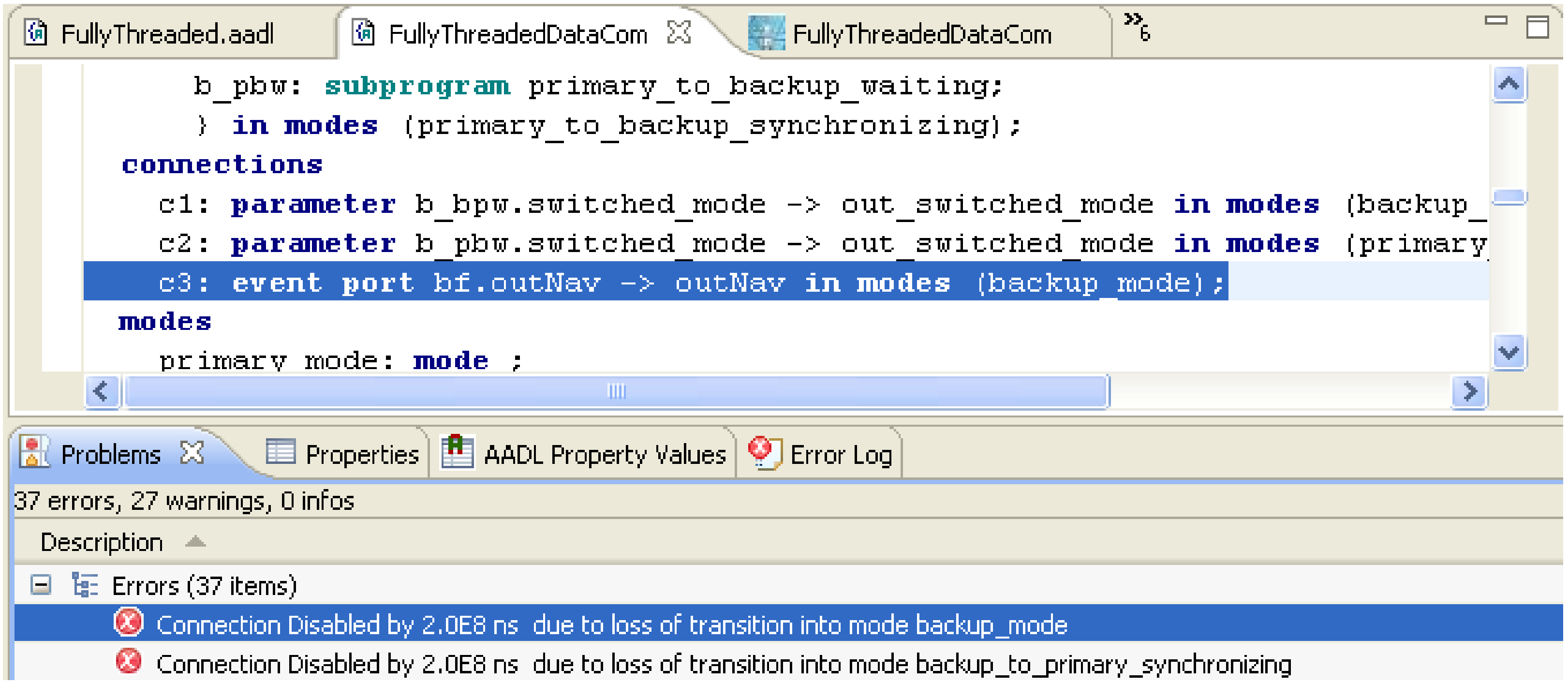
Problems | Properties | AADL Property Values | Error Log

37 errors, 27 warnings, 0 infos

Description

```
Mode transition to mode backup_mode can be disabled by 2.0E8 ns  due to message loss
Mode transition to mode backup_mode can be disabled by 2.0E8 ns  due to message loss
Mode transition to mode backup_to_primary_synchronizing can be disabled by 2.0E8 ns  due to message loss
```

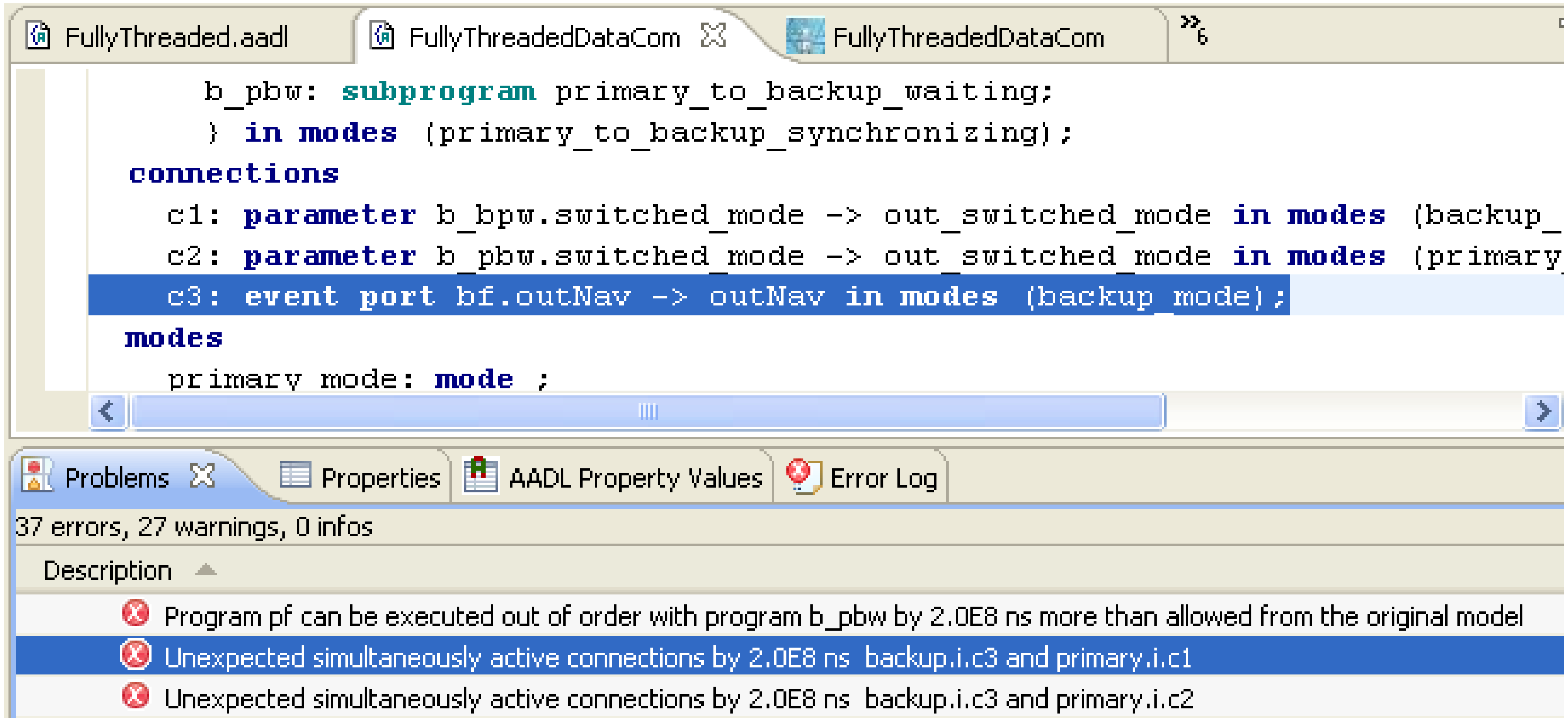
Worst-Case Communication Interruptions



```
      b_pbw: subprogram primary_to_backup_waiting;
      } in modes (primary_to_backup_synchronizing);
  connections
    c1: parameter b_bpw.switched_mode -> out_switched_mode in modes (backup_
    c2: parameter b_pbw.switched_mode -> out_switched_mode in modes (primary
    c3: event port bf.outNav -> outNav in modes (backup_mode);
  modes
    primary_mode: mode ;
```

```
37 errors, 27 warnings, 0 infos
Description
Errors (37 items)
    Connection Disabled by 2.0E8 ns  due to loss of transition into mode backup_mode
    Connection Disabled by 2.0E8 ns  due to loss of transition into mode backup_to_primary_synchronizing
```

Worst-Case Communication Duplication



```
      b_pbw: subprogram primary_to_backup_waiting;
      } in modes (primary_to_backup_synchronizing);
  connections
    c1: parameter b_bpw.switched_mode -> out_switched_mode in modes (backup_
    c2: parameter b_pbw.switched_mode -> out_switched_mode in modes (primary
    c3: event port bf.outNav -> outNav in modes (backup_mode);
  modes
    primary_mode: mode ;
```

```
37 errors, 27 warnings, 0 infos
Description
Program pf can be executed out of order with program b_pbw by 2.0E8 ns more than allowed from the original model
Unexpected simultaneously active connections by 2.0E8 ns  backup.i.c3 and primary.i.c1
Unexpected simultaneously active connections by 2.0E8 ns  backup.i.c3 and primary.i.c2
```

Concluding Remarks

Analysis of Concurrency in AADL model leverages semantics of AADL

- Processor bindings, failing processors,
- Duration of errors

Keeps analysis at architectural level

Focuses on problems introduced by the runtime architecture